Решения и методики построения запросов для разработчиков баз данных

Включает решения для SQL Server, PostgreSQL, Oracle, MySQL и DB2

SQL

Сборник рецептов

СИМВОЛ®

O'REILLY®

Энтони Молинаро

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-125-8, название «SQL. Сборник рецептов» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.

SQL Cookbook

Anthony Molinaro

O'REILLY®

SQL
Сборник рецептов

Энтони Молинаро

Санкт-Петербург — Москва
2009

Энтони Молинаро

SQL. Сборник рецептов

Перевод Н. Шатохиной

Главный редактор	*А. Галунов*
Зав. редакцией	*Н. Макарова*
Выпускающий редактор	*П. Щеголев*
Научный редактор	*К. Козинский*
Редактор	*О. Меркулова*
Корректор	*С. Минин*
Верстка	*Д. Орлова*

Молинаро Э.
SQL. Сборник рецептов. – Пер. с англ. – СПб: Символ-Плюс, 2009. – 672 с., ил.
ISBN-13: 978-5-93286-125-7
ISBN-10: 5-93286-125-8

Книга Энтони Молинаро «SQL. Сборник рецептов» предназначена тем, кто уже знаком с основами языка запросов SQL и хочет повысить свой профессиональный уровень. Она будет полезна и экспертам SQL, поскольку автор предлагает варианты решения задач для разных СУБД: DB2, Oracle, PostgreSQL, MySQL и SQL Server. Если вы постоянно работаете с SQL на одной платформе, то, возможно, найдете в рецептах более эффективное решение на другой. Вы научитесь использовать SQL для решения более широкого спектра задач – от операций внутри баз данных до передачи данных по сети в приложения. Для этого достаточно открыть книгу на странице с интересующим вас рецептом.

Вы узнаете, как применять оконные функции и специальные операторы, а также расширенные методы работы с хранилищами данных: создание гистограмм, резюмирование данных в блоки, выполнение агрегации скользящего диапазона значений, формирование текущих сумм и подсумм. Вы сможете разворачивать строки в столбцы и наоборот, упрощать вычисления внутри строки и выполнять двойное разворачивание результирующего множества, выполнять обход строки, что позволяет использовать SQL для синтаксического разбора строки на символы, слова или элементы строки с разделителями. Приемы, предлагаемые автором, позволят оптимизировать код ваших приложений и откроют перед вами новые возможности языка SQL.

ISBN-13: 978-5-93286-125-7
ISBN-10: 5-93286-125-8
ISBN 0-596-00976-3 (англ)

© Издательство Символ-Плюс, 2009
Authorized translation of the English edition © 2006 O'Reilly Media, Inc. This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

Все права на данное издание защищены Законодательством РФ, включая право на полное или частичное воспроизведение в любой форме. Все товарные знаки или зарегистрированные товарные знаки, упоминаемые в настоящем издании, являются собственностью соответствующих фирм.

Издательство «Символ-Плюс». 199034, Санкт-Петербург, 16 линия, 7,
тел. (812) 324-5353, www.symbol.ru. Лицензия ЛП N 000054 от 25.12.98.

Подписано в печать 29.09.2008. Формат 70×100¹/16. Печать офсетная.
Объем 42 печ. л. Тираж 2500 экз. Заказ №
Отпечатано с готовых диапозитивов в ГУП «Типография «Наука»
199034, Санкт-Петербург, 9 линия, 12.

Моей маме:
Ты лучшая! Спасибо тебе за все.

Оглавление

Предисловие 13

1. Извлечение записей 33

Извлечение всех строк и столбцов из таблицы 33
Извлечение подмножества строк из таблицы 34
Выбор строк по нескольким условиям 34
Извлечение подмножества столбцов из таблицы 35
Как задать столбцам значимые имена 36
Обращение к столбцу в предикате WHERE по псевдониму 37
Конкатенация значений столбцов 38
Использование условной логики в выражении SELECT 39
Ограничение числа возвращаемых строк 40
Возвращение *n* случайных записей таблицы 42
Поиск значений NULL 43
Преобразование значений NULL в не-NULL значения 44
Поиск по шаблону 45

2. Сортировка результатов запроса 47

Возвращение результатов запроса в заданном порядке 47
Сортировка по нескольким полям 48
Сортировка по подстрокам 49
Сортировка смешанных буквенно-цифровых данных 50
Обработка значений NULL при сортировке 53
Сортировка по зависящему от данных ключу 60

3. Работа с несколькими таблицами 62

Размещение одного набора строк под другим 62
Объединение взаимосвязанных строк 64
Поиск одинаковых строк в двух таблицах 66
Извлечение из одной таблицы значений,
которых нет в другой таблице 68
Извлечение из таблицы строк, для которых
нет соответствия в другой таблице 72

Независимое добавление объединений в запрос 74
Выявление одинаковых данных в двух таблицах 76
Идентификация и устранение некорректного использования декартова произведения 83
Осуществление объединений при использовании агрегатных функций 85
Внешнее объединение при использовании агрегатных функций 90
Возвращение отсутствующих данных из нескольких таблиц 93
Значения NULL в операциях и сравнениях 97

4. Вставка, обновление, удаление 99

Вставка новой записи 100
Вставка значений по умолчанию 100
Переопределение значения по умолчанию значением NULL 102
Копирование строк из одной таблицы в другую 103
Копирование описания таблицы 103
Вставка в несколько таблиц одновременно 104
Блокировка вставки в определенные столбцы 106
Изменение записей в таблице 107
Обновление в случае существования соответствующих строк в другой таблице 109
Обновление значениями из другой таблицы 110
Слияние записей 113
Удаление всех записей из таблицы 115
Удаление определенных записей 116
Удаление одной записи 116
Удаление записей, которые нарушают ссылочную целостность 117
Уничтожение дублирующихся записей 117
Удаление записей, на которые есть ссылки в другой таблице 119

5. Запросы на получение метаданных 121

Как получить список таблиц схемы 121
Как получить список столбцов таблицы 122
Как получить список индексированных столбцов таблицы 123
Как получить список ограничений, наложенных на таблицу 125
Как получить список внешних ключей без соответствующих индексов 126
Использование SQL для генерирования SQL 130
Описание представлений словаря данных в базе данных Oracle 132

6. Работа со строками 134

Проход строки 135
Как вставить кавычки в строковые литералы 137
Как подсчитать, сколько раз символ встречается в строке 138

Удаление из строки ненужных символов 139
Разделение числовых и символьных данных 141
Как определить, содержит ли строка только буквенно-цифровые данные 145
Извлечение инициалов из имени 150
Упорядочивание по частям строки 154
Упорядочивание по числу в строке 155
Создание списка с разделителями из строк таблицы 161
Преобразование данных с разделителями в список оператора IN со множеством значений 168
Упорядочение строки в алфавитном порядке 174
Выявление строк, которые могут быть интерпретированы как числа 180
Извлечение *n*-ной подстроки с разделителями 187
Синтаксический разбор IP-адреса 194

7. Работа с числами 197

Вычисление среднего 197
Поиск минимального/максимального значения столбца 199
Вычисление суммы значений столбца 201
Подсчет строк в таблице 203
Подсчет значений в столбце 205
Вычисление текущей суммы 205
Вычисление текущего произведения 209
Вычисление текущей разности 212
Вычисление моды 213
Вычисление медианы 216
Вычисление доли от целого в процентном выражении 220
Агрегация столбцов, которые могут содержать NULL-значения 223
Вычисление среднего без учета наибольшего и наименьшего значений 224
Преобразование буквенно-цифровых строк в числа 226
Изменение значений в текущей сумме 228

8. Арифметика дат 231

Добавление и вычитание дней, месяцев и лет 231
Определение количества дней между двумя датами 234
Определение количества рабочих дней между двумя датами 236
Определение количества месяцев или лет между двумя датами 241
Определение количества секунд, минут или часов между двумя датами 244
Как подсчитать, сколько раз в году повторяется каждый день недели 246

Определение интервала времени в днях между текущей и следующей записями . . . 258

9. Работа с датами . . . 264

Как определить, является ли год високосным . . . 265
Как определить количество дней в году . . . 272
Извлечение единиц времени из даты . . . 275
Определение первого и последнего дней месяца . . . 277
Выбор всех дат года, выпадающих на определенный день недели . . . 280
Определение дат первого и последнего появления заданного дня недели . . . 287
Создание календаря . . . 295
Получение дат начала и конца кварталов года . . . 314
Определение дат начала и окончания заданного квартала . . . 320
Дополнение отсутствующих дат . . . 327
Поиск по заданным единицам времени . . . 337
Сравнение строк по определенной части даты . . . 339
Выявление наложений диапазонов дат . . . 342

10. Работа с диапазонами данных . . . 348

Поиск диапазона последовательных значений . . . 348
Вычисление разности между значениями строк одной группы или сегмента . . . 354
Определение начала и конца диапазона последовательных значений . . . 363
Вставка пропущенных значений диапазона . . . 368
Формирование последовательности числовых значений . . . 373

11. Расширенный поиск . . . 378

Разбиение результирующего множества на страницы . . . 378
Как пропустить *n* строк таблицы . . . 381
Использование логики OR во внешних объединениях . . . 384
Выявление строк со взаимообратными значениями . . . 387
Как выбрать записи с *n*-ым количеством наивысших значений . . . 389
Как найти записи с наибольшим и наименьшим значениями . . . 391
Сбор информации из последующих строк . . . 393
Смещение значений строк . . . 396
Ранжирование результатов . . . 400
Исключение дубликатов . . . 401
Ход конем . . . 403
Формирование простых прогнозов . . . 411

12. Составление отчетов и управление хранилищами данных 420

Разворачивание результирующего множества в одну строку 420
Разворачивание результирующего множества в несколько строк . . . 423
Обратное разворачивание результирующего множества 431
Обратное разворачивание результирующего множества в один столбец 433
Исключение повторяющихся значений из результирующего множества 436
Разворачивание результирующего множества для упрощения вычислений 440
Создание блоков данных фиксированного размера 441
Создание заданного количества блоков 445
Создание горизонтальных гистограмм 451
Создание вертикальных гистограмм 453
Как возвратить столбцы, не перечисленные в операторе GROUP BY 456
Вычисление простых подсумм 462
Вычисление подсумм для всех возможных сочетаний 466
Как выявить строки, в которых представлены не подсуммы 476
Использование выражений CASE для маркировки строк 478
Создание разреженной матрицы 480
Группировка строк по интервалам времени 481
Агрегация разных групп/сегментов одновременно 485
Агрегация скользящего множества значений 487
Разворачивание результирующего множества, содержащего подсуммы 495

13. Иерархические запросы 500

Представление отношений родитель-потомок 501
Представление отношений потомок-родитель-прародитель 505
Создание иерархического представления таблицы 510
Выбор всех дочерних строк для заданной строки 519
Определение узлов: ветвления, концевого, корневого 523

14. Всякая всячина 532

Создание отчетов с перекрестными ссылками с помощью оператора SQL Server PIVOT 532
Обратное разворачивание отчета с помощью оператора SQL Server UNPIVOT 534
Транспонирование результирующего множества с помощью оператора Oracle MODEL 536
Извлечение элементов строки, положение которых в строке неизвестно 541

Как определить количество дней в году (альтернативное решение для Oracle) 543
Поиск смешанных буквенно-цифровых строк 545
Преобразование целых чисел в их двоичное представление с использованием Oracle 547
Разворачивание ранжированного результирующего множества 550
Как добавить заголовок столбца в дважды развернутое результирующее множество 555
Преобразование скалярного подзапроса в составной подзапрос (Oracle) 566
Синтаксический разбор сериализованных данных в строки таблицы 569
Определение доли от целого в процентном выражении 573
Создание списка разделенных запятыми значений в Oracle 575
Выбор текста, не соответствующего шаблону (Oracle) 581
Преобразование данных с помощью вложенного запроса 584
Проверка существования значения в группе 585

A. Оконные функции, краткий обзор 589

B. Вспоминаем Розенштейна 617

Алфавитный указатель 655

Предисловие

SQL – язык мира баз данных. Приложения для работы с реляционными базами данных занимаются размещением и извлечением данных из базы, что в конечном счете является вопросом знания SQL. Большинство разработчиков используют SQL лишь формально и даже не подозревают о той мощи, которая имеется в их распоряжении. Цель данной книги – изменить существующее положение дел, открыв глаза пользователей на реальные возможности SQL.

Книга, которую вы держите в руках, является сборником рецептов. Это собрание обычных задач для SQL и их решений, которые, надеюсь, пригодятся в каждодневной работе с базами данных. Рецепты рассортированы по главам соответственно темам. Столкнувшись с новой задачей, выберите наиболее подходящую, по вашему мнению, главу, просмотрите названия рецептов, и, будем надеяться, вы найдете решение или по крайней мере дельный совет.

На страницах данной книги размещено свыше 150 рецептов. И это только малая часть того, что можно сделать с помощью SQL. Число различных решений, предлагаемых SQL для задач, с которыми мы сталкиваемся ежедневно, может затмить лишь число возникающих проблем. Вы не найдете здесь ответы на все вопросы. Все задачи просто невозможно охватить. Однако в книге рассмотрены многие часто встречающиеся задачи и их решения. А используемые в приводимых решениях техники вы научитесь распространять и на другие, новые проблемы, которые я не могу себе даже представить.

Мы с моим издателем постоянно ищем новые SQL-решения, достойные сборника рецептов. Если вам встретится талантливое и интересное решение, поделитесь им, пришлите его нам, и мы включим его в следующее издание этой книги. Наша контактная информация приведена в разделе «Вопросы и комментарии».

Почему я написал эту книгу

Запросы, запросы, запросы. С самого начала этого проекта моей целью было написать не столько «Сборник рецептов SQL», сколько «Сборник рецептов составления запросов». Я поставил перед собой задачу создать

книгу, в которой будут собраны запросы от относительно простых до довольно сложных, в надежде на то, что читатель поймет основные принципы и приемы, лежащие в их основе, и будет использовать их для решения собственных практических задач. Я представляю на рассмотрение читателей множество методов программирования на SQL, которые почерпнул из личного опыта. Вы можете изучать их, использовать и в конечном счете совершенствовать. Таким образом, все будут в выигрыше. Кажется так просто извлечь данные из базы данных, но в мире информационных технологий (ИТ) исключительно важно, чтобы такие операции проводились с максимально возможной эффективностью. Мы должны делиться приемами эффективного извлечения данных – это поможет повысить общий профессиональный уровень.

Отвлечемся на мгновение и вспомним о выдающемся вкладе в математику Георга Кантора, который первым обнаружил мощь множеств элементов (при работе с множеством, а не его составляющими). Сначала многие коллеги Кантора не приняли его работу, хотя со временем теория множеств была не только признана, но и стала фундаментальной основой математики! Однако самое главное, что место, занимаемое сегодня теорией множеств, является заслугой работы не столько самого Кантора, сколько других математиков, таких как Эрнст Цермело, Готтлоб Фреге, Абрахам Френкель, Торалф Сколем, Курт Гёдель и Джон ван Ньюман, которые разделили и развили его идеи. Такая коллективная работа помогла не только лучше понять теорию множеств, она способствовала ее совершенствованию.

Цель данной книги

В конечном счете цель данной книги – ознакомить читателя с тем, как с помощью SQL решать задачи, которые не входят в его обычную предметную область. SQL ушел далеко вперед за последние десять лет. Со многими задачами, обычно решаемыми с привлечением процедурных языков, например C или Java, теперь можно справиться непосредственно в SQL, но многие разработчики просто не знают об этом. Данная книга прольет свет на такую возможность.

Теперь, прежде чем мои слова будут неправильно истолкованы, позвольте заметить: я твердо убежден, что «не надо ремонтировать то, что не сломано». Скажем, имеется конкретная задача, при этом SQL используется только для извлечения данных, тогда как сложная бизнес-логика реализуется на каком-то другом языке программирования. Если код работает и обеспечивает приемлемую производительность, замечательно. Я ни в коем случае не предлагаю все ломать и переписывать с использованием SQL. Я только призываю избавиться от предрассудков и осознать, что тот SQL, на котором вы программировали в 1995 году, сильно отличается от SQL, используемого в 2005. Современный SQL обладает гораздо большими возможностями.

Для кого эта книга

Уникальность данной книги в том, что она ориентирована на широкую аудиторию, но это никак не отражается на качестве представленного материала. Учитывая то, что здесь показаны как сложные, так и простые решения, и что в случае отсутствия универсального решения приводятся решения для пяти разных баз данных, целевая аудитория, действительно, широка:

Новичок в SQL

Возможно, вы приобрели учебник по SQL или проходите обязательный курс по базам данных в университете и хотите дополнить свои вновь обретенные теоретические знания некоторыми сложными примерами из реальной жизни. А может, вы увидели запрос, который волшебным образом превращает строки в столбцы или проводит синтаксический разбор сериализованной строки. Рецепты данной книги объяснят, как создавать эти на первый взгляд неосуществимые запросы.

Разработчик не на SQL

Вы программируете на другом языке и в своей практике столкнулись с необходимостью работать с чужим сложным кодом на SQL. В рецептах данной книги, в частности в последних главах, проводятся разбор сложных запросов и их тщательный анализ, что поможет понять сложный код.

Рядовой специалист SQL

Для SQL-разработчиков средней руки эта книга – горшочек с золотом на конце радуги (ну, может быть, слишком сильно сказано; простите автору его экспрессивность). В частности, если вы пишете код на SQL уже довольно долго и не научились работать с оконными функциями, вы – наш читатель. Например, дни временных таблиц для хранения промежуточных результатов канули в лету; оконные функции могут обеспечить необходимый ответ за один запрос! Позвольте еще раз отметить, что я не намерен навязывать свои идеи опытным специалистам-практикам. Лучше рассматривайте данную книгу как средство совершенствования своих навыков, если вы не до конца понимаете некоторые нововведения языка SQL.

Эксперт SQL

Несомненно, вам знакомы эти рецепты, и, вероятно, у вас есть собственные варианты решения этих задач. Что тогда может дать эта книга вам? Возможно, вы всю жизнь работаете с SQL на одной платформе, скажем SQL Server, и теперь хотите выучить Oracle. А может, вы использовали только MySQL и хотите знать, как выглядели бы те же решения для PostgreSQL. Данная книга охватывает различные сис-

темы управления реляционными базами данных (СУБД[1]) и содержит решения для всех них параллельно. Для вас это шанс расширить свою базу знаний.

Как работать с этой книгой

Внимательно прочитайте это предисловие. В нем содержатся важные исходные данные и другая информация, которую вы упустите, если сразу углубитесь в отдельные рецепты. Раздел «Платформа и версия» рассказывает о том, какие СУБД рассматривает данная книга. Особое внимание следует обратить на раздел «Таблицы, используемые в данной книге», который поможет разобраться с примерами таблиц, используемыми в большинстве рецептов. В разделе «Условные обозначения» приведены важные соглашения по написанию кода и использованию шрифтов. Все эти разделы можно найти ниже в предисловии.

Необходимо помнить, что данная книга – сборник рецептов, собрание примеров кода, которые следует использовать как рекомендации при решении подобных (или аналогичных) задач. Не надо пытаться *выучить* по этой книге SQL, по крайней мере, не с нуля. Она должна дополнять, а не подменять академический учебник по SQL. Приведенные ниже подсказки помогут более эффективно использовать данную книгу:

- В книге используются специальные функции отдельных баз данных. Все их можно найти в книге Джонатана Генника (Jonathan Gennick) «SQL Pocket Guide», которую удобно иметь под рукой на случай, если вы не знаете некоторые функции, встречающиеся в моих рецептах.
- Если вы никогда не работали с оконными функциями или имели проблемы с запросами, использующими GROUP BY, прочитайте сначала Приложение А. В нем определяется и описывается на практических примерах, что такое группировка в SQL. Что еще более важно, там дается базовое представление о работе оконных функций. Оконные функции – одно из самых важных достижений SQL прошедшего десятилетия.
- Руководствуйтесь здравым смыслом! Поймите, что невозможно написать книгу, содержащую решения всех существующих задач. Используйте решения данной книги как шаблоны или руководства по применению необходимых техник при решении своих проблем. Вы должны говорить себе: «Замечательно, рецепт эффективен в этом конкретном случае, но мой набор данных отличается от приведенного, поэтому рецепт работает не вполне правильно». Попытайтесь найти общее между вашими данными и данными книги. Разложите приведенный запрос до его простейшей формы и постепенно услож-

[1] Хотя в английском языке используется аббревиатура RDBMS, в русском первую букву часто опускают и пишут просто СУБД. – *Примеч. науч. ред.*

няйте его. Все запросы начинаются с SELECT …FROM…, поэтому на элементарном уровне все они одинаковые. Постепенно усложняя, «достраивая» запрос шаг за шагом, функция за функцией, объединение за объединением, вы не только поймете, как эти конструкции меняют результирующее множество, но и увидите, чем рецепт отличается от того, который вам на самом деле нужен. И вот тогда вы уже сможете изменять рецепт соответственно своему конкретному набору данных.

- Тестируйте, тестируйте и еще раз тестируйте. Вне всякого сомнения, любая ваша таблица больше, чем таблица EMP из 14 строк, используемая в этой книге, поэтому, пожалуйста, тестируйте решения на своих данных, по крайней мере, чтобы убедиться, что они работают. Я не могу знать, как выглядят ваши таблицы, какие столбцы проиндексированы и какие отношения обозначены в вашей схеме. Поэтому, будьте добры, не применяйте слепо приведенные техники в своем коде, до тех пор пока не разберетесь в них полностью и не поймете, как они будут вести себя в применении к вашим конкретным данным.
- Не бойтесь экспериментировать. Подходите к работе творчески! Не бойтесь применять методики, отличающиеся от тех, которые использовал я. В данной книге я старался приводить разные функции, предоставляемые различными производителями, но во многих случаях есть и другие функции, которые могут работать так же хорошо, как и выбранные мною. Смело применяйте в рецептах данной книги собственные варианты.
- Новое не всегда значит лучшее. Если вы не используете некоторые из самых свежих возможностей SQL (например, оконные функции), это не означает, что ваш код неэффективен. Есть много примеров, когда традиционные решения SQL ничуть не хуже, а, подчас, даже лучше, чем новое решение. Пожалуйста, не забывайте об этом, особенно в Приложении B, «Вспомним Розенштейна». После прочтения данной книги у вас не должно возникнуть мысли о необходимости дополнить или изменить свой существующий код. Вы должны просто узнать о том, что теперь в SQL есть много новых и исключительно эффективных возможностей, которых не было 10 лет назад, и что они стоят времени, затраченного на их изучение.
- Ничего не бойтесь. Дойдя до раздела «Решение» рецепта и увидев запрос, который, кажется, невозможно понять, не пугайтесь. Я приложил максимум усилий, чтобы не только рассмотреть каждый запрос, начиная с его простейшей формы, но и показать промежуточные результаты выполнения каждой части запроса, прежде чем перейти к полному решению. Можно не охватить всей картины сразу, но по мере обсуждения и рассмотрения построения запроса и результатов каждого шага вы увидите, что не так сложно понять даже самые замысловатые запросы.

- Создавайте защищенный код, если это необходимо. Стараясь сделать запросы книги максимально лаконичными без ущерба их прозрачности, я опустил многие «меры защиты». Например, рассмотрим запрос, суммирующий заработные платы служащих. Может так случиться, что столбец объявлен типа VARCHAR и (к сожалению) числовые и строковые данные хранятся в одном поле. В приводимом в книге рецепте по вычислению суммы не выполняется проверка типов данных (и запрос даст сбой, поскольку функция SUM не умеет работать с символьными данными). Поэтому если у вас есть такие «данные», точнее сказать, «проблема», в коде необходимо обработать ее или (смею надеяться) исправить данные, потому что в предоставленных рецептах не предусмотрена такая практика, как смешение символьных и числовых данных в одном столбце. Основная цель – сосредоточиться на методике; когда вы поймете методику, такие отклонения не будут представлять сложности.
- Повторение – мать учения. Лучший способ освоить рецепты данной книги – сесть и написать их код самостоятельно. Прочитать и понять код – это одно, а вот самому написать его – совершенно другое. Читая код, можно лишь разобраться, почему что-то делается так, а не иначе, но только написав код, вы действительно научитесь создавать запросы.

Помните, что многие примеры данной книги искусственные. Но задачи не искусственные. Они реальные. Однако я построил все примеры вокруг небольшого набора таблиц с данными о служащих. Это сделано для того, чтобы, привыкнув к набору данных, можно было сосредоточиться на технике, иллюстрируемой каждым примером. Вы можете взглянуть на конкретную задачу и подумать: «Мне никогда не пришлось бы делать это с данными служащих». Но в таких случаях постарайтесь не обращать внимание на данные и сосредоточиться на иллюстрируемой методике. Методики – вот что важно. Я и мои коллеги используем их ежедневно. Думаем, и вам они будут полезны.

Чего нет в этой книге

Из-за ограничений по времени и размерам невозможно в одной книге представить решения всех встречающихся задач SQL, поэтому некоторые темы здесь не рассмотрены:

Описание данных

Данная книга не охватывает такие аспекты SQL, как создание индексов, наложение ограничений и загрузка данных. Синтаксис, используемый при реализации подобных задач, обычно сильно различается для разных баз данных, поэтому лучше всего обратиться к справочным материалам от производителя вашей СУБД. Кроме того, эти задачи не представляют особой проблемы, для их решения нет необходимости покупать книгу. Однако в главе 4 показаны ре-

цепты для решения распространенных задач, включающих вставку, обновление и удаление данных.

XML

По моему твердому убеждению, рецепты XML неуместны в книге по SQL. Хранение документов XML в реляционных базах данных приобретает все большую популярность, и в каждой СУБД имеются собственные расширения и инструментальные средства для извлечения и работы с такими данными. Работа с XML часто требует использования процедурного кода и, таким образом, выходит за рамки рассмотрения данной книги. Последние разработки, такие как XQuery, представляют собой совершенно не связанный с SQL предмет обсуждения и заслуживают отдельной книги (или книг).

Объектно-ориентированные расширения SQL

Пока не появился язык запросов, более подходящий для работы с объектами, я решительно против использования объектно-ориентированных возможностей и конструкций в реляционных базах данных. В настоящее время объектно-ориентированные возможности, поставляемые некоторыми производителями, больше подходят для процедурного программирования, чем для работы с множествами, для которой и был создан SQL.

Теоретические моменты

В данной книге вы не найдете обсуждения того, является ли SQL реляционным или нужны ли значения NULL. Такие теоретические дискуссии имеют место, но не в книге, целью которой является предоставление SQL-решений для реальных задач. Решая возникающие проблемы, мы просто используем доступные инструменты. Приходится работать с тем, что есть, а не с тем, что хотелось бы иметь.

Если вы хотите углубиться в теорию, начните с любой книги серии «Database Writings» Криса Дейта (Chris Date). Можете также приобрести экземпляр его последней книги «Database in Depth» (O'Reilly).

Политика производителей

В данной книге даются решения для пяти СУБД. Вполне естественно желание знать, решение какого производителя является «лучшим» или «самым быстрым». Любой производитель с радостью предоставит информацию, подтверждающую преимущества своего продукта, я не намерен заниматься этим.

Политика ANSI

Во многих книгах авторы обычно воздерживаются от использования функций, являющихся собственностью отдельных производителей. В данной книге эти функции применяются. Я не хотел писать сложный непроизводительный SQL-код, просто чтобы обеспечить его переносимость. Я никогда не работал в средах, в которых

были бы запрещены расширения, предоставляемые производителями. Вы платите за эти возможности; почему бы не использовать их?

Расширения производителей существуют не просто так, они предлагают во много раз более высокую производительность и делают программы гораздо более понятными, чем при использовании стандартного SQL. Предпочитаете стандартные решения – хорошо. Как уже говорилось ранее, я не заставляю вас переворачивать свой код с ног на голову. Если вы строго придерживаетесь ANSI и вам этого достаточно, замечательно. В конце концов, все мы ходим на работу, всем надо оплачивать счета, и все мы хотим приходить домой вовремя и наслаждаться тем, что нам осталось. Я не говорю, что стандартные решения неверны. Делайте то, что считаете нужным и что лучше всего вам подходит. Хочу лишь пояснить, что если вы ищете стандартные решения, эта книга не для вас.

Политика наследования

В рецептах данной книги используются новейшие возможности, доступные на момент ее написания. В более старых версиях СУБД многие из приведенных решений просто не будут работать. Технологии не стоят на месте, не должны отставать и мы. Если нужны решения для предыдущих версий, их можно найти во множестве выпущенных за прошедшие годы книг по SQL с массой примеров для более старых версий СУБД, чем рассматриваются в данной книге.

Структура данной книги

Данная книга включает в себя 14 глав и 2 приложения:

- Глава 1 «Извлечение записей» представляет очень простые запросы. В примерах показано, как с помощью предиката WHERE выбирать строки в результирующее множество, присваивать псевдонимы столбцам результирующего множества, использовать вложенный запрос для обращения к столбцам по псевдонимам, применять простую условную логику, ограничивать число возвращаемых в результате запроса строк, возвращать случайные строки и выявлять значения NULL. Большинство примеров очень простые, но некоторые из них появляются в более сложных рецептах. Поэтому, если вы не очень хорошо знакомы с SQL или нашли для себя что-то новое в перечисленных примерах, нелишним будет прочитать эту главу.
- Глава 2 «Сортировка результатов запроса» представляет рецепты для сортировки результатов запросов. Для этого применяется оператор ORDER BY. Сложность примеров варьируется от простого упорядочивания одного столбца до сортировки по подстрокам и сортировки с использованием условных выражений.
- Глава 3 «Работа с несколькими таблицами» представляет рецепты для сочетания данных нескольких таблиц. Всем новичкам в SQL и тем, кто немного недопонимает объединения, настоятельно реко-

мендую прочитать эту главу, прежде чем переходить к главе 5 и далее. Объединение таблиц – это суть SQL; чтобы добиться успеха в работе с SQL, необходимо понять объединения. В данной главе приведены примеры как внутренних, так и внешних объединений, показаны декартовы произведения, базовые операции над множествами (вычитание, объединение, пересечение) и влияние объединений на агрегатные функции.

- Глава 4 «Вставка, обновление, удаление» представляет рецепты для вставки, обновления и удаления данных. Большинство примеров очень просты (возможно, даже прозаичны). Тем не менее умение осуществлять операции типа вставки строк одной таблицы в другую и использовать связанные подзапросы при обновлениях, понимание последствий присутствия значений NULL и знание новых возможностей, например вставки в несколько таблиц и команды MERGE, – все это навыки, которые исключительно полезно иметь в своем профессиональном багаже.
- Глава 5 «Запросы на получение метаданных» представляет рецепты для получения доступа к метаданным используемой базы данных. Часто весьма полезно знать индексы, ограничения и таблицы схемы. Предложенные здесь простые рецепты позволят получать информацию о схеме. Кроме того, в этой главе показаны примеры «динамического SQL», т. е. SQL, сгенерированного SQL.
- Глава 6 «Работа со строками» представляет рецепты для работы со строками. SQL славен не своими возможностями синтаксического разбора строк, но немного смекалки (обычно с привлечением декартовых произведений) в сочетании с широким набором предоставляемых производителями функций позволяет достичь немалых успехов в этом деле. С данной главы начинается самое интересное. Среди наиболее любопытных примеров – подсчет экземпляров символа в строке, создание списков с разделителями из строк таблицы, преобразование списков с разделителями и строк в строки таблицы и разделение числовых и символьных данных строки, состоящей из буквенно-цифровых символов.
- Глава 7 «Работа с числами» представляет рецепты для решения обычных задач с числами. Здесь вы найдете совершенно обыкновенные примеры и узнаете, как легко оконные функции справляются с задачами, связанными с вычислениями и агрегацией. В этой главе рассматриваются вычисление текущей суммы; нахождение среднего, медианы и моды; вычисление процентилей и обработка значений NULL при проведении агрегации.
- Глава 8 «Арифметика дат» – первая из двух глав, посвященных работе с датами. Очень важно при решении каждодневных задач уметь осуществлять простые операции с датами. Примеры включают определение количества рабочих дней между двумя датами, вычисление разницы между двумя датами в разных единицах време-

ни (днях, месяцах, годах и т. д.) и подсчет количества определенных дней в месяце.

- Глава 9 «Работа с датами» – вторая из двух глав, посвященных работе с датами. В ней представлены рецепты самых распространенных операций над датами, которые приходится осуществлять ежедневно. Примеры включают возвращение всех дней года, поиск високосных годов, поиск первого и последнего дней месяца, создание календаря и дополнение пропущенных дат диапазона дат.
- Глава 10 «Работа с диапазонами данных» представляет рецепты для поиска значений в диапазонах и создания диапазонов значений. Примеры включают автоматическое формирование последовательности строк, вставку пропущенных числовых значений диапазона, определение начала и конца диапазона значений и выявление последовательности значений.
- Глава 11 «Расширенный поиск» представляет рецепты, которые жизненно важны для повседневной работы с базами данных, но подчас вызывают затруднения. Эти рецепты ничуть не сложнее остальных, но до сих пор многие разработчики очень неэффективно решают задачи, рассматриваемые данными рецептами. В примеры данной главы вошли поиск значений с использованием «хода конем», разбиение результирующего множества на страницы, пропуск строк таблицы, поиск взаимообратных значений, выбор n верхних записей и ранжирование результатов.
- Глава 12 «Составление отчетов и управление хранилищами данных» представляет запросы, обычно используемые при управлении хранилищами данных или формировании сложных отчетов. Данная глава изначально задумывалась как центральный раздел книги. Примеры включают преобразование строк в столбцы и наоборот (отчеты с перекрестными ссылками), создание групп данных, создание гистограмм, вычисление простых и полных подсумм, агрегацию скользящего окна строк и группировку строк по заданному интервалу времени.
- Глава 13 «Иерархические запросы» представляет рецепты для работы с иерархическими данными. Независимо от модели данных однажды возникает необходимость отформатировать их в виде дерева иерархии или отношений родитель-потомок. В этой главе приведены рецепты решения таких задач. С помощью традиционного SQL создавать структурированные в виде дерева результирующие множества сложно, поэтому в данной главе показано, как работать с чрезвычайно полезными предоставляемыми производителями специальными функциями. Примеры включают представление отношений родитель-потомок, обход иерархии от корневого узла до концевых узлов и накопление иерархии.
- Глава 14 «Всякая всячина» – это коллекция разнообразных рецептов, которые не вписались ни в одну из рассматриваемых тем, но

при этом интересны и полезны. Данная глава отличается от остальных, поскольку предлагает решения только для конкретных СУБД. Это единственная глава в книге, где в каждом рецепте обсуждается только одна база данных. Сделано это по двум причинам. Во-первых, глава предполагалась более как забава для энтузиастов. Во-вторых, некоторые рецепты приведены только для того, чтобы показать функции, не имеющие эквивалентов в других СУБД (примеры включают операторы SQL Server PIVOT/UNPIVOT и оператор Oracle MODEL). Однако в некоторых случаях представленное в этой главе решение можно с небольшими изменениями использовать для другой платформы.

- Приложение A «Оконные функции, краткий обзор» – это краткий курс по оконным функциям с подробным обсуждением группировки в SQL. Для большинства оконные функции являются новинкой, поэтому данное приложение представлено в виде краткого учебного руководства. Кроме того, по собственному опыту я знаю, что использование оператора GROUP BY в запросах приводит в замешательство многих разработчиков. В этой главе дается четкое определение SQL-группе и анализируются различные запросы, подтверждающие действительность этого определения. Продолжается разговор рассмотрением влияния значений NULL на группировку, агрегирование и разбиение на сегменты. В заключение предлагается обсуждение еще более непонятного и при этом исключительно мощного синтаксиса оператора OVER оконных функций (т. е. оператора «кадрирования» или «сегментирования»).
- Приложение B «Вспоминаем Розенштейна» – дань уважения Дэвиду Розенштейну, которому я обязан своим успехом в SQL. Книга Розенштейна «The Essence of SQL» (Coriolis Group Books) была первой книгой по SQL, которую я прочитал вне университетской программы. Именно она научила меня «думать на SQL». До сих пор львиную долю своего успеха в SQL я отношу на счет книги Дэвида. Она в самом деле не похожа на все остальные прочитанные мною работы по SQL, и я счастлив, что самостоятельно выбрал первой именно ее. Приложение B останавливается на некоторых запросах, представленных в «The Essence of SQL», и предлагает альтернативные решения с использованием оконных функций (которых не существовало на момент написания «The Essence of SQL»).

Платформа и версия

SQL – это движущаяся мишень. Производители постоянно добавляют новые возможности и функциональность в свои продукты. Таким образом, необходимо обозначить заранее, какие версии платформ использовались при подготовке данной книги:

- DB2 v.8

- Oracle Database 10*g* (за исключением небольшого числа рецептов, решения будут работать также для Oracle 8*i* Database и Oracle 9*i* Database)
- PostgreSQL 8
- SQL Server 2005
- MySQL 5

Таблицы, используемые в данной книге

Основная масса примеров данной книги использует две таблицы, EMP и DEPT. EMP – это простая таблица из 14 строк, содержащая только поля с числовыми, строковыми данными и датами. Таблица DEPT состоит из четырех строк с числовыми и строковыми полями. Эти таблицы можно встретить во многих книгах по базам данных, да и отношение многие-к-одному между отделами и служащими предельно понятно.

Говоря о таблицах, используемых в примерах, хочу еще раз заметить, что все, за исключением лишь нескольких, примеры данной книги работают с этими таблицами. Я никогда не меняю данные в своих примерах и не предлагаю решения, которые вы вряд ли когда-нибудь реализуете в действительности, как это делается в некоторых книгах.

И, говоря о решениях, позвольте упомянуть, что везде, где было возможно, я старался предоставить универсальное решение, подходящее для всех пяти СУБД, рассматриваемых в книге. Когда это было неосуществимо, я приводил решения, общие для нескольких производителей. Например, Oracle и DB2 поддерживают оконные функции, поэтому часто решения для них одинаковые. Если решения общие или, по крайней мере, очень похожи, обсуждения также аналогичны.

Содержимое таблиц EMP и DEPT приведено ниже:

```
select * from emp;

EMPNO ENAME  JOB        MGR HIREDATE     SAL COMM  DEPTNO
----- ------ --------- ---- ----------- ---- ---- -------
 7369 SMITH  CLERK     7902 17-DEC-1980  800           20
 7499 ALLEN  SALESMAN  7698 20-FEB-1981 1600  300      30
 7521 WARD   SALESMAN  7698 22-FEB-1981 1250  500      30
 7566 JONES  MANAGER   7839 02-APR-1981 2975           20
 7654 MARTIN SALESMAN  7698 28-SEP-1981 1250 1400      30
 7698 BLAKE  MANAGER   7839 01-MAY-1981 2850           30
 7782 CLARK  MANAGER   7839 09-JUN-1981 2450           10
 7788 SCOTT  ANALYST   7566 09-DEC-1982 3000           20
 7839 KING   PRESIDENT      17-NOV-1981 5000           10
 7844 TURNER SALESMAN  7698 08-SEP-1981 1500    0      30
 7876 ADAMS  CLERK     7788 12-JAN-1983 1100           20
 7900 JAMES  CLERK     7698 03-DEC-1981  950           30
 7902 FORD   ANALYST   7566 03-DEC-1981 3000           20
 7934 MILLER CLERK     7782 23-JAN-1982 1300           10
```

```
select * from dept;

DEPTNO DNAME          LOC
------ -------------- ---------
    10 ACCOUNTING     NEW YORK
    20 RESEARCH       DALLAS
    30 SALES          CHICAGO
    40 OPERATIONS     BOSTON
```

Кроме того, в книге используются четыре сводные таблицы: T1, T10, T100 и T500. Поскольку данные таблицы существуют только для того, чтобы упростить разворачивание, я не нашел необходимым присваивать им более содержательные имена. Число, следующее за «T» в имени каждой из сводных таблиц, соответствует числу строк таблицы, начиная с 1. Например, значения для T1 и T10:

```
select id from t1;

        ID
----------
         1

select id from t10;

        ID
----------
         1
         2
         3
         4
         5
         6
         7
         8
         9
        10
```

В качестве альтернативы некоторые производители допускают неполные выражения SELECT. Например, можно применять SELECT без оператора FROM. Мне не очень нравится такая практика, поэтому я пользуюсь вспомогательной таблицей T1 с одной строкой вместо неполного запроса.

Все остальные таблицы, используемые в отдельных рецептах и главах, будут представлены в тексте.

Условные обозначения

В данной книге используется ряд типографских обозначений и соглашений по написанию кода. Разберитесь с ними, это поможет лучше понимать текст. Обратите особое внимание на соглашения по написанию кода; мы не можем обсуждать их в каждом рецепте книги, поэтому они перечислены в этом разделе.

Типографские обозначения

В данной книге используются следующие типографские обозначения:

ВЕРХНИЙ РЕГИСТР

Используется для обозначения ключевых слов SQL в тексте.

нижний регистр

Используется для всех запросов в примерах кода. В других языках программирования, таких как C и JAVA, большинство ключевых слов пишутся строчными буквами, и мне такая запись показалась более удобной для чтения, чем запись прописными буквами. Таким образом, все запросы будут написаны в нижнем регистре.

`Моноширинный полужирный`

Показывает ввод пользователя в примерах, отображающих взаимодействие.

Обозначает подсказку, совет или общее замечание.

Обозначает предупреждение или предостережение.

Соглашения по написанию кода

Я предпочитаю всегда записывать выражения SQL, как ключевые слова, так и определяемые пользователем идентификаторы, строчными буквами. Например:

```
select empno, ename
  from emp;
```

Вы можете придерживаться другого стиля записи. Например, многим нравится записывать ключевые слова SQL прописными буквами. Стиль написания кода определяется только личными предпочтениями и требованиями проекта.

Несмотря на то что примеры кода записаны в нижнем регистре, в тексте я выделяю ключевые слова SQL и идентификаторы с помощью прописных букв. Например:

Предыдущее выражение SELECT представляет запрос к таблице EMP.

Хотя в данной книге рассматриваются разные базы данных, я решил придерживаться одного формата вывода:

```
EMPNO ENAME
----- ------
 7369 SMITH
```

```
 7499 ALLEN
...
```

Во многих решениях в операторе FROM используются вложенные *запросы* или подзапросы. Стандарт ANSI SQL требует, чтобы им присваивались псевдонимы. (Только в Oracle псевдонимы можно не задавать.) Таким образом, в моих решениях используются псевдонимы, например x и y, для обозначения результирующих множеств для вложенных запросов:

```
select job, sal
  from (select job, max(sal) sal
          from emp
        group by job) x;
```

Обратите внимание на букву X, стоящую после закрывающей круглой скобки. Эта X становится именем «таблицы», возвращаемой подзапросом оператора FROM. Тогда как псевдонимы столбцов являются ценным средством для написания кода с автоматическим формированием документации, псевдонимы вложенных запросов (для большинства рецептов данной книги) являются простой формальностью. Обычно для них используются простейшие имена, такие как X, Y, Z, TMP1 и TMP2. Я применял более описательные псевдонимы в тех случаях, когда чувствовал, что это обеспечит лучшее понимание.

Вы заметите, что строки SQL-кода в разделе «РЕШЕНИЕ» рецепта обычно пронумерованы, например:

```
1 select ename
2     from emp
3  where deptno = 10
```

Номера не являются частью синтаксиса; они включены для удобства ссылки на определенные части запроса в тексте.

Использование примеров кода

Данная книга призвана помочь вам в вашей работе. В общем, вы можете использовать код из этой книги в своих программах и документации. Не нужно обращаться в O'Reilly за разрешением на копирование небольших частей кода, например, при написании программы, в которой используется несколько блоков кода из этой книги. А вот продажа или распространение CD-ROM с примерами из книг O'Reilly *требуют* специального разрешения. Вы можете свободно ссылаться на книгу и цитировать примеры кода, но для включения больших частей кода из этой книги в документацию вашего продукта *требуется* наше согласие.

Будем благодарны, но не настаиваем на указании авторства. Обычно ссылка на источник включает название, автора, издателя и ISBN. Например: «SQL Cookbook, by Anthony Molinaro. Copyright 2006 O'Reilly Media, Inc., 0-596-00976-3».

Если вам кажется, что использование вами примеров кода выходит за рамки законного использования или разрешений, оговоренных выше, не стесняйтесь, обращайтесь к нам по адресу *permissions@oreilly.com*.

Вопросы и комментарии

Мы тщательно протестировали и проверили информацию, представленную в данной книге, но, возможно, вы найдете неточности или ошибки, которые мы пропустили. Пожалуйста, напишите нам об этом:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (в США или Канаде)
(707) 829-0515 (международный или местный)
(707) 829-0104 (факс)

Можно также присылать сообщения по электронной почте. Чтобы попасть в список почтовой рассылки или получить каталог, пришлите электронное письмо по адресу:

info@oreilly.com

Задавать технические вопросы, присылать коммментарии к книге или предлагать дополнительные рецепты для будущих изданий можно, написав по адресу:

bookquestions@oreilly.com

На веб-сайте данной книги можно найти примеры и список опечаток (выявленные ошибки и исправления представлены здесь):

http://www.oreilly.com/catalog/sqlckbk

Safari® Enabled

Если на обложке технической книги есть пиктограмма «Safari® Enabled», это означает, что книга доступна в Сети через O'Reilly Network Safari Bookshelf.

Safari предлагает намного лучшее решение, чем электронные книги. Это виртуальная библиотека, позволяющая без труда находить тысячи лучших технических книг, вырезать и вставлять примеры кода, загружать главы и находить быстрые ответы, когда требуется наиболее верная и свежая информация. Она свободно доступна по адресу *http://safari.oreilly.com*.

Благодарности

Эта книга не вышла бы в свет, если бы не поддержка огромного числа людей. Хочу поблагодарить свою маму Конни, которой и посвящаю

эту книгу. Без твоего тяжелого труда и самопожертвования я не был бы тем, кем являюсь сейчас. Спасибо тебе, мама. Я благодарен и признателен тебе за все, что ты сделала для нас с братом. Я счастлив, что у меня такая мать.

Спасибо моему брату Джо. Каждый раз, когда я приезжаю домой из Балтимора, чтобы отдохнуть от дел, ты напоминаешь мне, как замечательно не работать и что я должен оставить писательство, чтобы вернуться к более важным в жизни вещам. Ты замечательный человек, и я уважаю тебя. Я горжусь тобой, и для меня честь называть тебя своим братом.

Спасибо моей замечательной невесте Джорджии. Без твоей поддержки я не справился бы с более чем 600 страницами этой книги. Ты была рядом, день за днем разделяя все тяготы и невзгоды. Я знаю, что тебе было ничуть не легче, чем мне. Я целыми днями работал и ночи напролет писал, но ты замечательно справилась с этим. Ты была понимающей и любящей, и я до конца дней благодарен тебе за это. Спасибо. Я люблю тебя.

Спасибо моим будущим теще и тестю, Кики и Джорджу. Благодарю вас за поддержку в течение всего этого времени. У вас в гостях я всегда чувствовал себя как дома, и кормили вы нас с Джорджией от души. Спасибо моим свояченицам, Анне и Кэти, мне всегда нравилось потусоваться с вами, девчонки. Это позволяло нам с Джорджией отвлечься от книги и отдохнуть от Балтимора, что было так необходимо.

Спасибо моему редактору Джонатану Геннику, без которого этой книги просто не было бы. Джонатан, твоя заслуга в создании этой книги громадна. Ты далеко вышел за рамки обычных обязанностей редактора, за что я тебе бесконечно благодарен. Все, что ты делал, начиная от предоставления рецептов до внесения огромного количества корректив и сохранения бодрости духа, несмотря на поджимающие сроки, помогло мне справиться с книгой. Я благодарен тебе за то, что ты был моим редактором, и счастлив, что ты предоставил мне эту возможность. Для меня, опытного администратора баз данных и автора, было удовольствием работать со специалистом такого уровня. Я не знаю другого такого редактора, кроме Джонатана, который смело мог бы оставить редакторское дело и устроиться куда угодно администратором баз данных (database administrator, DBA). То, что ты являешься DBA, безусловно, является твоим преимуществом как редактора, поскольку ты обычно знаешь, что я хочу сказать, даже когда у меня возникают сложности с формулированием мысли. O'Reilly повезло, что ты работаешь у них, и мне повезло, что ты был моим редактором.

Хотел бы поблагодарить Алеса Спетика и Джонатана Генника за книгу «Transact-SQL Cookbook». Всем известно изречение Исаака Ньютона: «Если я видел дальше других, то потому, что стоял на плечах гигантов». Слова Алеса Спетика, приведенные в разделе благодарностей книги «Transact-SQL Cookbook», являются подтверждением этого

высказывания, и, я убежден, должны присутствовать в каждой книге по SQL. Я привожу их здесь:

> Надеюсь, что эта книга будет достойным дополнением существующих произведений выдающихся авторов, таких как Джо Селко (Joe Celko), Дэвид Розенштейн (David Rozenshtein), Анатолий Абрамович (Anatoly Abramovich), Юджин Бергер (Eugine Berger), Ижтик Бен-Ган (Iztik Ben-Gan), Ричард Снодграсс (Richard Snodgrass) и других. Многие ночи я посвятил изучению их трудов, и практически все, что я знаю, я почерпнул из их книг. Создавая эти строки, я понимаю, что за каждой ночью, проведенной мной за изучением их секретов, стоят 10 ночей, которые они потратили, чтобы последовательно и понятно изложить свои знания на бумаге. Для меня честь иметь возможность отдать долг сообществу разработчиков на SQL.

Хотел бы поблагодарить Санжея Мишру (Sanjay Mishra) за блестящую книгу «Mastering Oracle SQL»[1], а также за то, что познакомил меня с Джонатаном. Если бы не Санжей, возможно, я никогда не встретил бы Джонатана и не написал бы эту книгу. Удивительно, как простое электронное письмо может изменить твою жизнь. Отдельное спасибо Дэвиду Розенштейну за его книгу «Essence of SQL», которая дала мне твердое понимание, как думать и решать задачи, оперируя категориями множеств/SQL. Выражаю благодарность Дэвиду Розенштейну, Анатолию Абрамовичу и Юджину Бергеру за книгу «Optimizing Transact-SQL», из которой я узнал о многих расширенных техниках SQL, которыми пользуюсь сегодня.

Хочу поблагодарить всю команду Wireless Generation, замечательную компанию с замечательными людьми. Большое спасибо всем, кто тратил свое время на обзоры и рецензирование книги или помогал советами: Джесси Девису (Jesse Davis), Джоэлу Паттерсону (Joel Patterson), Филиппу Зее (Philip Zee), Кевину Маршаллу (Kevin Marshall), Дугу Дениелсу (Doug Daniels), Отису Господнетику (Otis Gospodnetic), Кену Ганну (Ken Gunn), Джону Стюарту (John Stewart), Джиму Абрамсону (Jim Abramson), Адаму Майеру (Adam Mayer), Сьюзан Лау (Susan Lau), Алексис Ле-Кок (Alexis Le-Quoc) и Полу Фьюеру (Paul Feuer). Спасибо Мегги Хо (Maggie Ho) за тщательный обзор моей работы и исключительно полезные отзывы, касающиеся раздела «Оконные функции, краткий обзор». Хотелось бы выразить благодарность Чаку Ван Барену (Chuck Van Buren) и Джиллиан Гутенберг (Gillian Gutenberg) за их замечательный совет относительно бега. Утренняя зарядка помогала мне освежить голову и снять напряжение. Не думаю, что я смог бы закончить книгу, не давая себе немного передохнуть и отвлечься. Спасибо Стиву Кангу (Steve Kang) и Чаду Левинсону (Chad Levinson) за то, что терпели мои бесконечные разговоры по вечерам о разных тех-

[1] Санжей Мишра и Алан Бьюли «Секреты Oracle SQL». – Пер. с англ. – СПб: Символ-Плюс, 2003.

никах SQL, когда единственным их желанием в конце долгого рабочего дня было отправиться в Union Square и взять пиво и бургер в Heartland Brewery. Спасибо Аарону Бойду (Aaron Boyd) за его поддержку, добрые слова и, что самое важное, ценные советы. Аарон честный, трудолюбивый и очень открытый парень; такие люди, как он, являются украшением компании. Хочу поблагодарить Оливера Помела (Olivier Pomel) за его поддержку и помощь при написании этой книги, в частности, за решение для DB2 по преобразованию строк в списки с разделителями. Оливер предложил это решение, даже не имея системы DB2, чтобы протестировать его! Я объяснил ему, как работает оператор WITH, и минуту спустя он предложил решение, которое вы видите в книге.

Технические рецензии Джона Харриса (Jonah Harris) и Дэвида Розенштейна также были очень полезны. Арун Марат (Arun Marathe), Нуно Пинто до Сото (Nuno Pinto do Souto) и Эндрю Одеван (Andrew Odewahn) внесли неоценимый вклад в проектирование и выбор рецептов на стадии планирования книги. Большое спасибо всем вам.

Хочу поблагодарить Джона Хайду (John Haydu) и команду разработки оператора MODEL в корпорации Oracle Corporation за рецензирование моей статьи по MODEL для O'Reilly, в результате чего я глубже разобрался с принципом работы этого оператора. Спасибо Тому Кайту (Tom Kyte) из Oracle Corporation за то, что позволил мне использовать свою функцию TO_BASE в решении, реализованном только с помощью SQL; Бруно Деньюту (Bruno Denuit) из Microsoft за ответы на мои вопросы по функциональности оконных функций, появившихся в SQL Server 2005; Симону Риггсу (Simon Riggs) из PostgreSQL, который держал меня в курсе последних новостей об SQL в PostgreSQL (огромное спасибо: Симон, благодаря твоей информации о том, что и когда появляется, я мог включать новые возможности SQL, такие как замечательнейшая функция GENERATE_SERIES, которая, по моему мнению, обеспечивает более элегантные решения, чем сводные таблицы).

Последним в этом списке, но никак не по значимости, я хотел бы поблагодарить Кея Янга (Kay Young). Талантливому и увлеченному своим делом человеку всегда приятно сотрудничать с людьми, такими же одаренными и горящими. Многие рецепты данной книги являются результатом совместной работы с Кеем и выработки SQL-решений для каждодневных задач в Wireless Generation. Хочу поблагодарить тебя и сказать, что я безгранично признателен за всю ту помощь, которую получил от тебя: от совета до исправления грамматических ошибок и написания кода. Ты сыграл значительную роль в создании данной книги. Работать с тобой было просто замечательно, и Wireless Generation – лучшая компания, потому что ты работаешь в ней.

Энтони Молинаро
сентябрь 2005

1

Извлечение записей

Данная глава посвящена основным принципам работы с выражением SELECT. Очень важно иметь четкое понимание основ: многие рассматриваемые в данной главе вопросы не только являются составляющей частью самых трудных рецептов книги, но постоянно возникают при каждодневной работе с SQL.

Извлечение всех строк и столбцов из таблицы

Задача

Имеется таблица и требуется просмотреть все хранящиеся в ней данные.

Решение

Используйте выражение SELECT со специальным символом «*»:

```
1 select *
2   from emp
```

Обсуждение

В SQL символ «*» имеет специальное назначение. Его применение обусловливает извлечение всех столбцов заданной таблицы. Поскольку в данном случае предикат WHERE не задан, будут возвращены все строки таблицы. Альтернативный вариант – явно перечислить в выражении SELECT все столбцы:

```
select empno,ename,job,sal,mgr,hiredate,comm,deptno
  from emp
```

При разработке и/или отладке программ проще использовать SELECT *. Однако при написании запросов в приложениях лучше задавать каждый столбец в отдельности. Явное задание имен столбцов не влияет

на производительность, но при этом всегда точно известно, какие именно столбцы будут возвращены в результате запроса. Кроме того, такое задание делает запросы понятными и другим пользователям (которые могут не знать всех столбцов таблиц, присутствующих в запросе).

Извлечение подмножества строк из таблицы

Задача

Из имеющейся таблицы требуется извлечь строки, удовлетворяющие определенному условию.

Решение

Условие выбора строк задается с помощью предиката WHERE. Например, для получения списка всех служащих 10-го отдела необходим следующий запрос:

```
1 select *
2   from emp
3  where deptno = 10
```

Обсуждение

Предикат WHERE дает возможность извлекать только определенные строки. Если для данной строки выражение предиката WHERE истинно, то она будет возвращена.

Большинство производителей поддерживают обычные операторы, такие как =, <, >, <=, >=, !, <>. Кроме того, для выбора записей, отвечающих нескольким условиям, используются операторы AND (логическое И), OR (логическое ИЛИ) и круглые скобки, как показано в следующем рецепте.

Выбор строк по нескольким условиям

Задача

Требуется выбрать строки, удовлетворяющие нескольким условиям.

Решение

Используйте предикат WHERE в сочетании с операторами OR и AND. Например, по следующему запросу будут выбраны все служащие 10-го отдела, а также служащие, получающие комиссионные, и служащие 20-го отдела, зарабатывающие не более $2000:

```
1 select *
2   from emp
3  where deptno = 10
4     or comm is not null
5     or sal <= 2000 and deptno=20
```

Обсуждение

Чтобы выбрать строки, отвечающие нескольким условиям, используются операторы AND, OR в сочетании с круглыми скобками. В приведенном в «Решении» примере предикат WHERE фильтрует строки, для которых:

- DEPTNO равен 10 или
- COMM не NULL или
- зарплата любого служащего, для которого DEPTNO равен 20, составляет не более $2000.

Условия, заключенные в круглые скобки, должны выполняться одновременно.

Например, посмотрим, как изменится результирующее множество, если использовать в запросе круглые скобки, как показано ниже:

```
select *
 from emp
where (    deptno = 10
        or comm is not null
        or sal <= 2000
      )
  and deptno=20
```

```
EMPNO ENAME  JOB       MGR HIREDATE      SAL    COMM DEPTNO
----- ------ ------ ----- ----------- ----- ------- ------
 7369 SMITH  CLERK   7902 17-DEC-1980   800              20
 7876 ADAMS  CLERK   7788 12-JAN-1983  1100              20
```

Извлечение подмножества столбцов из таблицы

Задача

Требуется выбрать из таблицы значения лишь определенных столбцов.

Решение

Задайте интересующие вас столбцы. Например, чтобы выбрать только имя, номер отдела и зарплату служащих, можно использовать такой запрос:

```
1 select ename,deptno,sal
2   from emp
```

Обсуждение

Задавая в операторе SELECT имена столбцов, мы обеспечиваем вывод только интересующих нас данных. Это особенно важно при извлечении данных по сети, поскольку предотвращает пустые затраты времени на извлечение ненужных данных.

Как задать столбцам значимые имена

Задача

Требуется изменить имена возвращенных в результате запроса столбцов, чтобы сделать их более понятными и удобными для чтения. Рассмотрим запрос, возвращающий зарплаты и комиссионные всех служащих:

```
1 select sal,comm
2   from emp
```

Что такое «sal»? Сокращенная запись «sale» (продажа)? Может, это чье-то имя? Что означает «comm»? Сокращение от «communication» (общение)? Столбцы в результирующем наборе должны иметь более понятные названия.

Решение

Чтобы изменить имена в результатах запроса, используйте ключевое слово AS следующим образом: *исходное_имя AS новое_имя*. Для некоторых баз данных применение AS необязательно, но во всех оно допускается:

```
1 select sal as salary, comm as commission
2   from emp

SALARY COMMISSION
------- ----------
    800
   1600        300
   1250        500
   2975
   1250       1300
   2850
   2450
   3000
   5000
   1500          0
   1100
    950
   3000
   1300
```

Обсуждение

Задавая новые имена возвращаемым запросом столбцам с помощью ключевого слова AS, мы *присваиваем псевдонимы* (*aliasing*) этим столбцам. Новые имена являются *псевдонимами* (*aliases*). Хорошо подобранные псевдонимы способствуют пониманию запроса и его результатов пользователями.

Обращение к столбцу в предикате WHERE по псевдониму

Задача

Допустим, что именам столбцов результирующего множества присваиваются псевдонимы, чтобы сделать их более понятными и содержательными. Требуется исключить некоторые из строк с помощью предиката WHERE. Однако при использовании псевдонимов в предикате WHERE возникает ошибка:

```
select sal as salary, comm as commission
  from emp
 where salary < 5000
```

Решение

Чтобы обратиться к столбцу по псевдониму, необходимо использовать вложенный запрос:

```
1 select *
2   from (
3 select sal as salary, comm as commission
4   from emp
5        ) x
6  where salary < 5000
```

Обсуждение

В этом простом примере можно не прибегать к вложенному запросу, а в предикате WHERE напрямую обратиться к столбцам по их именам, COMM или SAL. Результат будет такой же. Данный пример показывает принципы использования в предикате WHERE следующих элементов языка:

- Агрегатных функций
- Скалярных подзапросов
- Оконных функций
- Псевдонимов

Использование вложеного запроса, присваивающего псевдонимы, дает возможность обращаться к столбцам по псевдонимам во внешнем запросе. Почему это необходимо? Предикат WHERE обрабатывается раньше оператора SELECT; таким образом, на момент обработки WHERE в запросе, приведенном в разделе «Задача», столбцов SALARY и COMMISSION еще не существует. Эти псевдонимы присваиваются уже после обработки WHERE. А вот оператор FROM выполняется до предиката WHERE. Размещение исходного запроса в операторе FROM обеспечивает формирование его результатов до обработки самого внешнего WHERE, следовательно, этот предикат WHERE будет «видеть»

псевдонимы. Данная техника очень полезна в случае, если при выводе результатов запроса требуется менять имена столбцов таблицы.

В данном решении вложенному запросу присвоен псевдоним X. Не все базы данных требуют явного присваивания псевдонима вложенному запросу, но все допускают его.

Конкатенация значений столбцов

Задача

Требуется извлечь значения нескольких столбцов в один столбец. Например, в результате запроса к таблице EMP необходимо получить следующий результат:

```
CLARK WORKS AS A MANAGER
KING WORKS AS A PRESIDENT
MILLER WORKS AS A CLERK
```

Однако данные, формирующие это результирующее множество, располагаются в таблице EMP в двух разных столбцах, ENAME и JOB (работа):

```
select ename, job
  from emp
 where deptno = 10
```

```
ENAME      JOB
---------- ---------
CLARK      MANAGER
KING       PRESIDENT
MILLER     CLERK
```

Решение

Найдите и используйте предоставляемую вашей СУБД встроенную функцию для конкатенации значений нескольких столбцов.

DB2, Oracle, PostgreSQL

В этих базах данных оператором конкатенации является двойная вертикальная черта:

```
1 select ename||' WORKS AS A '||job as msg
2   from emp
3  where deptno=10
```

MySQL

Эта база данных поддерживает функцию CONCAT:

```
1 select concat(ename, ' WORKS AS A ',job) as msg
2   from
3  where deptno=10
```

SQL Server

Для конкатенации используется оператор «+»:

```
1 select ename + ' WORKS AS A ' + job as msg
2   from emp
3  where deptno=10
```

Обсуждение

Используйте функцию CONCAT для конкатенации значений нескольких столбцов. Оператор || является сокращенной записью функции CONCAT в DB2, Oracle и PostgreSQL, тогда как + выполняет конкатенацию в SQL Server.

Использование условной логики в выражении SELECT

Задача

Требуется осуществить операцию IF-ELSE в выражении SELECT. Например, необходимо сформировать результирующее множество, в котором для служащих, получающих $2000 или менее, возвращается значение «UNDERPAID» (низкооплачиваемый), для служащих, получающих $4000 или более, возвращается значение «OVERPAID» (высокооплачиваемый), и для служащих, заработная плата которых находится между $2000 и $4000, возвращается значение «OK». Результирующее множество должно выглядеть следующим образом:

```
ENAME             SAL STATUS
---------- ---------- ---------
SMITH              800 UNDERPAID
ALLEN             1600 UNDERPAID
WARD              1250 UNDERPAID
JONES             2975 OK
MARTIN            1250 UNDERPAID
BLAKE             2850 OK
CLARK             2450 OK
SCOTT             3000 OK
KING              5000 OVERPAID
TURNER            1500 UNDERPAID
ADAMS             1100 UNDERPAID
JAMES              950 UNDERPAID
FORD              3000 OK
MILLER            1300 UNDERPAID
```

Решение

Для осуществления условной логики непосредственно в выражении SELECT используйте выражение CASE:

```
1 select ename,sal,
2        case when sal <= 2000 then 'UNDERPAID'
3             when sal >= 4000 then 'OVERPAID'
```

```
4             else 'OK'
5        end as status
6   from emp
```

Обсуждение

Выражение CASE позволяет применять условную логику к возвращаемым в результате запроса значениям. В целях формирования более удобного для чтения результирующего множества можно указать псевдоним для выражения CASE. В данном решении результаты выражения CASE выводятся в столбце под псевдонимом STATUS. Конструкция ELSE является необязательной. Если ELSE опущено, выражение CASE возвратит NULL для любой не удовлетворяющей условию строки.

Ограничение числа возвращаемых строк

Задача

Требуется ограничить число возвращаемых запросом строк. Порядок не имеет значения; подойдут любые *n* строк.

Решение

Для управления числом возвращаемых строк используйте встроенные функции, предоставляемые вашей базой данных.

DB2

В DB2 используйте оператор FETCH FIRST:

```
1 select *
2   from emp fetch first 5 rows only
```

MySQL and PostgreSQL

В MySQL и PostgreSQL то же самое можно выполнить, используя LIMIT:

```
1 select *
2   from emp limit 5
```

Oracle

В Oracle ограничение на число возвращаемых строк накладывается с помощью функции ROWNUM в предикате WHERE:

```
1 select *
2   from emp
3  where rownum <= 5
```

SQL Server

Для ограничения числа возвращаемых строк используйте ключевое слово TOP:

```
1 select top 5 *
2   from emp
```

Обсуждение

Многие производители предоставляют операторы FETCH FIRST и LIMIT, обеспечивающие возможность задавать число строк, которое должно быть возвращено в результате запроса. Oracle отличается тем, что в нем приходится использовать функцию ROWNUM, возвращающую порядковый номер каждой строки результирующего множества (возрастающую, начиная с 1, величину).

Рассмотрим, что происходит при использовании ROWNUM <= 5 для возвращения первых пяти строк:

1. Oracle выполняет запрос.
2. Oracle извлекает первую строку и называет ее строкой номер 1.
3. Номер строки больше 5? Если нет, Oracle возвращает строку, потому что она отвечает критерию: ее порядковый номер меньше или равен 5. Если да, Oracle не возвращает строку.
4. Oracle извлекает следующую строку и присваивает ей следующий порядковый номер по возврастанию (2, затем 3, затем 4 и т. д.).
5. Переходим к шагу 3.

Как видно из данного процесса, присвоение значений, возвращаемых функцией ROWNUM Oracle, происходит *после* извлечения очередной строки. Это очень важно и является ключевым моментом. Многие разработчики на Oracle пытаются реализовать извлечение только, скажем, пятой возвращенной запросом строки, задавая ROWNUM = 5. Такое использование условия равенства в сочетании с ROWNUM является неверным. При попытке возвратить пятую строку с помощью ROWNUM = 5 происходит следующее:

1. Oracle выполняет запрос.
2. Oracle извлекает первую строку и называет ее строкой номер 1.
3. Номер строки равен 5? Если нет, Oracle отбрасывает строку, потому что она не отвечает заданному критерию. Если да, Oracle возвращает строку. Но ответ всегда будет отрицательным!
4. Oracle извлекает следующую строку и называет ее строкой номер 1, поскольку первая возвращенная запросом строка должна быть пронумерована как первая строка.
5. Переходим к шагу 3.

После тщательного разбора этого процесса становится понятно, почему использование ROWNUM = 5 не обеспечивает возвращения пятой строки. Невозможно получить пятую строку, не возвратив перед этим строки с первой по четвертую!

Однако заметьте, что с помощью ROWNUM = 1 можно получить первую строку. Может показаться, что это противоречит приведенному выше объяснению. Причина, почему ROWNUM = 1 обеспечивает возвращение первой строки, в том, что Oracle для определения наличия строк в таблице приходится извлекать, по крайней мере, одну из них. Внимательно проанализируйте предыдущий процесс, подставив 1 вместо 5, и вы поймете, почему для возвращения одной строки можно в качестве условия задавать ROWNUM = 1.

Возвращение n случайных записей таблицы

Задача

Требуется возвратить некоторое число записей таблицы, выбранных случайным образом. Необходимо так изменить следующее выражение, чтобы при каждом последующем выполнении оно возвращало разные пять строк:

```
select ename, job
  from emp
```

Решение

Возьмите любую встроенную функцию, возвращающую случайные значения, которую поддерживает ваша СУБД, и примените ее в операторе ORDER BY, чтобы сортировать строки в случайном порядке. Затем с помощью описанной в предыдущем рецепте техники ограничьте число возвращаемых строк.

DB2

Используйте встроенную функцию RAND в сочетании с ORDER BY и FETCH:

```
1 select ename,job
2   from emp
3  order by rand() fetch first 5 rows only
```

MySQL

Используйте встроенную функцию RAND в сочетании с LIMIT и ORDER BY:

```
1 select ename,job
2   from emp
3  order by rand() limit 5
```

PostgreSQL

Используйте встроенную функцию RANDOM в сочетании с LIMIT и ORDER BY:

```
1 select ename,job
2   from emp
3  order by random() limit 5
```

Oracle

Используйте встроенную функцию VALUE, которая находится во встроенном пакете DBMS_RANDOM, в сочетании с ORDER BY и встроенную функцию ROWNUM:

```
1 select *
2   from (
3  select ename, job
4    from emp
6   order by dbms_random.value()
7         )
8   where rownum <= 5
```

SQL Server

Возвращение случайного результирующего множества обеспечит использование встроенной функции NEWID в сочетании с TOP и ORDER BY:

```
1  select top 5 ename,job
2    from emp
3   order by newid()
```

Обсуждение

Оператор ORDER BY может принимать возвращаемое функцией значение и использовать его для изменения порядка расположения элементов результирующего набора. Все предлагаемые решения ограничивают число возвращаемых строк *после* выполнения функции в операторе ORDER BY. Пользователям иных СУБД полезно рассмотреть решение Oracle, потому что на его примере можно (на концептуальном уровне) понять то, что в других решениях происходит «за кадром».

Важно четко различать, что происходит при использовании в ORDER BY функции и числовой константы. При задании в операторе ORDER BY числовой константы сортировка осуществляется по столбцу с заданным в списке SELECT порядковым номером. Когда в ORDER BY задается функция, сортировке подвергается результат, возвращаемый функцией для каждой строки.

Поиск значений NULL

Задача

Требуется найти все строки, имеющие в заданном столбце NULL (неопределенное) значение.

Решение

Чтобы выяснить, является ли значение NULL, необходимо использовать оператор IS NULL:

```
1 select *
2   from emp
3  where comm is null
```

Обсуждение

NULL никогда не бывает равен или не равен ни одному значению, даже самому себе, поэтому с помощью операторов = или ! = нельзя определить, равно ли значение NULL или не равно. Для проверки наличия в строке значений NULL должен использоваться оператор IS NULL. Кроме того, с помощью оператора IS NOT NULL можно выбрать строки, не содержащие NULL значения в заданном столбце.

Преобразование значений NULL в не-NULL значения

Задача

Имеются строки, содержащие NULL значения, и требуется возвратить не-NULL значения вместо имеющихся NULL.

Решение

Чтобы подставить не-NULL значение вместо NULL, используйте функцию COALESCE:

```
1 select coalesce(comm,0)
2   from emp
```

Обсуждение

Функция COALESCE принимает в качестве аргументов одно или более значений. Функция возвращает первое не-NULL значение из списка. В данном решении, если значение COMM NULL, то возвращается ноль. В противном случае возвращается значение COMM.

При работе с NULL значениями лучше всего пользоваться преимуществами встроенных функций, предоставляемых вашей СУБД. Довольно часто несколько подходов могут обеспечить одинаковый результат для данной задачи. COALESCE применима во всех СУБД, как и CASE:

```
select case
       when comm is null then 0
       else comm
       end
  from emp
```

Хотя CASE и позволяет производить преобразование NULL значений в не-NULL значения, использование COALESCE, как видно из примеров, отличается простотой и лаконичностью.

Поиск по шаблону

Задача

Требуется выбрать строки, соответствующие определенной подстроке или шаблону. Рассмотрим следующий запрос и результирующее множество:

```
select ename, job
  from emp
 where deptno in (10,20)
```

```
ENAME      JOB
---------- ---------
SMITH      CLERK
JONES      MANAGER
CLARK      MANAGER
SCOTT      ANALYST
KING       PRESIDENT
ADAMS      CLERK
FORD       ANALYST
MILLER     CLERK
```

Из служащих отделов 10 и 20 требуется выбрать только тех, в имени которых встречается буква «I» или чье название должности заканчивается на «ER»:

```
ENAME      JOB
---------- ---------
SMITH      CLERK
JONES      MANAGER
CLARK      MANAGER
KING       PRESIDENT
MILLER     CLERK
```

Решение

Используйте оператор LIKE в сочетании с оператором подстановки SQL («%»):

```
1 select ename, job
2   from emp
3  where deptno in (10,20)
4    and (ename like '%I%' or job like '%ER')
```

Обсуждение

При использовании в операции сопоставления с шаблоном LIKE оператор «%» выбирает любую последовательность символов. Во многих реализациях SQL также предоставляется оператор подчеркивания («_») для выбора одного символа. Возвращение любой строки, содержащей «I» (в любом месте), обеспечивается заключением шаблона поиска «I» в операторы «%». Если шаблон поиска не заключен в «%», результаты запроса зависят от местоположения «%». Например, чтобы найти названия должностей, заканчивающиеся на «ER», оператор «%» ставится перед «ER». Если требуется найти все должности, начинающиеся с «ER», «%» должен следовать после «ER».

2

Сортировка результатов запроса

Данная глава посвящена настройке представления результатов запросов. Понимая, как управлять и изменять результирующие множества, можно обеспечивать более удобный для чтения и понятный вывод данных.

Возвращение результатов запроса в заданном порядке

Задача

Требуется представить имена, должности и заработные платы служащих 10-го отдела и упорядочить их соответственно заработным платам (от наименьшей к наибольшей). Необходимо получить следующее результирующее множество:

```
ENAME      JOB              SAL
---------- --------- ----------
MILLER     CLERK           1300
CLARK      MANAGER         2450
KING       PRESIDENT       5000
```

Решение

Используйте оператор ORDER BY:

```
1 select ename,job,sal
2   from emp
3  where deptno = 10
4  order by sal asc
```

Обсуждение

Оператор ORDER BY позволяет упорядочивать строки результирующего множества. В разделе «Решение» строки сортируются по столбцу SAL в возрастающем порядке. По умолчанию ORDER BY осуществляет сортировку по возрастанию, поэтому оператор ASC является необязательным. Чтобы обеспечить сортировку по убыванию, используется оператор DESC:

```
select ename,job,sal
  from emp
 where deptno = 10
 order by sal desc
```

```
ENAME      JOB              SAL
---------- --------- ----------
KING       PRESIDENT       5000
CLARK      MANAGER         2450
MILLER     CLERK           1300
```

Имя столбца, по которому должна проводиться сортировка, задавать необязательно. Можно указать порядковый номер столбца. Нумерация столбцов в списке оператора SELECT начинается с 1 и осуществляется в направлении слева направо. Например:

```
select ename,job,sal
  from emp
 where deptno = 10
 order by 3 desc
```

```
ENAME      JOB              SAL
---------- --------- ----------
KING       PRESIDENT       5000
CLARK      MANAGER         2450
MILLER     CLERK           1300
```

Число 3 в операторе ORDER BY данного примера соответствует третьему столбцу списка оператора SELECT, т. е. столбцу SAL.

Сортировка по нескольким полям

Задача

Требуется сортировать строки таблицы EMP сначала по столбцу DEPTNO по возрастанию, а затем по заработным платам по убыванию. Необходимо получить следующее результирующее множество:

```
     EMPNO     DEPTNO        SAL ENAME      JOB
---------- ---------- ---------- ---------- ---------
      7839         10       5000 KING       PRESIDENT
      7782         10       2450 CLARK      MANAGER
      7934         10       1300 MILLER     CLERK
      7788         20       3000 SCOTT      ANALYST
```

```
7902        20       3000 FORD       ANALYST
7566        20       2975 JONES      MANAGER
7876        20       1100 ADAMS      CLERK
7369        20        800 SMITH      CLERK
7698        30       2850 BLAKE      MANAGER
7499        30       1600 ALLEN      SALESMAN
7844        30       1500 TURNER     SALESMAN
7521        30       1250 WARD       SALESMAN
7654        30       1250 MARTIN     SALESMAN
7900        30        950 JAMES      CLERK
```

Решение

В операторе ORDER BY через запятую перечислите столбцы, по которым должна проводиться сортировка:

```
1 select empno,deptno,sal,ename,job
2   from emp
3  order by deptno, sal desc
```

Обсуждение

Старшинство столбцов, по которым осуществляется сортировка, в операторе ORDER BY определяется слева направо. Если столбец задается его порядковым номером в списке оператора SELECT, то это число не должно превышать количества элементов в списке SELECT. Можно проводить сортировку по столбцу, не входящему в список SELECT, но для этого необходимо явно указать его имя. Тем не менее при использовании в запросе операторов GROUP BY или DISTINCT сортировка может осуществляться только по столбцам из списка оператора SELECT.

Сортировка по подстрокам

Задача

Требуется сортировать результаты запроса по частям строки. Например, имена и должности служащих, возвращенные из таблицы EMP, должны быть упорядочены по последним двум символам поля должности. Необходимо получить следующее результирующее множество:

```
ENAME      JOB
---------- ---------
KING       PRESIDENT
SMITH      CLERK
ADAMS      CLERK
JAMES      CLERK
MILLER     CLERK
JONES      MANAGER
CLARK      MANAGER
BLAKE      MANAGER
ALLEN      SALESMAN
```

```
MARTIN     SALESMAN
WARD       SALESMAN
TURNER     SALESMAN
SCOTT      ANALYST
FORD       ANALYST
```

Решение

DB2, MySQL, Oracle и PostgreSQL

Используйте в операторе ORDER BY функцию SUBSTR:

```
select ename,job
  from emp
 order by substr(job,length(job)-2)
```

SQL Server

Используйте в операторе ORDER BY функцию SUBSTRING:

```
select ename,job
  from emp
 order by substring(job,len(job)-2,2)
```

Обсуждение

Используя функции работы с подстроками конкретной СУБД, можно без труда проводить сортировку по любой части строки. Чтобы сортировать по последним двум символам строки, находим конец строки (что соответствует ее длине) и вычитаем 2. Таким образом, начальная позиция будет располагаться на предпоследнем символе строки. Теперь выбираем все символы после этой позиции. SQL Server требует задания для SUBSTRING третьего параметра, определяющего количество выбираемых символов. В данном примере подходит любое число, большее или равное 2.

Сортировка смешанных буквенно-цифровых данных

Задача

Требуется сортировать смешанные буквенно-цифровые данные по числовой или символьной части. Рассмотрим следующее представление:

```
create view V
as
select ename||' '||deptno as data
  from emp

select * from V

DATA
-------------
SMITH 20
ALLEN 30
```

```
WARD 30
JONES 20
MARTIN 30
BLAKE 30
CLARK 10
SCOTT 20
KING 10
TURNER 30
ADAMS 20
JAMES 30
FORD 20
MILLER 10
```

Требуется сортировать результаты по DEPTNO или ENAME. Сортировка по DEPTNO обеспечивает следующее результирующее множество:

```
DATA
----------
CLARK 10
KING 10
MILLER 10
SMITH 20
ADAMS 20
FORD 20
SCOTT 20
JONES 20
ALLEN 30
BLAKE 30
MARTIN 30
JAMES 30
TURNER 30
WARD 30
```

Сортировка по ENAME обеспечивает следующее результирующее множество:

```
DATA
---------
ADAMS 20
ALLEN 30
BLAKE 30
CLARK 10
FORD 20
JAMES 30
JONES 20
KING 10
MARTIN 30
MILLER 10
SCOTT 20
SMITH 20
TURNER 30
WARD 30
```

Решение

Oracle и PostgreSQL

Вносим коррективы в строку, подлежащую сортировке, с помощью функций REPLACE (заместить) и TRANSLATE (переместить):

```
/* СОРТИРУЕМ ПО DEPTNO */

1 select data
2   from V
3  order by replace(data,
4           replace(
5         translate(data,'0123456789','##########'),'#',''),'')

/* СОРТИРУЕМ ПО ENAME */

1 select data
2   from emp
3  order by replace(
4           translate(data,'0123456789','##########'),'#','')
```

DB2

В DB2 неявное преобразование типов более строгое, чем в Oracle или PostgreSQL, поэтому чтобы представление V было корректным, необходимо привести DEPTNO к типу CHAR. В этом решении мы не будем повторно создавать V, а просто используем вложенный запрос. Функции REPLACE и TRANSLATE используются аналогично тому, как это делалось в решении для Oracle и PostrgreSQL, но в данном случае порядок аргументов TRANSLATE немного другой:

```
/* СОРТИРУЕМ ПО DEPTNO */

1  select *
2    from (
3  select ename||' '||cast(deptno as char(2)) as data
4    from emp
5         ) v
6   order by replace(data,
7             replace(
8           translate(data,'##########','0123456789'),'#',''),'')

/* СОРТИРУЕМ ПО ENAME */

1  select *
2    from (
3  select ename||' '||cast(deptno as char(2)) as data
4    from emp
5         ) v
6   order by replace(
7             translate(data,'##########','0123456789'),'#','')
```

MySQL и SQL Server

В настоящее время данные платформы не поддерживают функцию TRANSLATE, таким образом, решения для этой задачи нет.

Обсуждение

Функции TRANSLATE и REPLACE удаляют из каждой строки числа или символы соответственно, что обеспечивает возможность сортировки по данным одного или другого типа. Значения, переданные в ORDER BY, показаны в результатах следующего запроса (в качестве примера используется решение Oracle, поскольку аналогичная техника применима в продуктах всех трех производителей; решение DB2 отличается только порядком параметров, передаваемых в TRANSLATE):

```
select data,
       replace(data,
       replace(
     translate(data,'0123456789','##########'),'#',''),'') nums,
       replace(
     translate(data,'0123456789','##########'),'#','') chars
  from V
```

```
DATA         NUMS   CHARS
------------ ------ ----------
SMITH 20     20     SMITH
ALLEN 30     30     ALLEN
WARD 30      30     WARD
JONES 20     20     JONES
MARTIN 30    30     MARTIN
BLAKE 30     30     BLAKE
CLARK 10     10     CLARK
SCOTT 20     20     SCOTT
KING 10      10     KING
TURNER 30    30     TURNER
ADAMS 20     20     ADAMS
JAMES 30     30     JAMES
FORD 20      20     FORD
MILLER 10    10     MILLER
```

Обработка значений NULL при сортировке

Задача

Требуется сортировать выборку из таблицы EMP по столбцу COMM, но это поле может содержать NULL значения. Необходим способ задания, указывающий, где в результате сортировки должны располагаться NULL значения – после строк с определенными значениями:

```
ENAME             SAL       COMM
---------- ---------- ----------
TURNER           1500          0
```

```
ALLEN            1600        300
WARD             1250        500
MARTIN           1250       1400
SMITH             800
JONES            2975
JAMES             950
MILLER           1300
FORD             3000
ADAMS            1100
BLAKE            2850
CLARK            2450
SCOTT            3000
KING             5000
```

или до них:

```
ENAME             SAL       COMM
---------- ---------- ----------
SMITH             800
JONES            2975
CLARK            2450
BLAKE            2850
SCOTT            3000
KING             5000
JAMES             950
MILLER           1300
FORD             3000
ADAMS            1100
MARTIN           1250       1400
WARD             1250        500
ALLEN            1600        300
TURNER           1500          0
```

Решение

В зависимости от того, как должны быть представлены данные (и как конкретная СУБД сортирует значения NULL), столбцы, допускающие неопределенные значения, можно сортировать по возврастанию или по убыванию:

```
1 select ename,sal,comm
2   from emp
3  order by 3
```

```
1 select ename,sal,comm
2   from emp
3  order by 3 desc
```

Такое решение привело бы к сортировке столбца, не содержащего неопределенные значения, в убывающем или возврастающем порядке. Если необходимо сортировать неопределенные значения иначе, чем определенные, например требуется расположить определенные значения по убыванию или возрастанию, а все значения NULL вывести по-

сле них, то можно использовать выражение CASE, которое обеспечит сортировку столбца по условию.

DB2, MySQL, PostgreSQL и SQL Server

Выражение CASE позволяет отметить строку со значением NULL специальным «флагом». Идея состоит в том, чтобы установить флаг с двумя значениями, одно из которых представляет неопределенные значения (NULL), а другое – определенные (не-NULL). Сделав это, просто добавляем столбец с флагом в оператор ORDER BY. Такой прием позволяет без труда размещать строки с неопределенными значениями в начале или конце списка без смешения их со строками, содержащими определенные значения:

```
/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО ВОЗВРАСТАНИЮ, ПОСЛЕ НИХ
РАСПОЛАГАЮТСЯ ВСЕ СТРОКИ С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ */

1  select ename,sal,comm
2    from (
3  select ename,sal,comm,
4         case when comm is null then 0 else 1 end as is null
5    from emp
6         ) x
7   order by is_null desc,comm
```

```
ENAME    SAL       COMM
------ ----- ----------
TURNER  1500          0
ALLEN   1600        300
WARD    1250        500
MARTIN  1250       1400
SMITH    800
JONES   2975
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
BLAKE   2850
CLARK   2450
SCOTT   3000
KING    5000
```

```
/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО УБЫВАНИЮ, ПОСЛЕ НИХ
РАСПОЛАГАЮТСЯ ВСЕ СТРОКИ С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ */

1  select ename,sal,comm
2    from (
3  select ename,sal,comm,
4         case when comm is null then 0 else 1 end as is_null
5    from emp
6         ) x
7   order by is_null desc,comm desc
```

```
ENAME    SAL       COMM
------ ----- ----------
MARTIN  1250       1400
WARD    1250        500
ALLEN   1600        300
TURNER  1500          0
SMITH    800
JONES   2975
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
BLAKE   2850
CLARK   2450
SCOTT   3000
KING    5000
```

```
/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО ВОЗВРАСТАНИЮ, ВСЕ СТРОКИ
С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ РАСПОЛАГАЮТСЯ ПЕРЕД НИМИ */

1  select ename,sal,comm
2    from (
3  select ename,sal,comm,
4         case when comm is null then 0 else 1 end as is_null
5    from emp
6         ) x
7   order by is_null,comm
```

```
ENAME    SAL       COMM
------ ----- ----------
SMITH    800
JONES   2975
CLARK   2450
BLAKE   2850
SCOTT   3000
KING    5000
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
TURNER  1500          0
ALLEN   1600        300
WARD    1250        500
MARTIN  1250       1400
```

```
/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО УБЫВАНИЮ, ВСЕ СТРОКИ
С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ РАСПОЛАГАЮТСЯ ПЕРЕД НИМИ */

1  select ename,sal,comm
2    from (
3  select ename,sal,comm,
4         case when comm is null then 0 else 1 end as is_null
```

```
5    from emp
6         ) x
7   order by is_null,comm desc
```

```
ENAME    SAL       COMM
------ ----- ----------
SMITH    800
JONES   2975
CLARK   2450
BLAKE   2850
SCOTT   3000
KING    5000
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
MARTIN  1250       1400
WARD    1250        500
ALLEN   1600        300
TURNER  1500          0
```

Oracle

Пользователи Oracle 8*i* Database и более ранних версий могут применять решения для других платформ. Пользователям Oracle 9*i* Database и более поздних версий доступны расширения NULLS FIRST и NULLS LAST оператора ORDER BY, позволяющие располагать строки с неопределенными значениями (NULL) в начале или конце итогового списка независимо от порядка сортировки строк с определенными значениями:

```
/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО ВОЗРАСТАНИЮ,
ВСЕ СТРОКИ С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ РАСПОЛАГАЮТСЯ ПОСЛЕ НИХ */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm nulls last
```

```
ENAME    SAL       COMM
------ ----- ----------
TURNER  1500          0
ALLEN   1600        300
WARD    1250        500
MARTIN  1250       1400
SMITH    800
JONES   2975
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
BLAKE   2850
CLARK   2450
SCOTT   3000
KING    5000
```

```
/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО УБЫВАНИЮ,
ВСЕ СТРОКИ С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ РАСПОЛАГАЮТСЯ ПОСЛЕ НИХ */

1 select ename,sal,comm
2   from emp
3  order by comm desc nulls last

ENAME    SAL       COMM
------ ----- ----------
MARTIN  1250       1400
WARD    1250        500
ALLEN   1600        300
TURNER  1500          0
SMITH    800
JONES   2975
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
BLAKE   2850
CLARK   2450
SCOTT   3000
KING    5000


/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО ВОЗРАСТАНИЮ,
ВСЕ СТРОКИ С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ РАСПОЛАГАЮТСЯ ПЕРЕД НИМИ */

1 select ename,sal,comm
2   from emp
3  order by comm nulls first

ENAME    SAL       COMM
------ ----- ----------
SMITH    800
JONES   2975
CLARK   2450
BLAKE   2850
SCOTT   3000
KING    5000
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
TURNER  1500          0
ALLEN   1600        300
WARD    1250        500
MARTIN  1250       1400

/* ОПРЕДЕЛЕННЫЕ ЗНАЧЕНИЯ COMM СОРТИРУЮТСЯ ПО УБЫВАНИЮ,
ВСЕ СТРОКИ С НЕОПРЕДЕЛЕННЫМИ ЗНАЧЕНИЯМИ РАСПОЛАГАЮТСЯ ПЕРЕД НИМИ */
```

```
1 select ename,sal,comm
2   from emp
3  order by comm desc nulls first
```

```
ENAME    SAL       COMM
------ ----- ----------
SMITH    800
JONES   2975
CLARK   2450
BLAKE   2850
SCOTT   3000
KING    5000
JAMES    950
MILLER  1300
FORD    3000
ADAMS   1100
MARTIN  1250       1400
WARD    1250        500
ALLEN   1600        300
TURNER  1500          0
```

Обсуждение

Если ваша СУБД не предоставляет средства для выноса значений NULL в начало или конец результирующего списка без изменения определенных (не-NULL) значений столбца (как это делает Oracle), то необходимо использовать вспомогательный столбец.

На момент написания данной книги пользователи DB2 могут использовать NULLS FIRST и NULLS LAST в подоператоре ORDER BY оператора OVER в оконных функциях, но не в операторе ORDER BY для всего результирующего множества.

Назначение этого дополнительного столбца (только в запросе, не в таблице) состоит в идентификации значений NULL, обеспечивающей возможность их выноса в начало или конец выводимого списка. Следующий запрос возвращает результирующее множество для вложенного запроса X (решение не Oracle):

```
select ename,sal,comm,
       case when comm is null then 0 else 1 end as is_null
  from emp
```

```
ENAME    SAL       COMM    IS_NULL
------ ----- ---------- ----------
SMITH    800                   0
ALLEN   1600        300         1
WARD    1250        500         1
JONES   2975                    0
MARTIN  1250       1400         1
BLAKE   2850                    0
CLARK   2450                    0
```

```
SCOTT   3000                    0
KING    5000                    0
TURNER  1500          0         1
ADAMS   1100                    0
JAMES    950                    0
FORD    3000                    0
MILLER  1300                    0
```

Используя значения столбца IS_NULL, можно без труда вынести строки со значениями NULL в начало или конец списка, не вмешиваясь в сортировку COMM.

Сортировка по зависящему от данных ключу

Задача

Требуется провести сортировку с применением некоторой условной логики. Например, если значение JOB – «SALESMAN» (агент по продажам), сортировка должна осуществляться по столбцу COMM; в противном случае сортируем по SAL. Необходимо получить следующее результирующее множество:

```
ENAME             SAL JOB             COMM
---------- ---------- --------- ----------
TURNER            1500 SALESMAN           0
ALLEN             1600 SALESMAN         300
WARD              1250 SALESMAN         500
SMITH              800 CLERK
JAMES              950 CLERK
ADAMS             1100 CLERK
MARTIN            1250 SALESMAN        1400
MILLER            1300 CLERK
CLARK             2450 MANAGER
BLAKE             2850 MANAGER
JONES             2975 MANAGER
SCOTT             3000 ANALYST
FORD              3000 ANALYST
KING              5000 PRESIDENT
```

Решение

Используйте выражение CASE в операторе ORDER BY:

```
1 select ename,sal,job,comm
2   from emp
3  order by case when job = 'SALESMAN' then comm else sal end
```

Обсуждение

Для динамического изменения принципа сортировки результатов запроса может использоваться выражение CASE. Значения, передаваемые в оператор ORDER BY, выглядят следующим образом:

```
select ename,sal,job,comm,
       case when job = 'SALESMAN' then comm else sal end as ordered
  from emp
 order by 5
```

```
ENAME             SAL JOB             COMM    ORDERED
---------- ---------- --------- ---------- ----------
TURNER            1500 SALESMAN           0          0
ALLEN             1600 SALESMAN         300        300
WARD              1250 SALESMAN         500        500
SMITH              800 CLERK                        800
JAMES              950 CLERK                        950
ADAMS             1100 CLERK                       1100
MARTIN            1250 SALESMAN        1300       1300
MILLER            1300 CLERK                       1300
CLARK             2450 MANAGER                     2450
BLAKE             2850 MANAGER                     2850
JONES             2975 MANAGER                     2975
SCOTT             3000 ANALYST                     3000
FORD              3000 ANALYST                     3000
KING              5000 PRESIDENT                   5000
```

3

Работа с несколькими таблицами

Данная глава знакомит с использованием объединений и операций над множествами для комбинирования данных нескольких таблиц. Объединения лежат в основе SQL. Операции над множествами также имеют очень большое значение. Путь к овладению сложными запросами, о которых рассказывается в последующих главах книги начинается здесь, с объединений и операций над множествами.

Размещение одного набора строк под другим

Задача

Требуется извлечь данные, хранящиеся в нескольких таблицах, размещая одно результирующее множество под другим. Необязательно, чтобы у таблиц был общий ключ, но типы их столбцов должны совпадать. Например, необходимо вывести на экран имена и номер отдела служащих 10-го отдела, хранящиеся в таблице EMP, а также названия и номера всех отделов из таблицы DEPT. Должно быть получено следующее результирующее множество:

```
ENAME_AND_DNAME     DEPTNO
--------------- ----------
CLARK                   10
KING                    10
MILLER                  10
----------
ACCOUNTING              10
RESEARCH                20
SALES                   30
OPERATIONS              40
```

Решение

Объединение строк из нескольких таблиц осуществляется с помощью операции над множествами UNION ALL (объединить все):

```
1  select ename as ename_and_dname, deptno
2    from emp
3   where deptno = 10
4   union all
5  select '----------', null
6    from t1
7   union all
8  select dname, deptno
9    from dept
```

Обсуждение

Оператор UNION ALL объединяет строки из нескольких источников в одно результирующее множество. Как и для всех операций над множествами, число и тип элементов, перечисленных в операторах SELECT, должны совпадать. Например, оба приведенных ниже запроса дадут сбой:

```
select deptno   |  select deptno, dname
  from dept     |    from dept
 union all      |   union
select ename    |  select deptno
  from emp      |    from emp
```

Важно отметить, что UNION ALL включит и дубликаты, если они есть. Если дубликаты не нужны, используется оператор UNION. Например, в результате объединения оператором UNION столбцов EMP.DEPTNO и DEPT.DEPTNO возвращаются только четыре строки:

```
select deptno
  from emp
 union
select deptno
  from dept

    DEPTNO
----------
        10
        20
        30
        40
```

Использование UNION вместо UNION ALL, вероятнее всего, приведет к операции сортировки с целью устранения дубликатов. Об этом необходимо помнить при работе с большими результирующими множествами. Операция UNION примерно эквивалентна следующему запросу, в котором к результату операции UNION ALL применяется ключевое слово DISTINCT:

```
select distinct deptno
  from (
select deptno
  from emp
 union all
select deptno
  from dept
       )
```

```
    DEPTNO
----------
        10
        20
        30
        40
```

DISTINCT следует применять в запросах только в случае необходимости. То же правило относится и к операции UNION: не используйте ее вместо UNION ALL, если в этом нет необходимости.

Объединение взаимосвязанных строк

Задача

Требуется возвратить строки нескольких таблиц, объединяя их по известному общему столбцу или по столбцам с общими значениями. Например, необходимо вывести имена всех служащих 10-го отдела, а также местонахождение отдела для каждого служащего, но эти данные хранятся в двух разных таблицах. Должно быть получено следующее результирующее множество:

```
ENAME      LOC
---------- -------------
CLARK      NEW YORK
KING       NEW YORK
MILLER     NEW YORK
```

Решение

Объедините таблицу EMP с таблицей DEPT по столбцу DEPTNO:

```
1 select e.ename, d.loc
2   from emp e, dept d
3  where e.deptno = d.deptno
4    and e.deptno = 10
```

Обсуждение

Решение является примером *объединения* (*join*) или, чтобы быть более точным, *эквиобъединения* (*equi-join*), которое является разновидностью *внутреннего объединения* (*inner join*). Объединение – это операция, в результате которой строки двух таблиц соединяются в одну. Эк-

виобъединение – это объединение, в котором условием объединения является равенство (например, равенство номеров отделов). Внутреннее объединение – исходный тип объединения; возвращаемые строки содержат данные всех таблиц.

Концептуально при выполнении объединения сначала создается декартово произведение (все возможные сочетания строк) таблиц, перечисленных в конструкции FROM, как показано ниже:

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
  from emp e, dept d
 where e.deptno = 10
```

```
ENAME      LOC           EMP_DEPTNO DEPT_DEPTNO
---------- ------------- ---------- -----------
CLARK      NEW YORK              10          10
KING       NEW YORK              10          10
MILLER     NEW YORK              10          10
CLARK      DALLAS                10          20
KING       DALLAS                10          20
MILLER     DALLAS                10          20
CLARK      CHICAGO               10          30
KING       CHICAGO               10          30
MILLER     CHICAGO               10          30
CLARK      BOSTON                10          40
KING       BOSTON                10          40
MILLER     BOSTON                10          40
```

Каждому служащему из таблицы EMP (10-го отдела) ставится в соответствие каждый отдел из таблицы DEPT. Затем выражение предиката WHERE, включающее столбцы e.deptno и d.deptno (объединение), ограничивает результирующее множество таким образом, что возвращены будут только те строки, для которых значения EMP.DEPTNO и DEPT.DEPTNO равны:

```
select e.ename, d.loc,
       e.deptno as emp_deptno,
       d.deptno as dept_deptno
  from emp e, dept d
 where e.deptno = d.deptno
   and e.deptno = 10
```

```
ENAME      LOC           EMP_DEPTNO DEPT_DEPTNO
---------- ------------- ---------- -----------
CLARK      NEW YORK              10          10
KING       NEW YORK              10          10
MILLER     NEW YORK              10          10
```

Альтернативное решение – явное использование оператора JOIN (объединить) (ключевое слово «INNER» (внутренний) является необязательным):

```
select e.ename, d.loc
  from emp e inner join dept d
    on (e.deptno = d.deptno)
 where e.deptno = 10
```

Если вы предпочитаете реализовывать логику объединения не в WHERE, а в конструкции FROM, используйте JOIN. Оба варианта допускаются стандартом ANSI и могут применяться для всех последних версий СУБД, обсуждаемых в данной книге.

Поиск одинаковых строк в двух таблицах

Задача

Требуется найти общие строки в двух таблицах, но объединение возможно по нескольким столбцам. Например, рассмотрим такое представление V:

```
create view V
as
select ename,job,sal
  from emp
 where job = 'CLERK'

select * from V
```

```
ENAME      JOB              SAL
---------- --------- ----------
SMITH      CLERK             800
ADAMS      CLERK            1100
JAMES      CLERK             950
MILLER     CLERK            1300
```

Представление V содержит только строки для клерков. Но в него включены не все столбцы EMP. Требуется выбрать значения столбцов EMPNO, ENAME, JOB, SAL и DEPTNO для всех служащих таблицы EMP соответственно строкам представления V. Результирующее множество должно быть таким:

```
   EMPNO ENAME      JOB              SAL     DEPTNO
-------- ---------- --------- ---------- ----------
    7369 SMITH      CLERK             800         20
    7876 ADAMS      CLERK            1100         20
    7900 JAMES      CLERK             950         30
    7934 MILLER     CLERK            1300         10
```

Решение

Проведите объединение таблиц по всем столбцам, необходимым для формирования требуемого результата. Или используйте операцию над множествами INTERSECT (пересекать), чтобы избежать объединения и возвратить пересечение (общие строки) двух таблиц.

MySQL и SQL Server[1]

Объедините таблицу EMP с представлением V, используя несколько условий объединения:

```
1  select e.empno,e.ename,e.job,e.sal,e.deptno
2    from emp e, V
3   where e.ename = v.ename
4     and e.job   = v.job
5     and e.sal   = v.sal
```

Или аналогичное объединение можно осуществить посредством оператора JOIN:

```
1  select e.empno,e.ename,e.job,e.sal,e.deptno
2    from emp e join V
3      on (    e.ename = v.ename
4          and e.job   = v.job
5          and e.sal   = v.sal )
```

DB2, Oracle и PostgreSQL

Решение для MySQL и SQL Server подходит и для DB2, Oracle и PostgreSQL. Его следует применять, если требуется возвратить значения представления V.

Если нет необходимости возвращать столбцы представления V, можно использовать операцию над множествами INTERSECT в сочетании с предикатом IN:

```
1  select empno,ename,job,sal,deptno
2    from emp
3   where (ename,job,sal) in (
4    select ename,job,sal from emp
5    intersect
6    select ename,job,sal from V
7   )
```

Обсуждение

Чтобы получить требуемый результат, необходимо правильно выбирать столбцы для объединения. Это особенно важно, когда значения строк в одних столбцах могут быть одинаковыми, а в других – нет.

Операция над множествами INTERSECT возвращает строки, общие для обоих источников. При использовании INTERSECT должно сравниваться одинаковое количество элементов одного типа из двух таблиц. При работе с операциями над множествами необходимо помнить, что по умолчанию строки-дубликаты не возвращаются.

---

[1] В SQL Server 2005 работает вариант, описанный для DB2 и PostgreSQL, т. к. в нем уже поддерживается конструкция INTERSECT. – *Примеч. науч. ред.*

Извлечение из одной таблицы значений, которых нет в другой таблице

Задача

Требуется в одной таблице, назовем ее исходной, найти значения, которых нет в другой таблице, назовем ее целевой. Например, необходимо выяснить, каких отделов (если таковые имеются), представленных в таблице DEPT, нет в таблице EMP. В примере базы данных в таблице DEPT есть DEPTNO 40, которого нет в EMP; таким образом, результирующее множество должно быть следующим:

```
    DEPTNO
----------
        40
```

Решение

Для решения этой задачи очень полезны функции, осуществляющие операцию вычитания множеств. DB2, PostgreSQL и Oracle поддерживают операции вычитания множеств. Если ваша СУБД не поддерживает таких функций, используйте подзапрос, как показано для MySQL и SQL Server.

DB2 and PostgreSQL

Используйте операцию над множествами EXCEPT (за исключением):

```
1 select deptno from dept
2 except
3 select deptno from emp
```

Oracle

Используйте операцию над множествами MINUS (минус):

```
1 select deptno from dept
2 minus
3 select deptno from emp
```

MySQL и SQL Server

Используйте подзапрос, возвращающий все значения столбца DEPTNO таблицы EMP. Внешний запрос будет искать в таблице DEPT строки, которых нет среди строк, возвращенных подзапросом:

```
1 select deptno
2   from dept
3  where deptno not in (select deptno from emp)
```

Обсуждение

DB2 и PostgreSQL

Встроенные функции, предоставляемые DB2 и PostgreSQL, существенно упрощают решение поставленной задачи. Оператор EXCEPT принимает первое результирующее множество и удаляет из него все строки, обнаруженные во втором результирующем множестве. Операция очень похожа на вычитание.

Существуют некоторые ограничения на использование операторов над множествами, включая EXCEPT: типы данных и количество сравниваемых значений в списках обоих операторов SELECT должны совпадать. Кроме того, EXCEPT не возвращает дубликаты значений и, в отличие от подзапроса, использующего NOT IN, значения NULL не представляют проблемы (смотрите обсуждение для MySQL и SQL Server). Оператор EXCEPT возвращает строки, полученные в результате верхнего запроса (запроса, предшествующего EXCEPT), которых нет в запросе ниже (запросе, следующим за EXCEPT).

Oracle

Решение для Oracle идентично решению для DB2 и PostgreSQL за исключением того, что в Oracle оператор вычитания множеств называется MINUS, а не EXCEPT. Во всем остальном предыдущее объяснение применимо и к Oracle.

MySQL и SQL Server[1]

Подзапрос возвращает все значения столбца DEPTNO таблицы EMP. Внешний запрос возвращает все значения столбца DEPTNO таблицы DEPT, которых «нет» или которые «не включены» в результирующее множество, возвращенное подзапросом.

Исключение дубликатов является важным аспектом решений для MySQL и SQL Server. Функции EXCEPT и MINUS, используемые в решениях для других платформ, обеспечивают устранение дублирующихся строк из результирующего множества, что гарантирует однократное представление каждого значения столбца DEPTNO. Конечно, это произойдет в любом случае, поскольку DEPTNO является ключевым полем в данных моего примера. Если DEPTNO – не ключевое поле, использование DISTINCT, как в примере ниже, гарантирует, что каждое значение DEPTNO, отсутствующее в EMP, будет выведено всего один раз:

```
select distinct deptno
  from dept
 where deptno not in (select deptno from emp)
```

[1] В SQL Server 2005 работает вариант, описанный для DB2 и PostgreSQL, т. к. в нем уже поддерживается конструкция EXCEPT. – *Примеч. науч. ред.*

При использовании оператора NOT IN не забывайте о значениях NULL. Рассмотрим следующую таблицу NEW_DEPT:

```
create table new_dept(deptno integer)
insert into new_dept values (10)
insert into new_dept values (50)
insert into new_dept values (null)
```

Если попытаться найти в таблице DEPT значения DEPTNO, которых нет в таблице NEW_DEPT, и использовать для этого подзапрос с оператором NOT IN, запрос не возвратит ни одной строки:

```
select *
  from dept
 where deptno not in (select deptno from new_dept)
```

В таблице NEW_DEPT нет значений 20, 30 и 40 столбца DEPTNO, тем не менее они не были возвращены в результате запроса. Причина в значении NULL, присутствующем в таблице NEW_DEPT. Подзапрос возвращает три строки со значениями DEPTNO 10, 50 и NULL. По сути, IN и NOT IN – операции логического ИЛИ. Формируемый ими результат зависит от того, как интерпретируются значения NULL при вычислении логического ИЛИ. Рассмотрим пример использования IN и эквивалентный ему пример с применением оператора OR (ИЛИ):

```
select deptno
  from dept
 where deptno in ( 10,50,null )

 DEPTNO
-------
     10

select deptno
  from dept
 where (deptno=10 or deptno=50 or deptno=null)

DEPTNO
-------
     10
```

Теперь рассмотрим тот же пример с использованием NOT IN и NOT OR:

```
select deptno
  from dept
 where deptno not in ( 10,50,null )

(возвращает пустое множество)

select deptno
  from dept
 where not (deptno=10 or deptno=50 or deptno=null)

(возвращает пустое множество)
```

Как видите, условие DEPTNO NOT IN (10, 50, NULL) эквивалентно:

```
not (deptno=10 or deptno=50 or deptno=null)
```

Вот как вычисляется это выражение в случае, когда DEPTNO 50:

```
not (deptno=10 or deptno=50 or deptno=null)
(false or false or null)
(false or null)
null
```

В SQL выражению «TRUE or NULL» соответствует TRUE, а «FALSE or NULL» – NULL! Полученный результат NULL обеспечит NULL при последующих вычислениях (если только не провести проверку на NULL, применяя технику, представленную в рецепте «Поиск NULL значений» главы 1). Об этом необходимо помнить при использовании предикатов IN и осуществлении вычислений логического ИЛИ, когда присутствуют значения NULL.

Во избежание проблем с NOT IN и значениями NULL применяются связанные подзапросы в сочетании с предикатом NOT EXISTS. Термин «связанные подзапросы» возник потому, что подзапрос использует строки, возвращаемые внешним запросом. Следующий пример является альтернативным решением, в котором строки со значениями NULL не представляют никакой проблемы (возвращаясь к исходному запросу из раздела «Задача»):

```
select d.deptno
  from dept d
 where not exists ( select null
                      from emp e
                     where d.deptno = e.deptno )
    DEPTNO
----------
        40
```

По существу, внешний запрос в данном решении рассматривает все строки таблицы DEPT. С каждой строкой DEPT происходит следующее:

1. Выполняется подзапрос с целью проверки существования данного номера отдела в таблице EMP. Обратите внимание на условие D.DEPTNO = E.DEPTNO, с помощью которого сопоставляются номера отделов из двух таблиц.
2. Если подзапрос возвращает результаты, выражение EXISTS (...) возвращает значение TRUE, а NOT EXISTS (...) соответственно FALSE, и строка, рассматриваемая внешним запросом, не включается в результирующее множество.
3. Если подзапрос не возвращает результатов, выражение NOT EXISTS (...) возвращает TRUE, и рассматриваемая внешним запросом строка включается в результирующее множество (поскольку она соответствует отделу, который еще не представлен в таблице EMP).

При использовании связанного подзапроса с EXISTS/NOT EXISTS не важно, какие элементы перечислены в операторе SELECT подзапроса.

Поэтому я указал NULL, чтобы обратить ваше внимание на объединение в подзапросе, а не на элементы списка SELECT.

Извлечение из таблицы строк, для которых нет соответствия в другой таблице

Задача

В одной из двух таблиц, имеющих общие ключи, требуется найти строки, которых нет в другой таблице. Например, необходимо определить, в каком отделе нет служащих. Результирующее множество должно быть таким:

```
    DEPTNO DNAME          LOC
---------- -------------- -------------
        40 OPERATIONS     BOSTON
```

Поиск отдела, в котором работает каждый из служащих, требует проведения операции эквиобъединения таблиц EMP и DEPT по столбцу DEPTNO. Столбец DEPTNO представляет общее значение для обеих таблиц. К сожалению, эквиобъединение не даст информацию о том, в каком отделе нет служащих, поскольку в результате этой операции будут возвращены все строки, удовлетворяющие условию объединения. А нам нужны только те строки таблицы DEPT, которые не удовлетворяют условию объединения.

Данная задача несколько отличается от той, что была поставлена в предыдущем рецепте, хотя на первый взгляд может показаться такой же. Разница в том, что в предыдущем рецепте формировался только список номеров отделов, не представленных в таблице EMP. Однако, используя настоящий рецепт, можно извлечь и другие столбцы таблицы DEPT и возвратить не только номера отделов.

Решение

Извлеките все строки одной таблицы и строки другой таблицы, которые могут иметь или не иметь соответствия по общему столбцу. Затем оставьте только те строки, которые не имеют соответствия.

DB2, MySQL, PostgreSQL, SQL Server

Используйте внешнее объединение и фильтр для значений NULL (ключевое слово OUTER (внешний) является необязательным):

```
1 select d.*
2   from dept d left outer join emp e
3     on (d.deptno = e.deptno)
4  where e.deptno is null
```

Oracle

Предыдущее решение подойдет пользователям, работающим с Oracle 9*i* Database и более поздними версиями. В качестве альтернативы может использоваться собственный синтаксис Oracle для внешнего объединения:

```
1 select d.*
2   from dept d, emp e
3  where d.deptno = e.deptno (+)
4    and e.deptno is null
```

Этот собственный синтаксис (обратите внимание на использование «+» в скобках) – единственный доступный в Oracle 8*i* Database и более ранних версиях синтаксис внешнего объединения.

Обсуждение

Данное решение обеспечивает внешнее объединение с последующим возвращением только тех строк, которые не имеют соответствия. Операции такого рода иногда называют *антиобъединением* (*anti-join*). Чтобы лучше понять принцип антиобъединения, рассмотрим результирующее множество, получаемое без фильтрации значений NULL:

```
select e.ename, e.deptno as emp_deptno, d.*
  from dept d left join emp e
    on (d.deptno = e.deptno)

ENAME      EMP_DEPTNO     DEPTNO DNAME          LOC
---------- ---------- ---------- -------------- ----------
SMITH              20         20 RESEARCH       DALLAS
ALLEN              30         30 SALES          CHICAGO
WARD               30         30 SALES          CHICAGO
JONES              20         20 RESEARCH       DALLAS
MARTIN             30         30 SALES          CHICAGO
BLAKE              30         30 SALES          CHICAGO
CLARK              10         10 ACCOUNTING     NEW YORK
SCOTT              20         20 RESEARCH       DALLAS
KING               10         10 ACCOUNTING     NEW YORK
TURNER             30         30 SALES          CHICAGO
ADAMS              20         20 RESEARCH       DALLAS
JAMES              30         30 SALES          CHICAGO
FORD               20         20 RESEARCH       DALLAS
MILLER             10         10 ACCOUNTING     NEW YORK
                              40 OPERATIONS     BOSTON
```

Обратите внимание, в столбцах EMP.ENAME и EMP_DEPTNO последней строки отсутствуют значения, потому что в 40-м отделе нет служащих. С помощью предиката WHERE в решении выбираются только строки со значением NULL в столбце EMP_DEPTNO (таким образом, возвращаются только те строки DEPT, которым нет соответствия в EMP).

Независимое добавление объединений в запрос

Задача

Имеется запрос, возвращающий требуемые результаты. Возникает необходимость в дополнительной информации, но при попытке получить ее теряются данные исходного результирующего множества. Например, необходимо получить имена всех служащих, местонахождение отделов, в которых они работают, и даты выдачи им премий. Для выполнения этой задачи существует таблица EMP_BONUS со следующими данными:

```
select * from emp_bonus

     EMPNO RECEIVED          TYPE
---------- ----------- ----------
      7369 14-MAR-2005          1
      7900 14-MAR-2005          2
      7788 14-MAR-2005          3
```

Исходный запрос выглядит следующим образом:

```
select e.ename, d.loc
  from emp e, dept d
 where e.deptno=d.deptno

ENAME      LOC
---------- -------------
SMITH      DALLAS
ALLEN      CHICAGO
WARD       CHICAGO
JONES      DALLAS
MARTIN     CHICAGO
BLAKE      CHICAGO
CLARK      NEW YORK
SCOTT      DALLAS
KING       NEW YORK
TURNER     CHICAGO
ADAMS      DALLAS
JAMES      CHICAGO
FORD       DALLAS
MILLER     NEW YORK
```

К этим результатам требуется добавить даты выдачи премии, но объединение с таблицей EMP_BONUS приведет к тому, что будут возвращены только те строки, которые соответствуют служащим, получившим премию:

```
select e.ename, d.loc,eb.received
  from emp e, dept d, emp_bonus eb
 where e.deptno=d.deptno
   and e.empno=eb.empno
```

```
ENAME      LOC           RECEIVED
---------- ------------- -----------
SCOTT      DALLAS        14-MAR-2005
SMITH      DALLAS        14-MAR-2005
JAMES      CHICAGO       14-MAR-2005
```

А необходимо получить такое результирующее множество:

```
ENAME      LOC           RECEIVED
---------- ------------- -----------
ALLEN      CHICAGO
WARD       CHICAGO
MARTIN     CHICAGO
JAMES      CHICAGO       14-MAR-2005
TURNER     CHICAGO
BLAKE      CHICAGO
SMITH      DALLAS        14-MAR-2005
FORD       DALLAS
ADAMS      DALLAS
JONES      DALLAS
SCOTT      DALLAS        14-MAR-2005
CLARK      NEW YORK
KING       NEW YORK
MILLER     NEW YORK
```

Решение

Для получения дополнительной информации без утраты данных, возвращенных в результате исходного запроса, можно использовать внешнее объединение. Сначала объединим таблицы EMP и DEPT и получим список всех служащих и местонахождений отделов, в которых они работают. Затем выполним внешнее объединение с таблицей EMP_BONUS, чтобы возвратить даты получения премий, если таковые были выданы. Ниже приведен синтаксис для DB2, MySQL, PostgreSQL и SQL Server:

```
1 select e.ename, d.loc, eb.received
2   from emp e join dept d
3     on (e.deptno=d.deptno)
4   left join emp_bonus eb
5     on (e.empno=eb.empno)
6  order by 2
```

Предыдущее решение подходит также для Oracle 9*i* Database и более поздних версий. В качестве альтернативы можно использовать собственный синтаксис Oracle для внешнего объединения, который будет работать и для Oracle 8*i* Database и более ранних версий:

```
1 select e.ename, d.loc, eb.received
2   from emp e, dept d, emp_bonus eb
3  where e.deptno=d.deptno
4    and e.empno=eb.empno (+)
5  order by 2
```

Имитировать внешнее объединение можно с помощью скалярного подзапроса (подзапроса, размещаемого в списке SELECT):

```
1 select e.ename, d.loc,
2        (select eb.received from emp_bonus eb
3          where eb.empno=e.empno) as received
4   from emp e, dept d
5  where e.deptno=d.deptno
6  order by 2
```

Решение с использованием скалярного подзапроса подходит для всех платформ.

Обсуждение

В результате внешнего объединения будут возвращены все строки одной таблицы и соответствующие им строки другой таблицы. Еще один пример такого объединения представлен в предыдущем рецепте. Причина, почему внешнее объединение обеспечивает решение данной задачи, в том, что оно не приводит к удалению строк. Запрос возвратит все строки, которые были бы возвращены без внешнего объединения, а также дополнительные данные из другой таблицы, если таковые имеются.

Для решения задач такого рода также удобно пользоваться скалярными подзапросами, поскольку при этом не приходится менять уже состоявшиеся объединения основного запроса. Скалярный подзапрос – простой способ внести дополнительные данные в запрос, не подвергая опасности текущее результирующее множество. Скалярные подзапросы должны гарантированно возвращать скалярное (одно) значение. Если подзапрос в списке SELECT возвращает более одной строки, будет получена ошибка.

См. также

В главе 14 в разделе «Преобразование скалярного подзапроса в составной подзапрос (Oracle)» показано, как обеспечить возвращение нескольких строк из подзапроса, размещаемого в списке оператора SELECT.

Выявление одинаковых данных в двух таблицах

Задача

Требуется выяснить, имеются ли в двух таблицах или представлениях одинаковые данные (учитываются кардинальность и значения). Рассмотрим следующее представление:

```
create view V
as
select * from emp where deptno != 10
 union all
select * from emp where ename = 'WARD'
```

```
select * from V

EMPNO ENAME  JOB        MGR HIREDATE      SAL  COMM DEPTNO
----- ------ -------- ----- ----------- ----- ----- ------
 7369 SMITH  CLERK     7902 17-DEC-1980   800           20
 7499 ALLEN  SALESMAN  7698 20-FEB-1981  1600   300     30
 7521 WARD   SALESMAN  7698 22-FEB-1981  1250   500     30
 7566 JONES  MANAGER   7839 02-APR-1981  2975           20
 7654 MARTIN SALESMAN  7698 28-SEP-1981  1250  1400     30
 7698 BLAKE  MANAGER   7839 01-MAY-1981  2850           30
 7788 SCOTT  ANALYST   7566 09-DEC-1982  3000           20
 7844 TURNER SALESMAN  7698 08-SEP-1981  1500     0     30
 7876 ADAMS  CLERK     7788 12-JAN-1983  1100           20
 7900 JAMES  CLERK     7698 03-DEC-1981   950           30
 7902 FORD   ANALYST   7566 03-DEC-1981  3000           20
 7521 WARD   SALESMAN  7698 22-FEB-1981  1250   500     30
```

Требуется выяснить, имеются ли в этом представлении такие же данные, как и в таблице EMP. Строка для служащего WARD продублирована, чтобы продемонстрировать, что решение обеспечит выявление не только различных, но и дублированных данных. Сравнение с таблицей EMP дает три строки для служащих 10-го отдела и две строки для служащего WARD. Должно быть получено следующее результирующее множество:

```
EMPNO ENAME   JOB          MGR HIREDATE      SAL  COMM DEPTNO CNT
----- ------- --------- ----- ----------- ----- ----- ------ ---
 7521 WARD    SALESMAN   7698 22-FEB-1981  1250   500     30   1
 7521 WARD    SALESMAN   7698 22-FEB-1981  1250   500     30   2
 7782 CLARK   MANAGER    7839 09-JUN-1981  2450           10   1
 7839 KING    PRESIDENT       17-NOV-1981  5000           10   1
 7934 MILLER  CLERK      7782 23-JAN-1982  1300           10   1
```

Решение

Функции, осуществляющие вычитание множеств (MINUS или EXCEPT, в зависимости от СУБД), существенно упрощают задачу по сравнению таблиц. Если ваша СУБД не поддерживает таких функций, можно использовать связанный подзапрос.

DB2 и PostgreSQL

Чтобы найти совокупность отличий между представлением V и таблицей EMP, используйте операции над множествами EXCEPT и UNION ALL:

```
1  (
2   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
3          count(*) as cnt
4     from V
5    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
6   except
7   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
```

```
 8          count(*) as cnt
 9     from emp
10    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11  )
12    union all
13  (
14   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15          count(*) as cnt
16     from emp
17    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18   except
19   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20          count(*) as cnt
21     from v
22    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23  )
```

Oracle

Чтобы найти совокупность отличий между представлением V и таблицей EMP, используйте операции над множествами MINUS и UNION ALL:

```
 1  (
 2   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
 3          count(*) as cnt
 4     from V
 5    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
 6    minus
 7   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
 8          count(*) as cnt
 9     from emp
10    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
11  )
12     union all
13  (
14   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
15          count(*) as cnt
16     from emp
17    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
18    minus
19   select empno,ename,job,mgr,hiredate,sal,comm,deptno,
20          count(*) as cnt
21     from v
22    group by empno,ename,job,mgr,hiredate,sal,comm,deptno
23  )
```

MySQL и SQL Server

Чтобы найти сочетание строк, которых нет в представлении V, но есть в таблице EMP, и наоборот, используйте связанный подзапрос и операцию UNION ALL:

```
select *
  from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
       e.sal,e.comm,e.deptno, count(*) as cnt
  from emp e
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
       ) e
 where not exists (
select null
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
       v.sal,v.comm,v.deptno, count(*) as cnt
  from v
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
       ) v
  where v.empno    = e.empno
    and v.ename    = e.ename
    and v.job      = e.job
    and v.mgr      = e.mgr
    and v.hiredate = e.hiredate
    and v.sal      = e.sal
    and v.deptno   = e.deptno
    and v.cnt      = e.cnt
    and coalesce(v.comm,0) = coalesce(e.comm,0)
)
 union all
select *
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
       v.sal,v.comm,v.deptno, count(*) as cnt
  from v
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
       ) v
 where not exists (
select null
  from (
select e.empno,e.ename,e.job,e.mgr,e.hiredate,
       e.sal,e.comm,e.deptno, count(*) as cnt
  from emp e
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
       ) e
  where v.empno    = e.empno
    and v.ename    = e.ename
    and v.job      = e.job
    and v.mgr      = e.mgr
    and v.hiredate = e.hiredate
    and v.sal      = e.sal
```

```
52       and v.deptno   = e.deptno
53       and v.cnt      = e.cnt
54       and coalesce(v.comm,0) = coalesce(e.comm,0)
55   )
```

Обсуждение

Несмотря на различие техник, суть всех решений одинакова:

1. Сначала находим в таблице EMP строки, которых нет в представлении V.
2. Затем комбинируем (UNION ALL) эти строки со строками представления V, которых нет в таблице EMP.

Если рассматриваемые таблицы идентичны, то не будет возвращено ни одной строки. Если таблицы отличаются, будут возвращены строки, которые обусловливают различие. Начать сравнение таблиц можно с проверки количества элементов в них, не углубляясь сразу в сопоставление данных. Следующий запрос – простой пример такого сравнения, которое подходит для всех СУБД:

```
select count(*)
  from emp
 union
select count(*)
  from dept
```

```
COUNT(*)
--------
       4
      14
```

Поскольку операция UNION отфильтровывает дубликаты, в случае равного количества элементов в таблицах была бы возвращена всего одна строка. Тот факт, что в данном примере возвращено две строки, означает, что наборы строк таблиц разные.

DB2, Oracle и PostgreSQL

MINUS и EXCEPT работают одинаково, поэтому в данном обсуждении используем EXCEPT. Запросы перед и после операции UNION ALL очень похожи. Таким образом, чтобы понять решение, просто выполним отдельно запрос, предшествующий UNION ALL. Следующее множество получается в результате выполнения строк 1–11, приведенных в разделе «Решение»:

```
(
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from V
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
  except
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
```

```
        count(*) as cnt
    from emp
  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)
```

```
EMPNO ENAME      JOB          MGR HIREDATE      SAL  COMM DEPTNO CNT
----- ---------- --------- ----- ----------- ----- ----- ------ ---
 7521 WARD       SALESMAN   7698 22-FEB-1981  1250   500     30   2
```

Результирующее множество включает строку представления V, которой или нет в таблице EMP, или ее кардинальное число отличается от кардинального числа строки таблицы EMP. В данном случае обнаруживается и возвращается строка-дубликат для служащего WARD. Если вам по-прежнему не до конца понятно, как формируется результирующее множество, выполните каждый из запросов до и после EXCEPT по отдельности. Вы увидите, что единственное отличие между двумя результирующими множествами – значение столбца CNT для служащего WARD из представления V.

Часть запроса после UNION ALL выполняет операцию, являющуюся зеркальным отображением запроса, предшествующего UNION ALL. Этот запрос возвращает строки таблицы EMP, которых нет в представлении V:

```
(
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from emp
   group by empno,ename,job,mgr,hiredate,sal,comm,deptno
   minus
  select empno,ename,job,mgr,hiredate,sal,comm,deptno,
         count(*) as cnt
    from v
  group by empno,ename,job,mgr,hiredate,sal,comm,deptno
)
```

```
EMPNO ENAME      JOB          MGR HIREDATE      SAL  COMM DEPTNO CNT
----- ---------- --------- ----- ----------- ----- ----- ------ ---
 7521 WARD       SALESMAN   7698 22-FEB-1981  1250   500     30   1
 7782 CLARK      MANAGER    7839 09-JUN-1981  2450           10   1
 7839 KING       PRESIDENT       17-NOV-1981  5000           10   1
 7934 MILLER     CLERK      7782 23-JAN-1982  1300           10   1
```

Потом UNION ALL комбинирует результаты и формирует окончательное результирующее множество.

MySQL и SQL Server

Запросы перед и после операции UNION ALL очень похожи. Чтобы понять, как работает решение с использованием подзапроса, просто отдельно выполните запрос, предшествующий UNION ALL. Запрос ниже – это строки 1–27 кода раздела «Решение»:

```
select *
   from (
 select e.empno,e.ename,e.job,e.mgr,e.hiredate,
        e.sal,e.comm,e.deptno, count(*) as cnt
   from emp e
  group by empno,ename,job,mgr,hiredate,
           sal,comm,deptno
        ) e
  where not exists (
select null
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
       v.sal,v.comm,v.deptno, count(*) as cnt
  from v
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
       ) v
  where v.empno    = e.empno
    and v.ename    = e.ename
    and v.job      = e.job
    and v.mgr      = e.mgr
    and v.hiredate = e.hiredate
    and v.sal      = e.sal
    and v.deptno   = e.deptno
    and v.cnt      = e.cnt
    and coalesce(v.comm,0) = coalesce(e.comm,0)
)
```

```
EMPNO ENAME      JOB         MGR HIREDATE      SAL  COMM DEPTNO CNT
----- ---------- --------- ----- ----------- ----- ----- ------ ---
 7521 WARD       SALESMAN   7698 22-FEB-1981  1250   500     30   1
 7782 CLARK      MANAGER    7839 09-JUN-1981  2450           10   1
 7839 KING       PRESIDENT       17-NOV-1981  5000           10   1
 7934 MILLER     CLERK      7782 23-JAN-1982  1300           10   1
```

Обратите внимание, что сравниваются не таблица EMP и представление V, а вложенное представление E и вложенное представление V. Для каждой строки определяется кардинальное число, которое возвращается как атрибут этой строки. Происходит сравнение всех строк и этих их атрибутов, показывающих количество таких строк в таблице. Если возникают сложности с пониманием, как происходит сравнение, выполните подзапросы отдельно. На следующем этапе нужно найти во вложенном представлении E все строки (включая CNT), которых нет во вложенном представлении V. Для сравнения используются связанный подзапрос и операция NOT EXISTS. Объединения выявят все одинаковые строки, а в результат войдут те строки вложенного представления E, которых нет среди строк, возвращенных объединением. Запрос, следующий после UNION ALL, делает противоположную операцию: он находит во вложенном представлении V строки, которых нет во вложенном представлении E:

```
select *
  from (
select v.empno,v.ename,v.job,v.mgr,v.hiredate,
       v.sal,v.comm,v.deptno, count(*) as cnt
  from v
 group by empno,ename,job,mgr,hiredate,
          sal,comm,deptno
       ) v
  where not exists (
select null
  from (
 select e.empno,e.ename,e.job,e.mgr,e.hiredate,
        e.sal,e.comm,e.deptno, count(*) as cnt
   from emp e
  group by empno,ename,job,mgr,hiredate,
           sal,comm,deptno
       ) e
  where v.empno    = e.empno
    and v.ename    = e.ename
    and v.job      = e.job
    and v.mgr      = e.mgr
    and v.hiredate = e.hiredate
    and v.sal      = e.sal
    and v.deptno   = e.deptno
    and v.cnt      = e.cnt
    and coalesce(v.comm,0) = coalesce(e.comm,0)
)
```

```
EMPNO ENAME      JOB         MGR HIREDATE      SAL  COMM DEPTNO CNT
----- ---------- --------- ----- ----------- ----- ----- ------ ---
 7521 WARD       SALESMAN   7698 22-FEB-1981  1250   500     30   2
```

Затем UNION ALL комбинирует результаты и формирует окончательное результирующее множество.

Алес Спектик (Ales Spectic) и Джонатан Генник (Jonothan Gennick) в своей книге «Transact-SQL Cookbook» (O'Reilly) приводят альтернативное решение. Смотрите раздел главы 2 «Comparing Two Sets for Equality».

Идентификация и устранение некорректного использования декартова произведения

Задача

Требуется возвратить имя каждого служащего 10-го отдела и местонахождение отдела. Следующий запрос не обеспечивает правильного формирования необходимого результирующего множества:

```
select e.ename, d.loc
  from emp e, dept d
```

```
 where e.deptno = 10

ENAME      LOC
---------- -------------
CLARK      NEW YORK
CLARK      DALLAS
CLARK      CHICAGO
CLARK      BOSTON
KING       NEW YORK
KING       DALLAS
KING       CHICAGO
KING       BOSTON
MILLER     NEW YORK
MILLER     DALLAS
MILLER     CHICAGO
MILLER     BOSTON
```

Результирующее множество должно быть таким:

```
ENAME      LOC
---------- -------------
CLARK      NEW YORK
KING       NEW YORK
MILLER     NEW YORK
```

Решение

Для получения необходимого результирующего множества используйте объединение таблиц в конструкции FROM:

```
1 select e.ename, d.loc
2   from emp e, dept d
3  where e.deptno = 10
4    and d.deptno = e.deptno
```

Обсуждение

Если мы посмотрим на данные таблицы DEPT:

```
select * from dept

    DEPTNO DNAME          LOC
---------- -------------- -------------
        10 ACCOUNTING     NEW YORK
        20 RESEARCH       DALLAS
        30 SALES          CHICAGO
        40 OPERATIONS     BOSTON
```

то увидим, что 10-й отдел находится в Нью-Йорке. Таким образом, понятно, что включение в результирующее множество служащих не из Нью-Йорка является ошибочным. Количество строк, возвращаемое неверным запросом, является следствием кардинальности двух таблиц конструкции FROM. В исходном запросе фильтр по выбору 10-го отдела, примененный к таблице EMP, обеспечит возвращение трех строк.

Поскольку для DEPT фильтра нет, возвращаются все четыре строки DEPT. Умножая три на четыре, получаем двенадцать; таким образом, неправильный запрос возвращает двенадцать строк. Обычно, чтобы не происходило прямого (декартова) произведения, применяется правило n–1, где n представляет количество таблиц в FROM, а n–1 – минимальное число объединений, необходимое во избежание прямого произведения. В зависимости от того, что представляют собой колонки и ключи в вашей таблице, вероятно, вам понадобится более, чем n–1 объединений, но при написании запроса можно начать с n–1 объединения.

При правильном использовании декартовы произведения могут быть очень полезны. Они используются в рецепте «Проход строки» главы 6 и во многих других запросах. Обычно декартовы произведения используются при транспонировании или повороте результирующего множества, формировании последовательности значений и имитации цикла.

Осуществление объединений при использовании агрегатных функций

Задача

Необходимо осуществить агрегацию, но запрос обращен к нескольким таблицам. Требуется обеспечить, что объединения не нарушат агрегацию. Например, стоит задача найти сумму заработных плат служащих 10-го отдела, а также сумму их премий. Некоторые служащие получили не одну премию, и объединение таблиц EMP и EMP_BONUS приводит к тому, что агрегатная функция SUM возвращает неверные значения. Для обсуждаемой задачи таблица EMP_BONUS содержит следующие данные:

```
select * from emp_bonus

EMPNO RECEIVED          TYPE
----- ----------- ----------
 7934 17-MAR-2005          1
 7934 15-FEB-2005          2
 7839 15-FEB-2005          3
 7782 15-FEB-2005          1
```

Теперь рассмотрим запрос, возвращающий зарплаты и премии всех служащих 10-го отдела. Столбец BONUS.TYPE определяет размер премии. Премия 1-го типа составляет 10% заработной платы служащего, премия 2-го типа – 20%, и 3-го типа – 30%.

```
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
```

```
                when eb.type = 2 then .2
                else .3
           end as bonus
  from emp e, emp_bonus eb
 where e.empno  = eb.empno
   and e.deptno = 10
```

```
    EMPNO ENAME             SAL     DEPTNO      BONUS
--------- ---------- ---------- ---------- ----------
     7934 MILLER           1300         10        130
     7934 MILLER           1300         10        260
     7839 KING             5000         10       1500
     7782 CLARK            2450         10        245
```

До сих пор все идет хорошо. Однако проблемы начинаются при попытке объединения с таблицей EMP_BONUS для суммирования премий:

```
select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3
             end as bonus
  from emp e, emp_bonus eb
 where e.empno  = eb.empno
   and e.deptno = 10
       ) x
 group by deptno
```

```
DEPTNO  TOTAL_SAL TOTAL_BONUS
------ ---------- -----------
    10      10050        2135
```

В столбце TOTAL_BONUS (сумма премий) получаем верное значение, тогда как значение столбца TOTAL_SAL (сумма заработных плат) ошибочное. Сумма всех заработных плат 10-го отдела составляет 8750, как показывает следующий запрос:

```
select sum(sal) from emp where deptno=10
```

```
  SUM(SAL)
----------
      8750
```

Почему в столбце TOTAL_SAL получено неверное значение? Причина состоит в дублировании строк, возникшем в результате объединения. Рассмотрим следующий запрос, который осуществляет объединение таблиц EMP и EMP_ BONUS:

```
select e.ename,
       e.sal
  from emp e, emp_bonus eb
 where e.empno  = eb.empno
   and e.deptno = 10

ENAME             SAL
---------- ----------
CLARK            2450
KING             5000
MILLER           1300
MILLER           1300
```

Теперь видно, почему значение TOTAL_SAL неверно: заработная плата служащего MILLER учитывается дважды. Окончательное результирующее множество, которое требовалось получить, на самом деле вот такое:

```
DEPTNO  TOTAL_SAL TOTAL_BONUS
------ ---------- -----------
    10       8750        2135
```

Решение

Необходимо быть очень аккуратным при совместном использовании агрегатных функций и объединений. Обычно избежать ошибок, обусловленных дублированием строк, возникающим при объединении, можно двумя способами: или просто использовать в вызове агрегатной функции ключевое слово DISTINCT, чтобы в вычислении участвовали только уникальные экземпляры значений, или сначала провести агрегацию (во вложенном запросе), а потом объединение. Во втором случае мы избежим ошибки при вычислении, поскольку агрегация будет выполнена еще до объединения, что полностью устраняет проблему. В приведенных ниже решениях используется ключевое слово DISTINCT. В разделе «Обсуждение» рассматривается, как с помощью вложенного запроса выполнить агрегацию до объединения.

MySQL и PostgreSQL

Суммируем только уникальные (DISTINCT) заработные платы:

```
 1  select deptno,
 2         sum(distinct sal) as total_sal,
 3         sum(bonus) as total_bonus
 4    from (
 5  select e.empno,
 6         e.ename,
 7         e.sal,
 8         e.deptno,
 9         e.sal*case when eb.type = 1 then .1
10                    when eb.type = 2 then .2
11                    else .3
```

```
12              end  as bonus
13    from emp e, emp_bonus eb
14   where e.empno  = eb.empno
15     and e.deptno = 10
16         ) x
17   group by deptno
```

DB2, Oracle и SQL Server

Эти платформы поддерживают предыдущее решение, но для них также можно использовать альтернативное решение с применением оконной функции SUM OVER:

```
 1  select distinct deptno,total_sal,total_bonus
 2    from (
 3  select e.empno,
 4         e.ename,
 5         sum(distinct e.sal) over
 6         (partition by e.deptno) as total_sal,
 7         e.deptno,
 8         sum(e.sal*case when eb.type = 1 then .1
 9                        when eb.type = 2 then .2
10                        else .3 end) over
11         (partition by deptno) as total_bonus
12    from emp e, emp_bonus eb
13   where e.empno  = eb.empno
14     and e.deptno = 10
15         ) x
```

Обсуждение

MySQL и PostgreSQL

Второй запрос в разделе «Задача» данного рецепта осуществляет объединение таблиц EMP и EMP_BONUS, в результате чего возвращаются две строки для служащего MILLER, что и обусловливает ошибочность значения суммы в EMP.SAL (заработная плата этого служащего суммируется дважды). Решение состоит в суммировании только уникальных значений EMP.SAL, возвращаемых запросом. Следующий запрос является альтернативным решением. Здесь сначала вычисляется сумма заработных плат всех служащих 10-го отдела, затем эта строка объединяется с таблицей EMP, которая в дальнейшем объединяется с таблицей EMP_BONUS. Запрос ниже применим для всех СУБД:

```
select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                      when eb.type = 2 then .2
                      else .3 end) as total_bonus
  from emp e,
       emp_bonus eb,
       (
```

```
  select deptno, sum(sal) as total_sal
    from emp
   where deptno = 10
   group by deptno
         ) d
   where e.deptno = d.deptno
     and e.empno  = eb.empno
   group by d.deptno,d.total_sal

   DEPTNO  TOTAL_SAL  TOTAL_BONUS
--------- ---------- ------------
       10       8750         2135
```

DB2, Oracle и SQL Server

В этом альтернативном решении используется оконная функция SUM OVER. Данный запрос занимает строки 3–14 примера раздела «Решение» и возвращает следующее результирующее множество:

```
select e.empno,
          e.ename,
          sum(distinct e.sal) over
          (partition by e.deptno) as total_sal,
          e.deptno,
          sum(e.sal*case when eb.type = 1 then .1
                         when eb.type = 2 then .2
                        else .3 end) over
         (partition by deptno) as total_bonus
    from emp e, emp_bonus eb
   where e.empno  = eb.empno
     and e.deptno = 10

EMPNO ENAME       TOTAL_SAL DEPTNO TOTAL_BONUS
----- ---------- ---------- ------ -----------
 7934 MILLER           8750     10        2135
 7934 MILLER           8750     10        2135
 7782 CLARK            8750     10        2135
 7839 KING             8750     10        2135
```

Функция для работы с окнами SUM OVER вызывается дважды. В первый раз – для вычисления суммы уникальных заработных плат для заданного сегмента или группы. В данном случае сегмент – это 10-й отдел (DEPTNO 10), и сумма уникальных заработных плат для DEPTNO 10 составляет 8750. Следующий вызов SUM OVER осуществляет вычисление суммы премий того же сегмента. Окончательное результирующее множество (значения столбцов TOTAL_SAL и TOTAL_BONUS) формируется путем суммирования уникальных значений заработных плат и премий.

Внешнее объединение при использовании агрегатных функций

Задача

Задача аналогична задаче раздела «Осуществление объединений при использовании агрегатных функций», но таблица EMP_BONUS изменена. В данном случае премии получали не все служащие 10-го отдела. Рассмотрим таблицу EMP_BONUS и запрос, обеспечивающий (якобы) вычисление суммы заработных плат и суммы премий всех служащих 10-го отдела:

```
select * from emp_bonus

     EMPNO RECEIVED          TYPE
---------- ----------- ----------
      7934 17-MAR-2005          1
      7934 15-FEB-2005          2

select deptno,
       sum(sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end as bonus
  from emp e, emp_bonus eb
 where e.empno  = eb.empno
   and e.deptno = 10
       )
 group by deptno

DEPTNO  TOTAL_SAL TOTAL_BONUS
------ ---------- -----------
    10       2600         390
```

Результат в TOTAL_BONUS верен, а значение, возвращенное для TOTAL_SAL, не является суммой заработных плат всех служащих 10-го отдела. Следующий запрос показывает, почему значение TOTAL_SAL ошибочно:

```
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type = 1 then .1
                  when eb.type = 2 then .2
             else .3 end as bonus
  from emp e, emp_bonus eb
```

```
 where e.empno  = eb.empno
   and e.deptno = 10

   EMPNO ENAME              SAL     DEPTNO      BONUS
-------- ---------- ---------- ---------- ----------
    7934 MILLER           1300         10        130
    7934 MILLER           1300         10        260
```

Вместо вычисления суммы всех заработных плат 10-го отдела суммируется только заработная плата служащего MILLER, и ошибочно она суммируется дважды. В конце концов необходимо было получить такое результирующее множество:

```
DEPTNO  TOTAL_SAL TOTAL_BONUS
------ ---------- -----------
    10       8750         390
```

Решение

Решение аналогично решению, представленному в разделе «Осуществление объединений при использовании агрегатных функций», но в данном случае осуществляется внешнее объединение с EMP_BONUS, что гарантирует включение всех служащих 10-го отдела.

DB2, MySQL, PostgreSQL, SQL Server

Осуществляем внешнее объединение с EMP_BONUS, затем суммируем только уникальные заработные платы 10-го отдела:

```
 1  select deptno,
 2         sum(distinct sal) as total_sal,
 3         sum(bonus) as total_bonus
 4    from (
 5  select e.empno,
 6         e.ename,
 7         e.sal,
 8         e.deptno,
 9         e.sal*case when eb.type is null then 0
10                    when eb.type = 1 then .1
11                    when eb.type = 2 then .2
12                    else .3 end as bonus
13    from emp e left outer join emp_bonus eb
14      on (e.empno = eb.empno)
15   where e.deptno = 10
16         )
17   group by deptno
```

Можно также использовать оконную функцию SUM OVER:

```
 1  select distinct deptno,total_sal,total_bonus
 2    from (
 3  select e.empno,
 4         e.ename,
 5         sum(distinct e.sal) over
```

```
        (partition by e.deptno) as total_sal,
        e.deptno,
        sum(e.sal*case when eb.type is null then 0
                       when eb.type = 1 then .1
                       when eb.type = 2 then .2
                       else .3
                  end) over
        (partition by deptno) as total_bonus
   from emp e left outer join emp_bonus eb
     on (e.empno = eb.empno)
  where e.deptno = 10
        ) x
```

Oracle

При работе с Oracle 9*i* Database или более поздними версиями можно использовать предыдущее решение. В качестве альтернативы применяется собственный синтаксис Oracle для внешнего объединения; этот вариант является единственно возможным для пользователей Oracle 8*i* Database и более ранних версий:

```
select deptno,
       sum(distinct sal) as total_sal,
       sum(bonus) as total_bonus
  from (
select e.empno,
       e.ename,
       e.sal,
       e.deptno,
       e.sal*case when eb.type is null then 0
                  when eb.type = 1 then .1
                  when eb.type = 2 then .2
                  else .3 end as bonus
  from emp e, emp_bonus eb
 where e.empno  = eb.empno (+)
   and e.deptno = 10
       )
 group by deptno
```

Пользователи Oracle 8*i* Database могут также применить решение с функцией SUM OVER, приведенное для DB2 и других баз данных, но в него необходимо внести изменения с использованием собственного синтаксиса Oracle для внешнего объединения, показанного в предыдущем запросе.

Обсуждение

Второй запрос в разделе «Задача» данного рецепта объединяет таблицы EMP и EMP_BONUS и возвращает только строки для служащего MILLER, что и является причиной ошибки при вычислении суммы EMP.SAL (остальные служащие 10-го отдела не получали премий, и их заработные платы не вошли в сумму). Решение – провести внеш-

нее объединение таблицы EMP с таблицей EMP_BONUS, чтобы даже служащие, не получавшие премии, были учтены при вычислении результата. Если служащий не получал премию, для него в столбце EMP_BONUS.TYPE будет возвращено значение NULL. Это важно помнить, поскольку выражение CASE в данном случае немного отличается от представленного в решении раздела «Осуществление объединений при использовании агрегатных функций». Если в столбце EMP_BONUS.TYPE содержится значение NULL, выражение CASE возвращает нуль, который не оказывает никакого влияния на сумму.

Следующий запрос является альтернативным решением. Сначала вычисляется сумма заработных плат всех служащих 10-го отдела, а затем это значение путем объединения добавляется в таблицу EMP, которая впоследствии объединяется с таблицей EMP_BONUS (таким образом мы уходим от внешнего объединения). Следующий запрос применим во всех СУБД:

```
select d.deptno,
       d.total_sal,
       sum(e.sal*case when eb.type = 1 then .1
                      when eb.type = 2 then .2
                      else .3 end)  as total_bonus
  from emp e,
       emp_bonus eb,
       (
select deptno, sum(sal) as total_sal
  from emp
 where deptno = 10
 group by deptno
       ) d
 where e.deptno = d.deptno
   and e.empno  = eb.empno
 group by d.deptno,d.total_sal
```

```
   DEPTNO  TOTAL_SAL TOTAL_BONUS
--------- ---------- -----------
       10       8750         390
```

Возвращение отсутствующих данных из нескольких таблиц

Задача

Одновременно из нескольких таблиц требуется выбрать данные, отсутствующие в той или иной таблице. Чтобы возвратить строки таблицы DEPT, которых нет в таблице EMP (любой отдел, в котором нет служащих), необходимо провести внешнее объединение. Рассмотрим следующий запрос, в результате которого будут возвращены все значения столбцов DEPTNO и DNAME таблицы DEPT и имена всех служащих всех отделов (если в отделе есть служащие):

```
select d.deptno,d.dname,e.ename
  from dept d left outer join emp e
    on (d.deptno=e.deptno)
```

```
    DEPTNO DNAME          ENAME
---------- -------------- ----------
        20 RESEARCH       SMITH
        30 SALES          ALLEN
        30 SALES          WARD
        20 RESEARCH       JONES
        30 SALES          MARTIN
        30 SALES          BLAKE
        10 ACCOUNTING     CLARK
        20 RESEARCH       SCOTT
        10 ACCOUNTING     KING
        30 SALES          TURNER
        20 RESEARCH       ADAMS
        30 SALES          JAMES
        20 RESEARCH       FORD
        10 ACCOUNTING     MILLER
        40 OPERATIONS
```

Последняя строка, отдел OPERATIONS (операции), возвращена, несмотря на то что в отделе нет ни одного служащего. Это является следствием левостороннего внешнего объединения таблицы DEPT с таблицей EMP. Теперь, предположим, есть служащий, не относящийся ни к одному отделу. Как получить приведенное выше результирующее множество со строкой служащего, не приписанного ни к одному отделу? Иначе говоря, в том же запросе необходимо провести внешнее объединение таблицы EMP с таблицей DEPT и таблицы DEPT с таблицей EMP. После создания нового служащего первая попытка может быть такой:

```
insert into emp (empno,ename,job,mgr,hiredate,sal,comm,deptno)
select 1111,'YODA','JEDI',null,hiredate,sal,comm,null
  from emp
 where ename = 'KING'
```

```
select d.deptno,d.dname,e.ename
  from dept d right outer join emp e
    on (d.deptno=e.deptno)
```

```
    DEPTNO DNAME          ENAME
---------- -------------- ----------
        10 ACCOUNTING     MILLER
        10 ACCOUNTING     KING
        10 ACCOUNTING     CLARK
        20 RESEARCH       FORD
        20 RESEARCH       ADAMS
        20 RESEARCH       SCOTT
        20 RESEARCH       JONES
        20 RESEARCH       SMITH
        30 SALES          JAMES
        30 SALES          TURNER
```

```
        30 SALES          BLAKE
        30 SALES          MARTIN
        30 SALES          WARD
        30 SALES          ALLEN
                          YODA
```

При таком внешнем объединении удается возвратить строку нового служащего, но теряется отдел OPERATIONS, присутствующий в исходном результирующем множестве. Окончательное результирующее множество должно содержать и строку служащего YODA, и строку отдела OPERATIONS, как показано ниже:

```
    DEPTNO DNAME          ENAME
---------- -------------- ----------
        10 ACCOUNTING     CLARK
        10 ACCOUNTING     KING
        10 ACCOUNTING     MILLER
        20 RESEARCH       ADAMS
        20 RESEARCH       FORD
        20 RESEARCH       JONES
        20 RESEARCH       SCOTT
        20 RESEARCH       SMITH
        30 SALES          ALLEN
        30 SALES          BLAKE
        30 SALES          JAMES
        30 SALES          MARTIN
        30 SALES          TURNER
        30 SALES          WARD
        40 OPERATIONS
                          YODA
```

Решение

Чтобы возвратить отсутствующие данные из обеих таблиц, используйте полное внешнее объединение на основании общего значения.

DB2, MySQL, PostgreSQL, SQL Server

Чтобы вместе с совпадающими строками возвратить из обеих таблиц отсутствующие в другой таблице строки, явно используйте команду FULL OUTER JOIN (полное внешнее объединение):

```
1 select d.deptno,d.dname,e.ename
2   from dept d full outer join emp e
3     on (d.deptno=e.deptno)
```

Или объедините результаты двух разных внешних объединений:

```
1 select d.deptno,d.dname,e.ename
2   from dept d right outer join emp e
3     on (d.deptno=e.deptno)
4  union
5 select d.deptno,d.dname,e.ename
```

```
6   from dept d left outer join emp e
7     on (d.deptno=e.deptno)
```

Oracle

При работе с Oracle 9*i* Database или более поздними версиями можно использовать любое из предыдущих решений. Как альтернатива применяется собственный синтаксис Oracle для внешнего объединения. Этот вариант является единственно возможным для пользователей Oracle 8*i* Database и более ранних версий:

```
1 select d.deptno,d.dname,e.ename
2   from dept d, emp e
3  where d.deptno = e.deptno(+)
4  union
5 select d.deptno,d.dname,e.ename
6   from dept d, emp e
7  where d.deptno(+) = e.deptno
```

Обсуждение

Полное внешнее объединение – это просто сочетание внешних объединений обеих таблиц. Чтобы понять принцип полного внешнего объединения, просто выполните каждое внешнее объединение по отдельности и затем произведите слияние результатов. Следующий запрос возвращает строки таблицы DEPT и все совпадающие строки таблицы EMP (если таковые имеются):

```
select d.deptno,d.dname,e.ename
  from dept d left outer join emp e
    on (d.deptno = e.deptno)

DEPTNO DNAME          ENAME
------ -------------- ----------
    20 RESEARCH       SMITH
    30 SALES          ALLEN
    30 SALES          WARD
    20 RESEARCH       JONES
    30 SALES          MARTIN
    30 SALES          BLAKE
    10 ACCOUNTING     CLARK
    20 RESEARCH       SCOTT
    10 ACCOUNTING     KING
    30 SALES          TURNER
    20 RESEARCH       ADAMS
    30 SALES          JAMES
    20 RESEARCH       FORD
    10 ACCOUNTING     MILLER
    40 OPERATIONS
```

Следующий запрос возвращает строки таблицы EMP и все совпадающие строки таблицы DEPT (если таковые имеются):

```
select d.deptno,d.dname,e.ename
  from dept d right outer join emp e
    on (d.deptno = e.deptno)

DEPTNO DNAME          ENAME
------ -------------- ----------
    10 ACCOUNTING     MILLER
    10 ACCOUNTING     KING
    10 ACCOUNTING     CLARK
    20 RESEARCH       FORD
    20 RESEARCH       ADAMS
    20 RESEARCH       SCOTT
    20 RESEARCH       JONES
    20 RESEARCH       SMITH
    30 SALES          JAMES
    30 SALES          TURNER
    30 SALES          BLAKE
    30 SALES          MARTIN
    30 SALES          WARD
    30 SALES          ALLEN
                      YODA
```

Для получения окончательного результирующего множества соединяем результаты этих двух запросов.

Значения NULL в операциях и сравнениях

Задача

Значение NULL никогда не бывает равным или не равным ни одному значению, даже самому себе, однако стоит задача проводить вычисления над данными столбца, который может содержать NULL значения. Например, требуется найти в таблице EMP всех служащих, сумма комиссионных (COMM) которых меньше, чем у служащего WARD. В результирующее множество должны быть включены и служащие, для которых значение комиссионных не определено (NULL).

Решение

С помощью такой функции, как COALESCE, преобразуйте значение NULL в действительное значение, которое может использоваться в обычных вычислениях:

```
1 select ename,comm
2   from emp
3  where coalesce(comm,0) < ( select comm
4                               from emp
5                              where ename = 'WARD' )
```

Обсуждение

Функция COALESCE возвращает из списка переданных в нее значений первое не являющееся NULL значение. Когда встречается значение NULL, оно замещается нулем, который потом сравнивается со значением комиссионных служащего WARD. Это можно увидеть, поместив функцию COALESCE в список SELECT:

```
select ename,comm,coalesce(comm,0)
  from emp
 where coalesce(comm,0) < ( select comm
                              from emp
                             where ename = 'WARD' )
```

```
ENAME            COMM COALESCE(COMM,0)
---------- ---------- ----------------
SMITH                              0
ALLEN             300              300
JONES                              0
BLAKE                              0
CLARK                              0
SCOTT                              0
KING                               0
TURNER              0              0
ADAMS                              0
JAMES                              0
FORD                               0
MILLER                             0
```

4

Вставка, обновление, удаление

Предыдущие главы посвящены базовым техникам запросов, и основное внимание в них сосредоточено на извлечении информации из базы данных. В данной главе центральными являются следующие три темы:

- Вставка новых записей в базу данных.
- Обновление существующих записей.
- Удаление ненужных записей.

Для облегчения поиска рецепты данной главы сгруппированы по темам: сначала идут все рецепты, касающиеся вставки, за ними следуют рецепты по обновлению записей и, наконец, рецепты по удалению данных.

Вставка записей обычно не представляет никакой сложности. Все начинается с простой задачи по вставке одной строки. Однако часто для создания новых строк эффективнее использовать подход, основанный на множествах. Здесь также представлены методики одновременной вставки нескольких строк.

Аналогично рецепты по обновлению и удалению начинаются с простых задач. Можно обновить одну запись и удалить одну запись. Но существуют также очень эффективные техники, позволяющие обновить несколько записей одновременно. Кроме того, есть много удобных методов удаления записей. Например, можно удалять записи одной таблицы в зависимости от того, присутствуют ли они в другой таблице.

В SQL даже есть возможность одновременной вставки, обновления и удаления данных. Правда, это довольно свежее нововведение в стандарт, и пока что, возможно, вы не видите в нем пользы. Однако новое выражение MERGE представляет очень мощное средство синхронизации таблицы базы данных с внешним источником данных (как, например, в случае ввода плоских файлов с удаленной системы). Подробнее смотрите в разделе данной главы, посвященном этому вопросу.

Вставка новой записи

Задача

Требуется вставить новую запись в таблицу. Например, стоит задача вставить новую запись в таблицу DEPT. В столбце DEPTNO должно быть сохранено значение 50, в столбце DNAME – «PROGRAMMING», и в столбце LOC – «BALTIMORE».

Решение

Чтобы вставить одну строку, используйте выражение INSERT (вставить) с блоком VALUES (значения):

```
insert into dept (deptno,dname,loc)
values (50,'PROGRAMMING','BALTIMORE')
```

В DB2 и MySQL можно вставлять как одну, так и несколько строк одновременно, включив в выражение несколько списков VALUES:

```
/* вставляем несколько строк */
insert into dept (deptno,dname,loc)
values (1,'A','B'),
       (2,'B','C')
```

Обсуждение

Выражение INSERT позволяет создавать новые строки в таблицах баз данных. Синтаксис вставки одной строки одинаков для баз данных всех производителей.

Запись выражения INSERT можно сократить, опустив список столбцов:

```
insert into dept
values (50,'PROGRAMMING','BALTIMORE')
```

Однако если имена столбцов не перечислены, вставка будет осуществляться во *все* столбцы таблицы, и порядок значений в списке VALUES должен соответствовать порядку отображения столбцов в ответе на запрос SELECT *.

Вставка значений по умолчанию

Задача

В таблице для некоторых столбцов могут быть определены значения по умолчанию. Требуется вставить строку значений по умолчанию, не задавая этих значений в запросе. Рассмотрим следующую таблицу:

```
create table D (id integer default 0)
```

Необходимо вставить нуль, не задавая его явно в списке значений выражения INSERT, т. е. вставить значение по умолчанию, каким бы оно ни было.

Решение

Все СУБД поддерживают ключевое слово DEFAULT (по умолчанию) как средство явного задания значения по умолчанию для столбца. Некоторые производители предоставляют дополнительные способы решения этой задачи.

Следующий пример иллюстрирует применение ключевого слова DEFAULT:

```
insert into D values (default)
```

Можно также явно задавать имя столбца. Если вставка осуществляется не во все столбцы таблицы, явное определение имени столбца обязательно:

```
insert into D (id) values (default)
```

Oracle 8*i* Database и более ранние версии не поддерживают ключевое слово DEFAULT. До Oracle 9*i* невозможно было явно вставить в столбец значение по умолчанию.

MySQL позволяет задавать пустой список значений, если для всех столбцов определены значения по умолчанию:

```
insert into D values ()
```

В этом случае во все столбцы будут вставлены их значения по умолчанию.

PostgreSQL и SQL Server поддерживают оператор DEFAULT VALUES (значения по умолчанию):

```
insert into D default values
```

Блок DEFAULT VALUES обусловливает сохранение во всех столбцах их значений по умолчанию.

Обсуждение

Ключевое слово DEFAULT в списке значений обеспечит вставку значения, которое было определено как применяемое по умолчанию во время создания конкретного столбца. Это ключевое слово используется во всех СУБД.

Пользователям MySQL, PostgreSQL и SQL Server доступен другой вариант заполнения всех столбцов таблицы значениями по умолчанию, если таковые определены (как для таблицы D в данном случае). Создание пустой строки со значениями по умолчанию во всех столбцах обеспечит использование пустого списка VALUES (MySQL) или блока DEFAULT VALUES (PostgreSQL и SQL Server). В противном случае необходимо задавать DEFAULT для каждого столбца таблицы.

Чтобы вставить значение по умолчанию в столбец таблицы, в которой значения по умолчанию заданы не для всех столбцов, необходимо

просто исключить этот столбец из списка вставки; ключевое слово DEFAULT не требуется. Скажем, в таблице D есть дополнительный столбец, для которого не было определено значение по умолчанию:

```
create table D (id integer default 0, foo varchar(10))
```

Можно вставить значение по умолчанию в столбец ID, указывая в списке вставки только столбец FOO:

```
 insert into D (foo) values ('Bar')
```

В результате этого выражения в таблицу будет вставлена строка со значением 0 в столбце ID и значением «Bar» в столбце FOO. Столбец ID принимает значение по умолчанию, потому что другого значения не задано.

Переопределение значения по умолчанию значением NULL

Задача

Осуществляется вставка в таблицу, столбец которой имеет значение по умолчанию, и требуется переопределить это значение значением NULL. Рассмотрим следующую таблицу:

```
create table D (id integer default 0, foo VARCHAR(10))
```

Требуется вставить строку со значением NULL в столбце ID.

Решение

Можно явно задать NULL в списке значений:

```
insert into d (id, foo) values (null, 'Brighten')
```

Обсуждение

Не все осознают, что значение NULL можно явно задавать в списке значений выражения INSERT. Как правило, если столбцу не задается значение, его просто не включают в список столбцов и значений:

```
insert into d (foo) values ('Brighten')
```

Здесь для ID значение не задано. Многие ожидают, что столбцу будет присвоено неопределенное значение. Но, вот незадача, при создании таблицы для этого столбца было определено значение по умолчанию, поэтому в результате выполнения указанного выше выражения INSERT ID получит значение 0 (значение по умолчанию). Определение NULL в качестве значения столбца позволяет присваивать ему значение NULL, несмотря на любое заданное значение по умолчанию.

Копирование строк из одной таблицы в другую

Задача

Требуется, используя запрос, скопировать строки из одной таблицы в другую. Запрос может быть сложным или простым, но требуется, чтобы в конечном счете его результат был вставлен в другую таблицу. Например, необходимо скопировать строки из таблицы DEPT в таблицу DEPT_EAST, которая уже создана, имеет такую же структуру (такие же столбцы тех же типов), что и DEPT, и на данный момент является пустой.

Решение

Используйте выражение INSERT, за которым следует запрос, возвращающий необходимые строки:

```
1 insert into dept_east (deptno,dname,loc)
2 select deptno,dname,loc
3   from dept
4  where loc in ( 'NEW YORK','BOSTON' )
```

Обсуждение

Просто поставьте после выражения INSERT запрос, возвращающий требуемые строки. Если необходимо скопировать все строки исходной таблицы, уберите из запроса блок WHERE. Как и при обычной вставке, необязательно явно задавать, в какие столбцы должны быть вставлены значения. Но если имена столбцов не заданы, вставка будет производиться во *все* столбцы таблицы, и необходимо строго придерживаться порядка значений в списке SELECT, как описывалось ранее в разделе «Вставка новой записи».

Копирование описания таблицы

Задача

Требуется создать новую таблицу с таким же набором столбцов, как и в существующей. Например, необходимо создать копию таблицы DEPT под именем DEPT_2, копируя при этом только структуру столбцов, а не строки таблицы.

Решение

DB2

Используйте оператор LIKE (как) с командой CREATE TABLE (создать таблицу):

```
create table dept_2 like dept
```

Oracle, MySQL и PostgreSQL

Используйте команду CREATE TABLE с подзапросом, который не возвращает ни одной строки:

```
1 create table dept_2
2 as
3 select *
4   from dept
5  where 1 = 0
```

SQL Server

Используйте оператор INTO в обычном SELECT запросе, который не возвращает ни одной строки:

```
1 select *
2   into dept_2
3   from dept
4  where 1 = 0
```

Обсуждение

DB2

Команда DB2 CREATE TABLE … LIKE позволяет использовать одну таблицу как шаблон для создания другой таблицы. Для этого после ключевого слова LIKE просто указывается имя таблицы-шаблона.

Oracle, MySQL и PostgreSQL

При использовании конструкции Create Table As Select (CTAS) все строки, полученные в результате запроса, будут вставлены в новую таблицу. В предлагаемом решении выражение «1 = 0» в предикате WHERE обеспечивает тот факт, что запрос не возвратит ни одной строки. Таким образом, результатом выполнения данного выражения CTAS является пустая таблица, созданная на основании столбцов, указанных в операторе SELECT запроса.

SQL Server

При копировании таблицы с помощью оператора INTO все строки, полученные в результате запроса, будут использоваться для заполнения новой таблицы. Выражение «1 = 0» в предикате обусловливает, что ни одна строка не будет возвращена. В результате получаем пустую таблицу, созданную на основании столбцов, указанных в операторе SELECT запроса.

Вставка в несколько таблиц одновременно

Задача

Требуется вставить строки, возвращенные запросом, в несколько таблиц. Например, необходимо вставить строки из таблицы DEPT в таб-

лицы DEPT_EAST, DEPT_WEST и DEPT_MID. Структура всех трех таблиц (количество и типы столбцов) такая же, как и у DEPT, и в настоящий момент они пусты.

Решение

Решение – вставить результат запроса в необходимые таблицы. Отличие от «Копирования строк из одной таблицы в другую» состоит в том, что в данном случае имеется несколько таблиц на выходе.

Oracle

Используйте выражения INSERT ALL (вставить во все) или INSERT FIRST (вставить в первую). Синтаксис обоих выражений одинаковый, кроме ключевых слов ALL и FIRST. В следующем примере используется выражение INSERT ALL, что обеспечивает вставку во все возможные таблицы:

```
1  insert all
2    when loc in ('NEW YORK','BOSTON') then
3      into dept_east (deptno,dname,loc) values (deptno,dname,loc)
4    when loc = 'CHICAGO' then
5      into dept_mid  (deptno,dname,loc) values (deptno,dname,loc)
6    else
7      into dept_west (deptno,dname,loc) values (deptno,dname,loc)
8  select deptno,dname,loc
9    from dept
```

DB2

Осуществляйте вставку во вложенное представление, в котором происходит объединение (UNION ALL) всех вставляемых таблиц. Причем необходимо наложить ограничения на таблицы, гарантирующие размещение строк в соответствующих таблицах:

```
create table dept_east
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc in ('NEW YORK','BOSTON')))

create table dept_mid
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc = 'CHICAGO'))

create table dept_west
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc = 'DALLAS'))

1  insert into (
2    select * from dept_west union all
3    select * from dept_east union all
4    select * from dept_mid
5  ) select * from dept
```

MySQL, PostgreSQL и SQL Server

На момент написания данной книги эти производители не поддерживали вставки в несколько таблиц.

Обсуждение

Oracle

В Oracle для вставки данных в несколько таблиц используются блоки WHEN-THEN-ELSE, которые обеспечивают обработку строк, возвращаемых вложенным SELECT, и соответствующую их вставку. В примере данного рецепта выражения INSERT ALL and INSERT FIRST обеспечили бы один и тот же результат, но между ними есть отличие. INSERT FIRST завершит вычисление WHEN-THEN-ELSE, как только условие будет выполнено; INSERT ALL проверит все условия, даже если предыдущие условия выполнены. Таким образом, INSERT ALL может использоваться для вставки одной строки в несколько таблиц.

DB2

Мое решение для DB2 немного сбивает с толку. Оно требует, чтобы на вставляемые таблицы были наложены ограничения, гарантирующие, что каждая строка, возвращаемая подзапросом, будет направлена в соответствующую таблицу. Техника заключается во вставке в представление, которое определено как объединение всех (UNION ALL) таблиц. Если проверочные ограничения выражения INSERT не уникальны (т. е. несколько таблиц имеют одинаковое проверочное ограничение), выражение INSERT не будет знать, где размещать строки, и даст сбой.

MySQL, PostgreSQL и SQL Server

На момент написания данной книги только Oracle и DB2 предоставляли механизмы для вставки строк, возвращенных запросом, в одну или более таблиц в рамках одного выражения.

Блокировка вставки в определенные столбцы

Задача

Требуется предотвратить вставку значений в определенные столбцы таблицы пользователями или программным обеспечением. Например, вы хотите предоставить программе возможность вставлять значения в таблицу EMP, но только в столбцы EMPNO, ENAME и JOB.

Решение

Создайте представление, базированное на данной таблице, содержащее столбцы, доступные для вставки. Затем обеспечьте, чтобы вся вставка осуществлялась только посредством этого представления. Например, создадим представление, содержащее три столбца таблицы EMP:

```
create view new_emps as
select empno, ename, job
  from emp
```

Открывая доступ к этому представлению, вы разрешаете пользователям и программам вставлять данные только в эти три поля представления. Не давайте пользователям разрешение на вставку в таблицу EMP напрямую. Пользователи получат возможность создавать новые записи в EMP, вставляя данные в представление NEW_EMPS, но они смогут размещать значения только в трех столбцах, которые заданы в описании представления.

Обсуждение

При вставке в простое представление, такое как приведено в данном решении, сервер базы данных вставит эти данные в базовую таблицу. Например, следующая вставка:

```
insert into new_emps
   (empno, ename, job)
   values (1, 'Jonathan', 'Editor')
```

будет преобразована сервером в:

```
insert into emp
   (empno, ename, job)
   values (1, 'Jonathan', 'Editor')
```

Также можно, но, вероятно, имеет меньший практический смысл, вставлять данные во вложенное представление (в настоящее время поддерживается только Oracle):

```
insert into
   (select empno, ename, job
      from emp)
values (1, 'Jonathan', 'Editor')
```

Вставка в представления – сложный вопрос. Правила сильно усложняются с повышением сложности представлений. Если вы планируете использовать возможность вставки в представления, обязательно досконально изучите и разберитесь с документацией производителя, касающейся этого вопроса.

Изменение записей в таблице

Задача

Требуется изменить значения некоторых или всех строк таблицы. Например, повысить заработные платы всех служащих 20-го отдела на 10%. В следующем результирующем множестве представлены столбцы DEPTNO, ENAME и SAL с данными служащих этого отдела:

```
select deptno,ename,sal
  from emp
```

```
 where deptno = 20
 order by 1,3

DEPTNO ENAME             SAL
------ ---------- ----------
    20 SMITH             800
    20 ADAMS            1100
    20 JONES            2975
    20 SCOTT            3000
    20 FORD             3000
```

Требуется увеличить все значения столбца SAL на 10%.

Решение

Для изменения существующих строк таблицы базы данных используйте выражение UPDATE (обновить). Например:

```
1 update emp
2    set sal = sal*1.10
3  where deptno = 20
```

Обсуждение

Используйте выражение UPDATE с предикатом WHERE, чтобы указать, какие строки подлежат обновлению. Если опустить предикат WHERE, обновлены будут все строки. Выражение SAL*1.10 в данном решении возвращает заработную плату, увеличенную на 10%.

При подготовке к масштабным обновлениям может потребоваться предварительно просмотреть результаты. Сделать это можно, создав выражение SELECT, включающее выражения, которые планируется поместить в операторы SET. Следующее выражение SELECT показывает результат повышения зарплаты на 10%:

```
select deptno,
       ename,
       sal      as orig_sal,
       sal*.10  as amt_to_add,
       sal*1.10 as new_sal
   from emp
  where deptno=20
  order by 1,5

DEPTNO ENAME  ORIG_SAL AMT_TO_ADD NEW_SAL
------ ------ -------- ---------- -------
    20 SMITH       800         80     880
    20 ADAMS      1100        110    1210
    20 JONES      2975        298    3273
    20 SCOTT      3000        300    3300
    20 FORD       3000        300    3300
```

Результат повышения заработной платы представлен в двух столбцах: в одном показана сумма, на которую была повышена заработная плата, а во втором – новая заработная плата.

Обновление в случае существования соответствующих строк в другой таблице

Задача

Требуется обновить строки одной таблицы, только если соответствующие строки существуют в другой таблице. Например, если данные о служащем имеются в таблице EMP_BONUS, его заработная плата (данные о которой хранятся в таблице EMP) должна повыситься на 20%. Следующее результирующее множество представляет данные, имеющиеся в настоящий момент в таблице EMP_BONUS:

```
select empno, ename
  from emp_bonus

     EMPNO ENAME
---------- ----------
      7369 SMITH
      7900 JAMES
      7934 MILLER
```

Решение

Чтобы найти в EMP служащих, которые есть в таблице EMP_BONUS, используйте подзапрос в предикате WHERE выражения UPDATE. Тогда выражение UPDATE будет применено только к строкам, возвращенным подзапросом, что обеспечит возможность увеличить заработную плату соответствующим работникам:

```
1 update emp
2    set sal=sal*1.20
3  where empno in ( select empno from emp_bonus )
```

Обсуждение

В результате подзапроса возвращаются строки, которые будут обновлены в таблице EMP. Предикат IN проверяет, есть ли значения столбца EMPNO таблицы EMP среди значений EMPNO, возвращенных запросом. Если он находит такое значение, обновляется соответствующее ему значение столбца SAL.

В качестве альтернативы вместо IN можно использовать EXISTS:

```
update emp
   set sal = sal*1.20
 where exists ( select null
                  from emp_bonus
                 where emp.empno=emp_bonus.empno )
```

Может вызвать удивление присутствие значения NULL в списке SELECT подзапроса EXISTS. Ничего страшного, этот NULL не оказывает неблагоприятного эффекта на обновление. По-моему, он делает код бо-

лее понятным, поскольку подчеркивает тот факт, что в отличие от решения с использованием подзапроса с оператором IN условие обновления (т. е. какие строки будут обновлены) будет контролироваться предикатом WHERE подзапроса, а не значениями, возвращенными как результат списка SELECT подзапроса.

Обновление значениями из другой таблицы

Задача

Требуется обновить строки одной таблицы, используя значения другой таблицы. Например, имеется таблица NEW_SAL, в которой хранятся новые размеры заработных плат определенных служащих. Содержимое таблицы NEW_SAL:

```
select *
  from new_sal

DEPTNO        SAL
------ ----------
    10       4000
```

Столбец DEPTNO – это первичный ключ таблицы NEW_SAL. Требуется обновить заработные платы и комиссионные определенных служащих в таблице EMP, используя значения таблицы NEW_SAL. Если значения столбцов EMP.DEPTNO и NEW_SAL.DEPTNO совпадают, то значение EMP.SAL обновляется значением NEW_SAL.SAL, а значение столбца EMP.COMM увеличивается на 50% от значения NEW_SAL.SAL. Вот строки таблицы EMP:

```
select deptno,ename,sal,comm
  from emp
 order by 1

DEPTNO ENAME             SAL       COMM
------ ---------- ---------- ----------
    10 CLARK            2450
    10 KING             5000
    10 MILLER           1300
    20 SMITH             800
    20 ADAMS            1100
    20 FORD             3000
    20 SCOTT            3000
    20 JONES            2975
    30 ALLEN            1600        300
    30 BLAKE            2850
    30 MARTIN           1250       1400
    30 JAMES             950
    30 TURNER           1500          0
    30 WARD             1250        500
```

Решение

Для поиска и возвращения новых значений COMM в выражение UPDATE используйте объединение таблиц NEW_SAL и EMP. Довольно часто такие обновления реализуются посредством связанных подзапросов. Другая техника предполагает создание представления (обычного или вложенного, в зависимости от того, что поддерживает СУБД) и последующее его обновление.

DB2 и MySQL

С помощью связанного подзапроса задайте новые значения столбцам SAL и COMM в таблице EMP. Используйте также связаный подзапрос, чтобы определить, какие строки EMP подлежат обновлению:

```
1  update emp e set (e.sal,e.comm) = (select ns.sal, ns.sal/2
2                                       from new_sal ns
3                                      where ns.deptno=e.deptno)
4   where exists ( select null
5                    from new_sal ns
6                   where ns.deptno = e.deptno )
```

Oracle

Метод, используемый в решении для DB2, несомненно, подходит и для Oracle, но в качестве альтернативы можно использовать в выражении UPDATE вложенное представление:

```
1 update (
2  select e.sal  as emp_sal, e.comm  as emp_comm,
3         ns.sal as ns_sal, ns.sal/2 as ns_comm
4    from emp e, new_sal ns
5   where e.deptno = ns.deptno
6  ) set emp_sal = ns_sal, emp_comm = ns_comm
```

PostgreSQL

Метод, используемый в решении для DB2, подойдет и для PostgreSQL, но в качестве альтернативы можно (что довольно удобно) проводить объединение непосредственно в выражении UPDATE:

```
1  update emp
2     set sal  = ns.sal,
3         comm = ns.sal/2
4    from new_sal ns
5   where ns.deptno = emp.deptno
```

SQL Server

Необходимо (как и в решении для PostgreSQL) проводить объединение прямо в выражении UPDATE:

```
1  update e
2     set e.sal  = ns.sal,
```

```
3         e.comm = ns.sal/2
4    from emp e,
5         new_sal ns
6   where ns.deptno = e.deptno
```

Обсуждение

Прежде чем приступить к обсуждению решений, мне хотелось бы напомнить несколько важных моментов относительно обновлений, при которых запросы используются для получения новых значений. Блок WHERE в подзапросе связанного обновления и блок WHERE, применяемый к обновляемой таблице, – совершенно разные вещи. Посмотрим на выражение UPDATE, представленное в разделе «Решение»: происходит объединение таблиц EMP и NEW_SAL по столбцу DEPTNO, и строки возвращаются в оператор SET выражения UPDATE. Для служащих 10-го отдела (DEPTNO 10) возвращаются значения, потому что в таблице NEW_SAL есть соответствующие им значения DEPTNO. А что же со служащими других отделов? В NEW_SAL иные отделы не представлены, поэтому столбцам SAL и COMM для служащих других отделов (DEPTNO 20 и 30) присваиваются значения NULL. Если производитель не предоставляет какого-либо механизма ограничения числа строк, возвращаемых в результирующем множестве (например, функции LIMIT или TOP), то единственным средством для этого в SQL является использование блока WHERE. Правильное выполнение этого UPDATE обеспечивает блок WHERE, применяемый к обновляемой таблице, в сочетании с блоком WHERE в связанном запросе.

DB2 и MySQL

Чтобы в таблице EMP обновлялись не все строки, необходимо включить в блок WHERE выражения UPDATE связанный подзапрос. Выноса объединения (связанный подзапрос) в оператор SET недостаточно. Использование блока WHERE в UPDATE гарантирует, что обновлены будут только те строки таблицы EMP, значение столбца DEPTNO которых совпадает со значением этого столбца в таблице NEW_SAL. Это правомочно для всех СУБД.

Oracle

В решении Oracle с обновлением объединенного представления для выявления строк, подлежащих обновлению, используются эквиобъединения. Убедиться в том, какие строки обновляются, можно, выполнив запрос отдельно. Чтобы успешно пользоваться обновлением такого типа, необходимо прежде всего понять принцип сохранения ключей. Столбец DEPTNO таблицы NEW_SAL является первичным ключом этой таблицы, таким образом, его значения уникальны в рамках таблицы. Однако при объединении таблиц EMP и NEW_SAL значения столбца NEW_SAL.DEPTNO в результирующем множестве не уникальны, как можно видеть ниже:

```
select e.empno, e.deptno e_dept, ns.sal, ns.deptno ns_deptno
   from emp e, new_sal ns
  where e.deptno = ns.deptno

EMPNO     E_DEPT        SAL  NS_DEPTNO
----- ---------- ---------- ----------
 7782         10       4000         10
 7839         10       4000         10
 7934         10       4000         10
```

Чтобы в Oracle можно было провести обновление этого объединения, одна из таблиц должна быть сохраняющей ключи (key-preserved). Это означает, что если значения неуникальны в результирующем множестве, то они должны быть уникальными, по крайней мере, в своей исходной таблице. В данном случае DEPTNO является первичным ключом NEW_SAL, что делает его значения уникальными в этой таблице. Они могут многократно повторяться в результирующем множестве, но исходная таблица все равно остается сохраняющей ключи, поскольку в ней эти значения уникальны, что позволяет успешно выполнить обновление.

PostgreSQL и SQL Server

Синтаксис для этих двух платформ немного отличается, но принцип тот же. Очень удобно осуществлять объединение непосредственно в выражении UPDATE. Поскольку обновляемая таблица задается явно (таблицы перечисляются после ключевого слова UPDATE), то не возникает путаницы с тем, строки какой таблицы изменяются. Кроме того, в связи с тем, что объединения осуществляются в процессе обновления (поскольку присутствует явный блок WHERE), можно избежать некоторых ошибок при реализации обновлений с использованием связанных подзапросов; в частности, если здесь пропустить объединение, то ошибка будет очевидной.

Слияние записей

Задача

Требуется вставлять, обновлять или удалять записи таблицы на основании условия существования или отсутствия соответствующих записей. (Если запись существует, проводится ее обновление; если нет, то она вставляется; если после обновления строка не отвечает определенному условию, то она удаляется.) Например, необходимо изменить таблицу EMP_COMMISSION по следующим правилам:

- Если запись о служащем, представленном в EMP_COMMISSION, также существует в таблице EMP, обновить его комиссионные (COMM) до 1000.
- Удалить всех служащих, которым потенциально обеспечено повышение комиссионных (COMM) до 1000, если их заработная плата (SAL) меньше 2000 (их не должно быть в EMP_COMMISSION).

- В противном случае перенести значения столбцов EMPNO, ENAME и DEPTNO из таблицы EMP в таблицу EMP_COMMISSION.

По сути, стоит задача обновления (UPDATE) или вставки (INSERT) в зависимости от того, имеет ли данная строка EMP соответствие в EMP_COMMISSION. Причем, если в результате выполнения UPDATE комиссионные оказываются слишком велики, требуется удалить эту строку (выполнить операцию DELETE).

В настоящий момент в таблицах EMP и EMP_COMMISSION имеются следующие строки соответственно:

```
select deptno,empno,ename,comm
  from emp
 order by 1
```

```
DEPTNO      EMPNO ENAME        COMM
------ ---------- ------ ----------
    10       7782 CLARK
    10       7839 KING
    10       7934 MILLER
    20       7369 SMITH
    20       7876 ADAMS
    20       7902 FORD
    20       7788 SCOTT
    20       7566 JONES
    30       7499 ALLEN         300
    30       7698 BLAKE
    30       7654 MARTIN       1400
    30       7900 JAMES
    30       7844 TURNER          0
    30       7521 WARD          500
```

```
select deptno,empno,ename,comm
  from emp_commission
 order by 1
```

```
    DEPTNO      EMPNO ENAME            COMM
---------- ---------- ---------- ----------
        10       7782 CLARK
        10       7839 KING
        10       7934 MILLER
```

Решение

В настоящее время Oracle является единственной СУБД[1], имеющей выражение для решения этой задачи. Это выражение – MERGE (слия-

[1] На момент выпуска книги в предварительных версиях Microsoft SQL Server 2008 данная возможность также доступна; синтаксис схож с тем, что используется для Oracle. Для более детального изучения обратитесь к документации по Microsoft SQL Server 2008. – *Примеч. науч. ред.*

ние), которое может по необходимости осуществлять обновление (UPDATE) или вставку (INSERT). Например:

```
1  merge into emp_commission ec
2  using (select * from emp) emp
3     on (ec.empno=emp.empno)
4   when matched then
5        update set ec.comm = 1000
6        delete where (sal < 2000)
7   when not matched then
8        insert (ec.empno,ec.ename,ec.deptno,ec.comm)
9        values (emp.empno,emp.ename,emp.deptno,emp.comm)
```

Обсуждение

Объединение в строке 3 решения определяет, какие строки уже существуют и будут обновлены. Объединяются таблица EMP_COMMISSION (под псевдонимом EC) и результат подзапроса (под псевдонимом EMP). Если объединение успешно, две строки считаются «совпавшими», и выполняется UPDATE в выражении WHEN MATCHED. В противном случае соответствие не обнаружено, и выполняется INSERT в выражении WHEN NOT MATCHED. Таким образом, строки таблицы EMP, значения столбца EMPNO которых не совпадают со значениями этого столбца ни в одной строке таблицы EMP_COMMISSION, будут вставлены в EMP_COMMISSION. Значения COMM в таблице EMP_COMMISSION будут обновлены только в тех записях служащих, для которых в таблице EMP столбец DEPTNO имеет значение 10. Записи всех остальных служащих будут просто вставлены. Кроме того, поскольку служащий MILLER имеет DEPTNO 10, он является кандидатом на обновление COMM, но в связи с тем, что его SAL меньше 2000, его запись удаляется из таблицы EMP_COMMISSION.

Удаление всех записей из таблицы

Задача

Требуется удалить все записи из таблицы.

Решение

Чтобы удалить записи из таблицы, используйте команду DELETE. Например, чтобы уничтожить все записи из таблицы EMP:

```
delete from emp
```

Обсуждение

При использовании команды DELETE без предиката WHERE удаляются все записи из заданной таблицы.

Удаление определенных записей

Задача

Требуется удалить из таблицы записи, отвечающие определенному критерию.

Решение

Используйте команду DELETE с предикатом WHERE, определяющим, какие строки подлежат удалению. Например, удалим всех служащих 10-го отдела:

```
delete from emp where deptno = 10
```

Обсуждение

Применяя команду DELETE в сочетании с блоком WHERE, можно удалять не все строки из таблицы, а только некоторые из них.

Удаление одной записи

Задача

Требуется удалить одну запись из таблицы.

Решение

Это особый случай удаления определенных записей. Суть в том, что критерий отбора должен быть настолько узок, чтобы он обеспечивал выбор только одной строки. Часто в качестве критерия для удаления используется первичный ключ. Например, удалим запись о служащем CLARK (EMPNO 7782):

```
delete from emp where empno = 7782
```

Обсуждение

Суть удаления – в определении строк, которые должны быть удалены; воздействие DELETE всегда определяется его предикатом WHERE. Опустите WHERE, и областью действия DELETE станет вся таблица. Задавая предикат WHERE, мы можем сузить эту область до определенной группы строк или до одной строки. При удалении одной строки она обычно идентифицируется с помощью первичного ключа или одного из ее уникальных ключей.

Если критерий удаления определен первичным или уникальным ключом, можно с уверенностью ожидать, что будет удалена только одна строка. (Потому что СУБД не допустит, чтобы две строки имели одинаковый первичный или уникальный ключ.) В противном случае может потребоваться проверка, удостоверяющая, что случайно не будут удалены лишние строки.

Удаление записей, которые нарушают ссылочную целостность

Задача

Требуется удалить записи из таблицы, если они ссылаются на несуществующие записи другой таблицы. Пример: некоторые служащие приписаны к отделу, которого не существует. Необходимо удалить записи этих служащих.

Решение

Для проверки действительности номеров отделов используйте предикат NOT EXISTS с подзапросом:

```
delete from emp
 where not exists (
   select * from dept
    where dept.deptno = emp.deptno
 )
```

Или можно написать запрос с предикатом NOT IN:

```
delete from emp
where deptno not in (select deptno from dept)
```

Обсуждение

При удалении самое главное – правильно описать в предикате WHERE строки, подлежащие удалению.

В решении с NOT EXISTS используется связанный подзапрос для проверки существования в таблице DEPT записи, значение DEPTNO которой соответствует DEPTNO заданной записи таблицы EMP. Если такая запись есть, то запись EMP сохраняется. В противном случае она удаляется. Такая проверка проводится для всех записей таблицы EMP.

В решении с предикатом IN используется подзапрос для извлечения списка действительных номеров отделов. Затем значения столбца DEPTNO всех записей таблицы EMP сравниваются с этим списком. Если обнаруживается запись, DEPTNO которой не соответствует значениям списка, она удаляется из таблицы EMP.

Уничтожение дублирующихся записей

Задача

Требуется уничтожить дублирующиеся записи из таблицы. Рассмотрим следующую таблицу:

```
create table dupes (id integer, name varchar(10))

insert into dupes values (1, 'NAPOLEON')
```

```
insert into dupes values (2, 'DYNAMITE')
insert into dupes values (3, 'DYNAMITE')
insert into dupes values (4, 'SHE SELLS')
insert into dupes values (5, 'SEA SHELLS')
insert into dupes values (6, 'SEA SHELLS')
insert into dupes values (7, 'SEA SHELLS')

select * from dupes order by 1

        ID NAME
---------- ----------
         1 NAPOLEON
         2 DYNAMITE
         3 DYNAMITE
         4 SHE SELLS
         5 SEA SHELLS
         6 SEA SHELLS
         7 SEA SHELLS
```

Для каждой группы дублирующихся имен, таких как «SEA SHELLS», необходимо сохранить один произвольно выбранный ID и уничтожить остальные. В случае с «SEA SHELLS» не важно, будут ли уничтожены записи с ID 5 и 6, или 5 и 7, или 6 и 7, но в итоге должна остаться всего одна запись для «SEA SHELLS».

Решение

Организуйте произвольный выбор сохраняемого ID с помощью подзапроса с агрегатной функцией, такой как MIN (в данном случае в таблице остается только запись с минимальным значением ID).

```
1  delete from dupes
2   where id not in ( select min(id)
3                       from dupes
4                      group by name )
```

Обсуждение

Первое, что необходимо сделать при удалении дубликатов, – точно определить, какие две строки считаются «дубликатами». Для моего примера в данном рецепте «дубликат» – это две записи, содержащие одинаковое значение в столбце NAME. Вооружившись этим определением, мы получаем множество дубликатов и уже выбираем другие столбцы как условие сохранения записей из этого набора. Лучше всего, если этим определяющим столбцом (или столбцами) будет первичный ключ. Я использовал столбец ID, потому что двух записей с одинаковым ID не существует.

Суть решения в том, что строки группируются по дублирующимся значениям (в данном случае, по значению столбца NAME), и затем с помощью агрегатной функции из группы выбирается всего одно значение, которое остается в таблице. Подзапрос в примере раздела «Ре-

шение» возвращает минимальный ID для каждого значения NAME, представляющий строку, которая не будет удалена:

```
select min(id)
  from dupes
 group by name

   MIN(ID)
----------
         2
         1
         5
         4
```

Затем команда DELETE удаляет из таблицы все строки с ID, не возвращенными подзапросом (в данном случае это 3, 6 и 7). Если возникают трудности с пониманием того, что происходит, выполните сначала подзапрос и включите NAME в список SELECT:

```
select name, min(id)
  from dupes
 group by name

NAME          MIN(ID)
---------- ----------
DYNAMITE            2
NAPOLEON            1
SEA SHELLS          5
SHE SELLS           4
```

Подзапрос возвращает строки, которые будут сохранены. Предикат NOT IN в выражении DELETE обусловливает удаление всех остальных строк.

Удаление записей, на которые есть ссылки в другой таблице

Задача

Требуется удалить из одной таблицы записи, на которые ссылается другая таблица. Рассмотрим следующую таблицу, DEPT_ACCIDENTS, в которой содержится по одной строке для каждого несчастного случая, произошедшего на производстве. Каждая строка включает номер отдела, в котором имело место происшествие, а также род происшествия.

```
create table dept_accidents
( deptno         integer,
  accident_name  varchar(20) )

insert into dept_accidents values (10,'BROKEN FOOT')
insert into dept_accidents values (10,'FLESH WOUND')
insert into dept_accidents values (20,'FIRE')
```

```
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FLOOD')
insert into dept_accidents values (30,'BRUISED GLUTE')

select * from dept_accidents

    DEPTNO ACCIDENT_NAME
---------- --------------------
        10 BROKEN FOOT
        10 FLESH WOUND
        20 FIRE
        20 FIRE
        20 FLOOD
        30 BRUISED GLUTE
```

Стоит задача удалить из таблицы EMP записи служащих, работающих в отделе, в котором произошло три или более несчастных случаев.

Решение

Чтобы найти отделы с тремя или более несчастными случаями, используйте подзапрос и агрегатную функцию COUNT. Затем удалите записи всех служащих, работающих в этих отделах:

```
1 delete from emp
2  where deptno in ( select deptno
3                      from dept_accidents
4                     group by deptno
5                    having count(*) >= 3 )
```

Обсуждение

Подзапрос определит, в каких отделах произошло три или более несчастных случая:

```
select deptno
  from dept_accidents
 group by deptno
having count(*) >= 3

    DEPTNO
----------
        20
```

DELETE удалит записи о всех служащих отделов, возвращенных подзапросом (в данном случае только 20-го отдела).

5

Запросы на получение метаданных

В этой главе представлены рецепты, позволяющие находить информацию о конкретной схеме. Например, может потребоваться узнать, какие таблицы были созданы или какие внешние ключи непроиндексированы. Все обсуждаемые здесь СУБД предоставляют таблицы и представления для получения подобных данных. Предложенные рецепты помогут научиться извлекать информацию из этих таблиц и представлений. Однако рассматриваемый вопрос намного шире, чем может охватить данная книга. Полный список таблиц/представлений каталога или словаря данных можно найти в документации СУБД.

Для простоты демонстрации все рецепты данной главы работают со схемой SMEAGOL.

Как получить список таблиц схемы

Задача

Требуется получить список всех таблиц, созданных в данной схеме.

Решение

Во всех решениях предполагается, что мы работаем со схемой SMEAGOL. Базовый подход к решению данной задачи одинаков для всех СУБД: делается запрос к системной таблице (или представлению), в которой содержатся записи для каждой таблицы базы данных.

DB2

Делаем запрос к SYSCAT.TABLES:

```
1 select tabname
```

```
2   from syscat.tables
3  where tabschema = 'SMEAGOL'
```

Oracle

Делаем запрос к SYS.ALL_TABLES:

```
select table_name
  from all_tables
 where owner = 'SMEAGOL'
```

PostgreSQL, MySQL и SQL Server

Делаем запрос к INFORMATION_SCHEMA.TABLES:

```
1 select table_name
2   from information_schema.tables
3  where table_schema = 'SMEAGOL'
```

Обсуждение

Базы данных предоставляют информацию о себе посредством тех же механизмов, которые вы используете в своих приложениях: таблиц и представлений. Oracle, например, содержит обширный набор системных представлений, ALL_TABLES, в которых предоставляется информация о таблицах, индексах, правах доступа и любых других объектах базы данных.

Системные представления Oracle – это именно представления. В их основе лежит базовый набор таблиц, содержащих информацию в очень неудобной для пользователей форме. Эти представления приводят данные каталога Oracle в вид, с которым можно работать.

В Oracle используются системные представления, а в DB2 – системные таблицы. PostgreSQL, MySQL и SQL Server[1], с другой стороны, поддерживают *информационную* схему, которая является набором представлений, определенных стандартом ISO SQL. Поэтому для всех трех баз данных может использоваться один и тот же запрос.

Как получить список столбцов таблицы

Задача

Требуется получить список столбцов таблицы, их типы и позиции в таблице.

---

[1] СУБД MySQL и SQL Server также имеют собственные системные представления, таблицы или команды для предоставления более детальной информации об объектах схемы. Например, SQL Server 2005 имеет системное представление sys.tables с информацией о таблицах. – *Примеч. науч. ред.*

Решение

Следующие решения предполагают, что требуется представить список столбцов, их типов и порядковых номеров для таблицы EMP схемы SMEAGOL.

DB2

Делаем запрос к SYSCAT.COLUMNS:

```
1 select colname, typename, colno
2   from syscat.columns
3  where tabname   = 'EMP'
4    and tabschema = 'SMEAGOL'
```

Oracle

Делаем запрос к ALL_TAB_COLUMNS:

```
1  select column_name, data_type, column_id
2    from all_tab_columns
3   where owner      = 'SMEAGOL'
4     and table_name = 'EMP'
```

PostgreSQL, MySQL и SQL Server

Делаем запрос к INFORMATION_SCHEMA.COLUMNS:

```
1 select column_name, data_type, ordinal_position
2   from information_schema.columns
3  where table_schema = 'SMEAGOL'
4    and table_name   = 'EMP'
```

Обсуждение

Каждый производитель предоставляет средства для получения подробной информации о столбцах. В приведенных выше примерах возвращаются только имена столбцов, их типы и порядковые номера. Кроме того, можно получить полезную информацию о длине столбца, возможности сохранения в нем значения NULL и его значениях по умолчанию.

Как получить список индексированных столбцов таблицы

Задача

Требуется получить список индексов, их столбцов и позиций столбцов (если такая информация доступна) в индексе для данной таблицы.

Решение

Во всех приведенных ниже специальных решениях для разных СУБД предполагается, что составляется список индексов таблицы EMP схемы SMEAGOL.

DB2

Делаем запрос к SYSCAT.INDEXES:

```
1  select a.tabname, b.indname, b.colname, b.colseq
2    from syscat.indexes a,
3         syscat.indexcoluse b
3   where a.tabname   = 'EMP'
4     and a.tabschema = 'SMEAGOL'
5     and a.indschema = b.indschema
6     and a.indname   = b.indname
```

Oracle

Делаем запрос к SYS.ALL_IND_COLUMNS:

```
select table_name, index_name, column_name, column_position
  from sys.all_ind_columns
 where table_name  = 'EMP'
   and table_owner = 'SMEAGOL'
```

PostgreSQL

Делаем запрос к PG_CATALOG.PG_INDEXES и INFORMATION_SCHEMA.COLUMNS:

```
1  select a.tablename,a.indexname,b.column_name
2    from pg_catalog.pg_indexes a,
3         information_schema.columns b
4   where a.schemaname = 'SMEAGOL'
5     and a.tablename  = b.table_name
```

MySQL

Используем команду SHOW INDEX:

```
show index from emp
```

SQL Server

Делаем запрос к SYS.TABLES, SYS.INDEXES, SYS.INDEX_COLUMNS и SYS.COLUMNS:

```
1  select a.name table_name,
2         b.name index_name,
3         d.name column_name,
4         c.index_column_id
5    from sys.tables a,
6         sys.indexes b,
7         sys.index_columns c,
```

```
 8         sys.columns d
 9  where a.object_id = b.object_id
10    and b.object_id = c.object_id
11    and b.index_id  = c.index_id
12    and c.object_id = d.object_id
13    and c.column_id = d.column_id
14    and a.name      = 'EMP'
```

Обсуждение

Делая запросы к таблице, важно знать, какие из ее столбцов проиндексированы, а какие – нет. Индексы могут обеспечить хорошую производительность запросов к столбцам, которые часто используются в фильтрах и к которым осуществляется много запросов. Индексы также полезны при объединении таблиц. Зная, какие столбцы индексированы, вы предупреждаете возможные проблемы с производительностью. Кроме того, может потребоваться информация о самих индексах: их глубине вложенности, количестве уникальных ключей в них, разветвленности и т. д. Подобную информацию можно получить из представлений/таблиц, запросы к которым показаны в решениях данного рецепта.

Как получить список ограничений, наложенных на таблицу

Задача

Требуется создать список ограничений, определенных для таблицы в некоторой схеме, и столбцов, на которые эти ограничения наложены. Например, стоит задача получить ограничения и столбцы, на которые они наложены, для таблицы EMP.

Решение

DB2

Делаем запрос к SYSCAT.TABCONST и SYSCAT.COLUMNS:

```
1  select a.tabname, a.constname, b.colname, a.type
2    from syscat.tabconst a,
3         syscat.columns b
4   where a.tabname   = 'EMP'
5     and a.tabschema = 'SMEAGOL'
6     and a.tabname   = b.tabname
7     and a.tabschema = b.tabschema
```

Oracle

Делаем запрос к SYS.ALL_CONSTRAINTS и SYS.ALL_CONS_COLUMNS:

```
1  select a.table_name,
2         a.constraint_name,
```

```
 3         b.column_name,
 4         a.constraint_type
 5    from all_constraints a,
 6         all_cons_columns b
 7   where a.table_name      = 'EMP'
 8     and a.owner           = 'SMEAGOL'
 9     and a.table_name      = b.table_name
10     and a.owner           = b.owner
11     and a.constraint_name = b.constraint_name
```

PostgreSQL, MySQL и SQL Server

Делаем запрос к INFORMATION_SCHEMA.TABLE_CONSTRAINTS и INFORMATION_ SCHEMA.KEY_COLUMN_USAGE:

```
 1  select a.table_name,
 2         a.constraint_name,
 3         b.column_name,
 4         a.constraint_type
 5    from information_schema.table_constraints a,
 6         information_schema.key_column_usage b
 7   where a.table_name      = 'EMP'
 8     and a.table_schema    = 'SMEAGOL'
 9     and a.table_name      = b.table_name
10     and a.table_schema    = b.table_schema
11     and a.constraint_name = b.constraint_name
```

Обсуждение

Ограничения – настолько важная часть реляционных баз данных, что даже не стоит объяснять, зачем необходимо знать, какие ограничения наложены на таблицу. Список наложенных на таблицу ограничений может понадобиться для разных целей: если необходимо найти таблицу без первичного ключа, если необходимо выяснить, какие столбцы должны быть внешними ключами, но ими не являются (т. е. данные дочерних таблиц отличаются от данных родительских таблиц, и вы хотите выяснить, почему это произошло), или если необходимо узнать проверочные ограничения (Допускает ли столбец значения NULL? Должны ли значения столбца удовлетворять какому-то специальному условию? и др.).

Как получить список внешних ключей без соответствующих индексов

Задача

Требуется составить список таблиц, в которых есть непроиндексированные столбцы внешних ключей. Например, вы хотите знать, проиндексированы ли внешние ключи таблицы EMP.

Решение

DB2

Делаем запрос к SYSCAT.TABCONST, SYSCAT.KEYCOLUSE, SYSCAT.INDEXES и SYSCAT.INDEXCOLUSE:

```
 1  select fkeys.tabname,
 2         fkeys.constname,
 3         fkeys.colname,
 4         ind_cols.indname
 5    from (
 6  select a.tabschema, a.tabname, a.constname, b.colname
 7    from syscat.tabconst a,
 8         syscat.keycoluse b
 9   where a.tabname   = 'EMP'
10     and a.tabschema = 'SMEAGOL'
11     and a.type      = 'F'
12     and a.tabname   = b.tabname
13     and a.tabschema = b.tabschema
14         ) fkeys
15         left join
16         (
17  select a.tabschema,
18         a.tabname,
19         a.indname,
20         b.colname
21    from syscat.indexes a,
22         syscat.indexcoluse b
23   where a.indschema = b.indschema
24     and a.indname   = b.indname
25         ) ind_cols
26      on (     fkeys.tabschema = ind_cols.tabschema
27           and fkeys.tabname   = ind_cols.tabname
28           and fkeys.colname   = ind_cols.colname )
29   where ind_cols.indname is null
```

Oracle

Делаем запрос к SYS.ALL_CONS_COLUMNS, SYS.ALL_CONSTRAINTS и SYS.ALL_ IND_COLUMNS:

```
 1  select a.table_name,
 2         a.constraint_name,
 3         a.column_name,
 4         c.index_name
 5    from all_cons_columns a,
 6         all_constraints b,
 7         all_ind_columns c
 8   where a.table_name       = 'EMP'
 9     and a.owner            = 'SMEAGOL'
10     and b.constraint_type  = 'R'
11     and a.owner            = b.owner
```

```
    and a.table_name       = b.table_name
    and a.constraint_name  = b.constraint_name
    and a.owner            = c.table_owner  (+)
    and a.table_name       = c.table_name   (+)
    and a.column_name      = c.column_name  (+)
    and c.index_name       is null
```

PostgreSQL

Делаем запрос к INFORMATION_SCHEMA.KEY_COLUMN_USAGE, INFORMATION_SCHEMA.REFERENTIAL_CONSTRAINTS, INFORMATION_SCHEMA.COLUMNS и PG_CATALOG.PG_INDEXES:

```
select fkeys.table_name,
       fkeys.constraint_name,
       fkeys.column_name,
       ind_cols.indexname
  from (
select a.constraint_schema,
       a.table_name,
       a.constraint_name,
       a.column_name
  from information_schema.key_column_usage a,
       information_schema.referential_constraints b
 where a.constraint_name    = b.constraint_name
   and a.constraint_schema  = b.constraint_schema
   and a.constraint_schema  = 'SMEAGOL'
   and a.table_name         = 'EMP'
       ) fkeys
       left join
       (
select a.schemaname, a.tablename, a.indexname, b.column_name
  from pg_catalog.pg_indexes a,
       information_schema.columns b
 where a.tablename  = b.table_name
   and a.schemaname = b.table_schema
       ) ind_cols
    on (     fkeys.constraint_schema = ind_cols.schemaname
         and fkeys.table_name        = ind_cols.tablename
         and fkeys.column_name       = ind_cols.column_name )
 where ind_cols.indexname is null
```

MySQL

Для получения информации об индексах, такой как имя индекса, столбцы в индексе и порядковые номера столбцов в индексе, можно использовать команду SHOW INDEX (показать индекс). Кроме того, можно запросить INFORMATION_SCHEMA.KEY_COLUMN_USAGE для получения списка внешних ключей данной таблицы. В MySQL 5 внешние ключи должны индексироваться автоматически, но порой этого не происходит. Чтобы выяснить, проиндексирован ли столбец внешнего ключа, выполняется команда SHOW INDEX для конкретной

таблицы и результат сравнивается с полученным INFORMATION_SCHEMA.KEY_COLUMN_USAGE.COLUMN_NAME для той же таблицы. Если COLUMN_NAME есть в KEY_COLUMN_USAGE, но не возвращено в результате выполнения SHOW INDEX, делаем вывод, что столбец непроиндексирован.

SQL Server

Делаем запрос к SYS.TABLES, SYS.FOREIGN_KEYS, SYS.COLUMNS, SYS.INDEXES и SYS.INDEX_COLUMNS:

```
 1  select fkeys.table_name,
 2         fkeys.constraint_name,
 3         fkeys.column_name,
 4         ind_cols.index_name
 5    from (
 6  select a.object_id,
 7         d.column_id,
 8         a.name table_name,
 9         b.name constraint_name,
10         d.name column_name
11    from sys.tables a
12         join
13         sys.foreign_keys b
14      on (    a.name      = 'EMP'
15          and a.object_id = b.parent_object_id
16         )
17         join
18         sys.foreign_key_columns c
19      on ( b.object_id = c.constraint_object_id )
20         join
21         sys.columns d
22      on (    c.constraint_column_id = d.column_id
23          and a.object_id            = d.object_id
24         )
25         ) fkeys
26         left join
27         (
28  select a.name index_name,
29         b.object_id,
30         b.column_id
31    from sys.indexes a,
32         sys.index_columns b
33   where a.index_id = b.index_id
34         ) ind_cols
35      on (     fkeys.object_id = ind_cols.object_id
36           and fkeys.column_id = ind_cols.column_id )
37   where ind_cols.index_name is null
```

Обсуждение

Каждый производитель использует собственные механизмы блокировки при изменении строк. В случаях, когда отношение родитель-потомок организовано посредством внешнего ключа, индексация дочернего(их) столбца(ов) может сократить время или количество блокировок (подробнее смотрите в специальной документации своей СУБД). В остальных случаях дочерняя таблица обычно объединяется с родительской по столбцу внешнего ключа, так что индекс может способствовать повышению производительности и в этом сценарии.

Использование SQL для генерирования SQL

Задача

Требуется создать динамические выражения SQL, возможно, для автоматизации задач по обслуживанию базы данных. В частности, необходимо реализовать три задачи: посчитать число строк в таблицах, отменить ограничения внешнего ключа, определенные в таблицах, и сгенерировать из данных, хранящихся в таблицах, сценарии вставки.

Решение

Основная идея состоит в использовании строки таблицы для построения выражений SQL. При этом значения, которыми должны быть заполнены выражения (например, имя объекта, к которому обращена команда), берутся из данных таблицы. Помните, запросы только генерируют выражения, которые вы затем должны выполнять посредством сценария, вручную или любым другим способом. Ниже приведены примеры запросов, которые работают в системе Oracle. Для других СУБД техника точно такая же, отличие будет лишь в именах таблиц словаря данных и в форматировании данных. Показанный результат выполнения приведенных ниже запросов является фрагментом строк, которые были возвращены экземпляром Oracle на моем компьютере. Вы, конечно же, получите другие результирующие множества.

```
/* генерируем SQL для подсчета всех строк всех таблиц */

 select 'select count(*) from '||table_name||';' cnts
   from user_tables;

CNTS
-------------------------------------
select count(*) from ANT;
select count(*) from BONUS;
select count(*) from DEMO1;
select count(*) from DEMO2;
select count(*) from DEPT;
select count(*) from DUMMY;
select count(*) from EMP;
```

```
select count(*) from EMP_SALES;
select count(*) from EMP_SCORE;
select count(*) from PROFESSOR;
select count(*) from T;
select count(*) from T1;
select count(*) from T2;
select count(*) from T3;
select count(*) from TEACH;
select count(*) from TEST;
select count(*) from TRX_LOG;
select count(*) from X;

/* деактивируем внешние ключи всех таблиц */

select 'alter table '||table_name||
       ' disable constraint '||constraint_name||';' cons
   from user_constraints
  where constraint_type = 'R';

CONS
------------------------------------------------------
alter table ANT disable constraint ANT_FK;
alter table BONUS disable constraint BONUS_FK;
alter table DEMO1 disable constraint DEMO1_FK;
alter table DEMO2 disable constraint DEMO2_FK;
alter table DEPT disable constraint DEPT_FK;
alter table DUMMY disable constraint DUMMY_FK;
alter table EMP disable constraint EMP_FK;
alter table EMP_SALES disable constraint EMP_SALES_FK;
alter table EMP_SCORE disable constraint EMP_SCORE_FK;
alter table PROFESSOR disable constraint PROFESSOR_FK;

 /* генерируем сценарий вставки из некоторых столбцов таблицы EMP */

select 'insert into emp(empno,ename,hiredate) '||chr(10)||
       'values( '||empno||','||''''||ename
       ||''',to_date('||''''||hiredate||''') );' inserts
  from emp
 where deptno = 10;

INSERTS
-----------------------------------------------------------------
insert into emp(empno,ename,hiredate)
values( 7782,'CLARK',to_date('09-JUN-1981 00:00:00') );

insert into emp(empno,ename,hiredate)
values( 7839,'KING',to_date('17-NOV-1981 00:00:00') );

insert into emp(empno,ename,hiredate)
values( 7934,'MILLER',to_date('23-JAN-1982 00:00:00') );
```

Обсуждение

Генерировать SQL с помощью SQL особенно удобно для создания переносимых сценариев, таких, которые можно использовать при тестировании в разных средах. Кроме того, как видно из приведенных выше примеров, это пригодится и для обслуживания пакетов данных, и для облегчения поиска информации о нескольких объектах сразу. Создание SQL-выражений в SQL – чрезвычайно простая операция, и чем чаще вы ее выполняете, тем проще она становится. Представленные примеры должны стать хорошей базой для начала построения собственных «динамических» SQL-сценариев, потому что, откровенно говоря, здесь у вас вряд ли возникнут трудности. Проработайте примеры и все станет понятным.

Описание представлений словаря данных в базе данных Oracle

Задача

Вы используете Oracle и не можете запомнить ни доступные представления словарей данных, ни описания их столбцов. Что еще хуже, у вас под рукой нет документации производителя.

Решение

Есть специальный рецепт для Oracle. Oracle не только поддерживает набор представлений словарей данных, но даже представления словарей данных для документирования представлений словарей данных. Все так замечательно взаимосвязано.

Сделайте запрос к представлению DICTIONARY (словарь), чтобы получить список представлений словарей данных и их назначений:

```
select table_name, comments
  from dictionary
 order by table_name;

TABLE_NAME                     COMMENTS
------------------------------ ------------------------------------------
ALL_ALL_TABLES                 Description of all object and relational
                               tables accessible to the user

ALL_APPLY                      Details about each apply process that
                               dequeues from the queue visible to the
                               current user
...
```

Запросите DICT_COLUMNS, чтобы описать столбцы заданного представления словаря данных:

```
select column_name, comments
     from dict_columns
```

```
 where table_name = 'ALL_TAB_COLUMNS';

COLUMN_NAME                    COMMENTS
------------------------------ ----------------------------------------
OWNER
TABLE_NAME                     Table, view or cluster name
COLUMN_NAME                    Column name
DATA_TYPE                      Datatype of the column
DATA_TYPE_MOD                  Datatype modifier of the column
DATA_TYPE_OWNER                Owner of the datatype of the column
DATA_LENGTH                    Length of the column in bytes
DATA_PRECISION                 Length: decimal digits (NUMBER) or binary
                               digits (FLOAT)
```

Обсуждение

В те времена, когда набор документации Oracle не был так свободно доступен в сети, было чрезвычайно удобно иметь возможность доступа к представлениям DICTIONARY и DICT_COLUMNS. Зная только эти два представления, можно было получить информацию обо всех остальных представлениях и, следовательно, увидеть всю базу данных.

Даже сегодня информация о DICTIONARY и DICT_COLUMNS не будет лишней. Если нет достаточной уверенности в том, какое представление описывает данный тип объектов, можно создать групповой запрос. Например, чтобы понять, какие представления описывают таблицы схемы, пишем следующий запрос:

```
select table_name, comments
  from dictionary
 where table_name LIKE '%TABLE%'
 order by table_name;
```

Такой запрос возвращает все имена представлений словарей данных, в которые входит слово «TABLE» (таблица). В этом подходе используется преимущество довольно последовательного соглашения о присваивании имен представлениям словарей данных Oracle. Имена всех представлений, описывающих таблицы, вероятнее всего, будут включать слово «TABLE». (Иногда, как в случае с ALL_TAB_COLUMNS, вместо TABLE используется сокращение TAB.)

6

Работа со строками

Данная глава посвящена работе со строками в SQL. Не надо забывать, что SQL разработан не для того, чтобы выполнять сложные операции над строками, поэтому подчас решение этих задач в SQL может показаться (и покажется) весьма громоздким и утомительным. Несмотря на ограниченность SQL в этом вопросе, есть несколько очень полезных встроенных функций, предоставляемых разными СУБД, и я попытался творчески подойти к их использованию. Данная глава, в частности, прекрасно иллюстрирует послание, которое я пытался передать во введении: SQL бывает хорош, плох и просто кошмарен. Надеюсь, из этой главы вы вынесете лучшее понимание того, что возможно и невозможно сделать со строками в SQL. Во многих случаях простота синтаксического разбора и преобразования строк просто поразительна, тогда как иногда SQL-запросы, которые приходится создавать для выполнения той или иной задачи, приводят в ужас.

Особенно важен первый рецепт, поскольку он используется в нескольких последующих решениях. Возможность посимвольного обхода строки пригодится во многих случаях. К сожалению, в SQL реализовать это не просто. Поскольку в SQL нет циклов (за исключением Oracle оператора MODEL), для обхода строки цикл приходится имитировать. Я называю эту операцию «проход строки» или «проход по строке», ее техника описывается в самом первом рецепте. Это фундаментальная операция синтаксического разбора строк при использовании SQL. Она упоминается и используется практически во всех рецептах данной главы. Настоятельно рекомендую досконально разобраться в этом вопросе.

Проход строки

Задача

Требуется обойти строку и возвратить каждый ее символ в отдельной строке таблицы, но в SQL нет операции цикла.[1] Например, значение столбца ENAME (KING) таблицы EMP необходимо представить в виде четырех строк таблицы по одному символу слова «KING» в каждой.

Решение

Чтобы получить количество строк, необходимое для вывода каждого символа в отдельной строке, используем декартово произведение. Затем с помощью встроенной функции СУБД для синтаксического разбора извлекаем интересующие нас символы (пользователи SQL Server будут работать с функцией SUBSTRING, а не SUBSTR[2]):

```
1 select substr(e.ename,iter.pos,1) as C
2   from (select ename from emp where ename = 'KING') e,
3        (select id as pos from t10) iter
4  where iter.pos <= length(e.ename)

C
-
K
I
N
G
```

Обсуждение

Основная идея перебора символов строки – осуществить объединение с таблицей, имеющей достаточно строк, чтобы обеспечить необходимое число итераций. В данном примере используется таблица T10, содержащая 10 строк (и один столбец, ID, в котором располагаются значения от 1 до 10). В результате этого запроса может быть возвращено максимум 10 строк.

В следующем примере показано декартово произведение представлений E и ITER (т. е. декартово произведение заданного имени и 10 строк таблицы T10) без синтаксического разбора ENAME:

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
```

---

[1] Большинство СУБД поддерживают процедурные расширения, в которых есть циклы. – *Примеч. науч. ред.*

[2] Кроме того, в SQL Server используется функция LEN вместо LENGTH. – *Примеч. науч. ред.*

```
ENAME             POS
---------- ----------
KING                1
KING                2
KING                3
KING                4
KING                5
KING                6
KING                7
KING                8
KING                9
KING               10
```

Кардинальность вложенного запроса E равна 1, а вложенного запроса ITER – 10. Таким образом, в результате декартова произведения получаем 10 строк. Формирование такого произведения – первый шаг при имитации цикла в SQL.

Обычно таблицу T10 называют сводной таблицей.

Для выхода из цикла после возвращения четырех строк в решении используется предикат WHERE. Чтобы результирующее множество содержало столько же строк, сколько символов в имени, WHERE накладывает условие ITER.POS <= LENGTH(E.ENAME):

```
select ename, iter.pos
  from (select ename from emp where ename = 'KING') e,
       (select id as pos from t10) iter
 where iter.pos <= length(e.ename)
```

```
ENAME             POS
---------- ----------
KING                1
KING                2
KING                3
KING                4
```

Теперь, когда количество строк соответствует количеству символов E.ENAME, ITER.POS можно использовать как параметр SUBSTR, что позволит перебирать символы строки имени. Каждое последующее значение ITER.POS больше предыдущего на единицу, таким образом, можно сделать, чтобы каждая последующая строка возвращала очередной символ E.ENAME. Таков принцип работы примера решения.

В строке может выводиться разное количество символов в зависимости от поставленной задачи. Следующий запрос является примером обхода E.ENAME и вывода разных частей (более одного символа) строки:

```
select substr(e.ename,iter.pos) a,
       substr(e.ename,length(e.ename)-iter.pos+1) b
```

```
  from (select ename from emp where ename = 'KING') e,
       (select id pos from t10) iter
 where iter.pos <= length(e.ename)

A          B
---------- ------
KING       G
ING        NG
NG         ING
G          KING
```

Чаще всего в рецептах данной главы используются сценарии обхода всей строки и вывода каждого ее символа в отдельной строке таблицы или обхода строки таким образом, чтобы число сгенерированных строк таблицы отражало количество символов или разделителей в строке.

Как вставить кавычки в строковые литералы

Задача

Требуется вставить кавычки в строковые литералы. Хотелось бы с помощью SQL получить результат, подобный приведенному ниже:

```
QMARKS
--------------
g'day mate
beavers' teeth
'
```

Решение

Следующие три выражения SELECT представляют разные способы создания кавычек: как в середине строки, так и отдельных:

```
1 select 'g''day mate' qmarks from t1 union all
2 select 'beavers'' teeth'    from t1 union all
3 select ''''                 from t1
```

Обсуждение

Работая с кавычками, обычно удобно рассматривать их как скобки. Если есть открывающая скобка, всегда должна быть соответствующая ей закрывающая скобка. То же самое и с кавычками. Необходимо помнить, что число кавычек в любой строке должно быть четным. Чтобы вставить в строку одну кавычку, мы должны использовать две кавычки:

```
select 'apples core', 'apple''s core',
       case when '' is null then 0 else 1 end
  from t1

'APPLESCORE  'APPLE''SCOR  CASEWHEN''ISNULLTHENOELSE1END
-----------  ------------  ------------------------------
apples core  apple's core                               0
```

Далее представлено решение, разобранное по элементам. Имеются две внешние кавычки, определяющие строковый литерал, и в строковом литерале есть еще две кавычки, вместе представляющие всего одну кавычку строки, которую, фактически, мы и получаем:

```
select '''' as quote from t1

 Q
 -
 '
```

При работе с кавычками необходимо помнить, что строковый литерал, состоящий только из двух кавычек, без символов между ними, соответствует пустой строке.

Как подсчитать, сколько раз символ встречается в строке

Задача

Требуется подсчитать, сколько раз символ или подстрока встречаются в заданной строке. Рассмотрим следующую строку:

```
10,CLARK,MANAGER
```

Необходимо определить количество запятых в строке.

Решение

Чтобы определить количество запятых в строке, вычтем ее длину без запятых из исходной длины. Каждая СУБД предоставляет функции для получения длины строки и удаления из нее символов. В большинстве случаев это функции LENGTH (длина) и REPLACE (заменить) соответственно (пользователям SQL Server вместо LENGTH предлагается встроенная функция LEN):

```
1 select (length('10,CLARK,MANAGER')-
2        length(replace('10,CLARK,MANAGER',',','')))/length(',')
3        as cnt
4   from t1
```

Обсуждение

Мы получили решение, используя простое вычитание. В результате вызова функции LENGTH в строке 1 получаем исходный размер строки, а вызов LENGTH в строке 2 возвращает размер строки без запятых, которые были удалены функцией REPLACE.

Вычитая одну длину из другой, получаем разницу в символах, которая соответствует количеству запятых в строке. Последней операцией является деление на длину строки поиска. Эта операция необходима, если длина искомой строки более 1 символа. В следующем примере

подсчет количества «LL» в строке «HELLO HELLO» без деления дает неверный результат:

```
select
       (length('HELLO HELLO')-
       length(replace('HELLO HELLO','LL','')))/length('LL')
       as correct_cnt,
       (length('HELLO HELLO')-
       length(replace('HELLO HELLO','LL',''))) as incorrect_cnt
  from t1

CORRECT_CNT INCORRECT_CNT
----------- -------------
          2             4
```

Удаление из строки ненужных символов

Задача

Требуется удалить из данных определенные символы. Рассмотрим следующее результирующее множество:

```
ENAME             SAL
---------- ----------
SMITH              800
ALLEN             1600
WARD              1250
JONES             2975
MARTIN            1250
BLAKE             2850
CLARK             2450
SCOTT             3000
KING              5000
TURNER            1500
ADAMS             1100
JAMES              950
FORD              3000
MILLER            1300
```

Необходимо удалить все нули и гласные, чтобы получить значения, представленные в столбцах STRIPPED1 и STRIPPED2:

```
ENAME      STRIPPED1         SAL STRIPPED2
---------- ---------- ---------- ---------
SMITH      SMTH              800 8
ALLEN      LLN              1600 16
WARD       WRD              1250 125
JONES      JNS              2975 2975
MARTIN     MRTN             1250 125
BLAKE      BLK              2850 285
CLARK      CLRK             2450 245
SCOTT      SCTT             3000 3
KING       KNG              5000 5
TURNER     TRNR             1500 15
```

```
ADAMS      DMS             1100 11
JAMES      JMS              950 95
FORD       FRD             3000 3
MILLER     MLLR            1300 13
```

Решение

Все СУБД предлагают функции для удаления ненужных символов из строки. Самыми полезными при решении данной задачи являются функции REPLACE и TRANSLATE.

DB2

Для удаления ненужных символов и строк используйте встроенные функции TRANSLATE и REPLACE:

```
1 select ename,
2        replace(translate(ename,'aaaaa','AEIOU'),'a','') stripped1,
3        sal,
4        replace(cast(sal as char(4)),'0','') stripped2
5   from emp
```

MySQL и SQL Server

MySQL и SQL Server не предлагают функции TRANSLATE, поэтому приходится несколько раз вызывать REPLACE:

```
 1  select ename,
 2         replace(
 3         replace(
 4         replace(
 5         replace(
 6         replace(ename,'A',''),'E',''),'I',''),'O',''),'U','')
 7         as stripped1,
 8         sal,
 9         replace(sal,0,'') stripped2
10    from emp
```

Oracle и PostgreSQL

Для удаления ненужных символов и строк используйте встроенные функции TRANSLATE и REPLACE:

```
1  select ename,
2         replace(translate(ename,'AEIOU','aaaaa'),'a')
3         as stripped1,
4         sal,
5         replace(sal,0,'') as stripped2
6    from emp
```

Обсуждение

Встроенная функция REPLACE удаляет из строки все нули. Для удаления гласных применяется функция TRANSLATE, которая заменяет

их определенным символом (я использовал «а», вы можете указать любой другой символ), а REPLACE затем удаляет все экземпляры этого символа.

Разделение числовых и символьных данных

Задача

Скажем, в одном столбце (к несчастью) хранятся и числовые, и символьные данные. Требуется отделить символьные данные от числовых. Рассмотрим следующее результирующее множество:

```
DATA
---------------
SMITH800
ALLEN1600
WARD1250
JONES2975
MARTIN1250
BLAKE2850
CLARK2450
SCOTT3000
KING5000
TURNER1500
ADAMS1100
JAMES950
FORD3000
MILLER1300
```

Хотелось бы получить такой результат:

```
ENAME             SAL
---------- ----------
SMITH              800
ALLEN             1600
WARD              1250
JONES             2975
MARTIN            1250
BLAKE             2850
CLARK             2450
SCOTT             3000
KING              5000
TURNER            1500
ADAMS             1100
JAMES              950
FORD              3000
MILLER            1300
```

Решение

Чтобы отделить символы от числовых данных, используйте встроенные функции TRANSLATE и REPLACE. Как и в остальных рецептах данной главы, фокус в том, чтобы с помощью функции TRANSLATE

превратить разные символы в один. Тогда вместо выбора множества чисел или символов задача сводится к поиску одного символа, представляющего все числа, или одного символа, представляющего все символы.

DB2

Чтобы выбрать и отделить числовые данные от символьных, используйте функции TRANSLATE и REPLACE:

```
 1 select replace(
 2      translate(data,'0000000000','0123456789'),'0','') ename,
 3         cast(
 4      replace(
 5    translate(lower(data),repeat('z',26),
 6           'abcdefghijklmnopqrstuvwxyz'),'z','') as integer) sal
 7    from (
 8  select ename||cast(sal as char(4)) data
 9    from emp
10         ) x
```

Oracle

Чтобы выбрать и отделить числовые данные от символьных, используйте функции TRANSLATE и REPLACE:

```
 1  select replace(
 2       translate(data,'0123456789','0000000000'),'0') ename,
 3       to_number(
 5         replace(
 6         translate(lower(data),
 7                   'abcdefghijklmnopqrstuvwxyz',
 8                    rpad('z',26,'z')),'z')) sal
 9    from (
10  select ename||sal data
11    from emp
12         )
```

PostgreSQL

Чтобы выбрать и отделить числовые данные от символьных, используйте функции TRANSLATE и REPLACE:

```
 1 select replace(
 2      translate(data,'0123456789','0000000000'),'0','') as ename,
 3           cast(
 4        replace(
 5      translate(lower(data),
 6                'abcdefghijklmnopqrstuvwxyz',
 7               rpad('z',26,'z')),'z','') as integer) as sal
 8   from (
 9 select ename||sal as data
10   from emp
11        ) x
```

Обсуждение

Для каждой СУБД синтаксис немного различается, но техника одна. Для обсуждения возьмем решение Oracle. Ключ к решению задачи – выбрать числовые и символьные данные. Для этого подойдут функции TRANSLATE и REPLACE. Чтобы извлечь числовые данные, сначала с помощью TRANSLATE выделим все символьные данные:

```
select data,
       translate(lower(data),
                 'abcdefghijklmnopqrstuvwxyz',
                  rpad('z',26,'z')) sal
  from (select ename||sal data from emp)
```

```
DATA                 SAL
-------------------- --------------------
SMITH800             zzzzz800
ALLEN1600            zzzzz1600
WARD1250             zzzz1250
JONES2975            zzzzz2975
MARTIN1250           zzzzzz1250
BLAKE2850            zzzzz2850
CLARK2450            zzzzz2450
SCOTT3000            zzzzz3000
KING5000             zzzz5000
TURNER1500           zzzzzz1500
ADAMS1100            zzzzz1100
JAMES950             zzzzz950
FORD3000             zzzz3000
MILLER1300           zzzzzz1300
```

Функция TRANSLATE заменяет все нечисловые символы строчной Z. Следующий шаг – с помощью REPLACE удалить все экземпляры строчной Z из всех записей, оставляя только числа, которые затем могут быть приведены к числовому типу данных:

```
select data,
       to_number(
         replace(
       translate(lower(data),
                 'abcdefghijklmnopqrstuvwxyz',
                  rpad('z',26,'z')),'z')) sal
  from (select ename||sal data from emp)
```

```
DATA                       SAL
-------------------- ----------
SMITH800                  800
ALLEN1600                1600
WARD1250                 1250
JONES2975                2975
MARTIN1250               1250
BLAKE2850                2850
```

```
CLARK2450                2450
SCOTT3000                3000
KING5000                 5000
TURNER1500               1500
ADAMS1100                1100
JAMES950                  950
FORD3000                 3000
MILLER1300               1300
```

Чтобы извлечь нечисловые символы, с помощью TRANSLATE выделим числовые символы:

```
select data,
       translate(data,'0123456789','0000000000') ename
  from (select ename||sal data from emp)
```

```
DATA                 ENAME
-------------------- ----------
SMITH800             SMITH000
ALLEN1600            ALLEN0000
WARD1250             WARD0000
JONES2975            JONES0000
MARTIN1250           MARTIN0000
BLAKE2850            BLAKE0000
CLARK2450            CLARK0000
SCOTT3000            SCOTT0000
KING5000             KING0000
TURNER1500           TURNER0000
ADAMS1100            ADAMS0000
JAMES950             JAMES000
FORD3000             FORD0000
MILLER1300           MILLER0000
```

Функция TRANSLATE заменяет все числовые символы нулями. Следующий шаг – с помощью REPLACE удалить все нули из всех записей, оставляя только нечисловые символы:

```
select data,
       replace(translate(data,'0123456789','0000000000'),'0') ename
  from (select ename||sal data from emp)
```

```
DATA                 ENAME
-------------------- --------
SMITH800             SMITH
ALLEN1600            ALLEN
WARD1250             WARD
JONES2975            JONES
MARTIN1250           MARTIN
BLAKE2850            BLAKE
CLARK2450            CLARK
SCOTT3000            SCOTT
KING5000             KING
TURNER1500           TURNER
```

```
ADAMS1100           ADAMS
JAMES950            JAMES
FORD3000            FORD
MILLER1300          MILLER
```

Объедините обе техники и получите решение.

Как определить, содержит ли строка только буквенно-цифровые данные

Задача

Требуется извлечь из таблицы строки, в которых в интересующем столбце содержатся только числа или буквы. Рассмотрим следующее представление V (в SQL для конкатенации используется оператор «+», а не «||»):

```
create view V as
select ename as data
  from emp
 where deptno=10
 union all
select ename||', $'|| cast(sal as char(4)) ||'.00' as data
  from emp
 where deptno=20
 union all
select ename|| cast(deptno as char(4)) as data
  from emp
 where deptno=30
```

Представление V представляет вашу таблицу и возвращает следующее:

```
DATA
--------------------
CLARK
KING
MILLER
SMITH, $800.00
JONES, $2975.00
SCOTT, $3000.00
ADAMS, $1100.00
FORD, $3000.00
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30
```

Однако из всех данных представления требуется получить только такие строки:

```
DATA
-------------
CLARK
KING
MILLER
ALLEN30
WARD30
MARTIN30
BLAKE30
TURNER30
JAMES30
```

Одним словом, стоит задача выбрать строки, в которых содержатся только буквы или цифры.

Решение

На первый взгляд, кажется логичным решать задачу через поиск всех символов, не являющихся буквами или числами. Однако, напротив, намного проще сделать прямо противоположное: найти все буквенно-цифровые символы. Для этого всех их можно заменить одним символом и рассматривать как единое целое. Когда есть копия строки, в которой все буквенно-цифровые символы представлены каким-то одним символом по вашему выбору, очень просто отделить буквенно-цифровые символы от всех остальных.

DB2

С помощью функции TRANSLATE замените все буквенно-цифровые символы каким-то одним символом. Затем выберите строки, в которых, кроме заданного, содержатся и другие символы. Пользователи DB2 обязательно должны вызвать функцию CAST в представлении V; в противном случае представление не будет создано из-за ошибок при преобразовании типов. Приведение к типу CHAR требует особой аккуратности, поскольку это тип фиксированной длины (с дополнением пробелами):

```
1  select data
2    from V
3   where translate(lower(data),
4                   repeat('a',36),
5                   '0123456789abcdefghijklmnopqrstuvwxyz') =
6                   repeat('a',length(data))
```

MySQL

В MySQL синтаксис для получения представления V немного другой:

```
create view V as
select ename as data
  from emp
 where deptno=10
```

```
 union all
select concat(ename,', $',sal,'.00') as data
  from emp
 where deptno=20
 union all
select concat(ename,deptno) as data
  from emp
 where deptno=30
```

Для выбора строк, содержащих не только буквенно-цифровые данные, используем регулярное выражение:

```
1 select data
2   from V
3  where data regexp '[^0-9a-zA-Z]' = 0
```

Oracle и PostgreSQL

С помощью функции TRANSLATE замените все буквенно-цифровые символы каким-то одним символом. Затем выберите строки, в которых, кроме заданного, содержатся и другие символы. В Oracle и PostgreSQL вызов функции CAST в представлении V не нужен. Будьте особенно аккуратными при приведении к CHAR, поскольку это тип фиксированной длины (с дополнением пробелами). Если необходимо привести к символьному типу данных, лучше использовать типы VARCHAR или VARCHAR2:

```
1  select data
2    from V
3   where translate(lower(data),
4                   '0123456789abcdefghijklmnopqrstuvwxyz',
5                   rpad('a',36,'a')) = rpad('a',length(data),'a')
```

SQL Server

Поскольку SQL Server не поддерживает функции TRANSLATE, приходится обходить каждую строку и выбирать из строк те, в которых содержатся символы, не являющиеся буквенно-цифровыми значениями. Это можно сделать разными способами, но в приведенном ниже решении используется анализ ASCII-значения:

```
 1  select data
 2    from (
 3  select v.data, iter.pos,
 4         substring(v.data,iter.pos,1) c,
 5         ascii(substring(v.data,iter.pos,1)) val
 6    from v,
 7         ( select id as pos from t100 ) iter
 8   where iter.pos <= len(v.data)
 9         ) x
10   group by data
11   having min(val) between 48 and 122
```

Обсуждение

Суть данных решений – в возможности ссылаться одновременно на множество символов. Используя функцию TRANSLATE, можно без труда манипулировать всеми числами или всеми символами без необходимости «перебора» и проверки каждого символа, один за другим.

DB2, Oracle и PostgreSQL

Лишь 9 из 14 строк представления V являются буквенно-цифровыми. Чтобы выбрать только их, просто используем функцию TRANSLATE. В данном примере TRANSLATE заменяет символы 0–9 и a–z символом «a». После выполнения замены преобразованные строки сравниваются со строкой символов «a» той же длины (что и рассматриваемая строка). Если длины строк одинаковые, значит, все символы проверяемой строки являются буквенно-цифровыми и других символов в ней нет.

Используя функцию TRANSLATE (применяется синтаксис Oracle):

```
where translate(lower(data),
                '0123456789abcdefghijklmnopqrstuvwxyz',
                rpad('a',36,'a'))
```

заменяем все числа и буквы особым символом (я выбрал «a»). После выполнения такой замены все строки, которые в самом деле являются буквенно-цифровыми, могут быть идентифицированы как строки, образованные одним символом (в данном случае «a»). Это можно увидеть, выполнив отдельно функцию TRANSLATE:

```
select data, translate(lower(data),
                 '0123456789abcdefghijklmnopqrstuvwxyz',
                 rpad('a',36,'a'))
  from V

DATA                 TRANSLATE(LOWER(DATA)
-------------------- --------------------
CLARK                aaaaa
...
SMITH, $800.00       aaaaa, $aaa.aa
...
ALLEN30              aaaaaaa
...
```

Буквенно-цифровые значения заменены, но длины строк остались неизменными. Поэтому выбрать необходимо те строки, для которых вызов TRANSLATE возвращает строку, состоящую только из символов «a». Выбор строк осуществляется через сравнение длины исходной строки с длиной соответствующей ей строки символов «a»:

```
select data, translate(lower(data),
                 '0123456789abcdefghijklmnopqrstuvwxyz',
                 rpad('a',36,'a')) translated,
        rpad('a',length(data),'a') fixed
```

```
  from V
DATA                 TRANSLATED           FIXED
-------------------- -------------------- ----------------
CLARK                aaaaa                aaaaa
...
SMITH, $800.00       aaaaa, $aaa.aa       aaaaaaaaaaaaaa
...
ALLEN30              aaaaaaa              aaaaaaa
...
```

Последний шаг – выбрать только те строки, для которых значение столбца TRANSLATED равно значению столбца FIXED.

MySQL

Выражение предиката WHERE:

```
where data regexp '[^0-9a-zA-Z]' = 0
```

обусловливает выбор строк, содержащих только числа или символы. Диапазоны значений в квадратных скобках, «0–9a–zA–Z», представляют все возможные числа и буквы. Символ «^» является символом отрицания. Таким образом, выражение можно прочитать как «не числа или не буквы». Возвращаемое значение 1 соответствует выполнению условия, 0 – невыполнению. Итак, все выражение гласит: «возвращение строк, в которых есть какие-то другие символы, кроме чисел и символов, является ошибкой».

SQL Server

Первый шаг – выполнить обход каждой строки, возвращенной представлением V. Каждый символ значения столбца DATA (данные) будет возвращен в отдельной строке таблицы. Столбец C представляет символы значений столбца DATA:

```
+-----------------+------+------+------+
| data            | pos  | c    | val  |
+-----------------+------+------+------+
| ADAMS, $1100.00 |    1 | A    |   65 |
| ADAMS, $1100.00 |    2 | D    |   68 |
| ADAMS, $1100.00 |    3 | A    |   65 |
| ADAMS, $1100.00 |    4 | M    |   77 |
| ADAMS, $1100.00 |    5 | S    |   83 |
| ADAMS, $1100.00 |    6 | ,    |   44 |
| ADAMS, $1100.00 |    7 |      |   32 |
| ADAMS, $1100.00 |    8 | $    |   36 |
| ADAMS, $1100.00 |    9 | 1    |   49 |
| ADAMS, $1100.00 |   10 | 1    |   49 |
| ADAMS, $1100.00 |   11 | 0    |   48 |
| ADAMS, $1100.00 |   12 | 0    |   48 |
| ADAMS, $1100.00 |   13 | .    |   46 |
| ADAMS, $1100.00 |   14 | 0    |   48 |
| ADAMS, $1100.00 |   15 | 0    |   48 |
```

Вложенный запрос Z не только возвращает каждый символ столбца DATA в отдельной строке, но и предоставляет ASCII-значение для каждого символа. Для используемой мной конкретной реализации SQL Server буквенно-цифровым символам соответствует диапазон значений 48–122. Зная это, можно осуществлять группировку по столбцу DATA и отфильтровывать те строки, минимальное ASCII-значение которых не входит в диапазон 48–122.

Извлечение инициалов из имени

Задача

Требуется преобразовать имя в инициалы. Возьмем имя:

```
Stewie Griffin
```

Необходимо получить:

```
S.G.
```

Решение

Важно помнить, что SQL не обеспечивает такой гибкости, как языки программирования C или Python; следовательно, создать универсальное решение для работы с любым форматом имен в SQL нелегко. Приведенные здесь решения предполагают, что имя представлено или именем и фамилией, или именем, отчеством/вторым инициалом и фамилией.

DB2

Для получения инициалов используйте встроенные функции REPLACE, TRANSLATE и REPEAT:

```
1 select replace(
2        replace(
3        translate(replace('Stewie Griffin', '.', ''),
4                  repeat('#',26),
5                  'abcdefghijklmnopqrstuvwxyz'),
6                   '#','' ), ' ','.' )
7                  ||'.'
8   from t1
```

MySQL

Для получения инициалов используйте встроенные функции CONCAT, CONCAT_WS, SUBSTRING и SUBSTRING_ INDEX:

```
1  select case
2           when cnt = 2 then
3             trim(trailing '.' from
4                  concat_ws('.',
5                   substr(substring_index(name,' ',1),1,1),
```

```
 6                  substr(name,
 7                         length(substring_index(name,' ',1))+2,1),
 8                  substr(substring_index(name,' ',-1),1,1),
 9                  '.'))
10          else
11            trim(trailing '.' from
12                 concat_ws('.',
13                  substr(substring_index(name,' ',1),1,1),
14                  substr(substring_index(name,' ',-1),1,1)
15                  ))
16          end as initials
17    from (
18  select name,length(name)-length(replace(name,' ','')) as cnt
19    from (
20  select replace('Stewie Griffin','.','') as name from t1
21         )y
22         )x
```

Oracle и PostgreSQL

Для получения инициалов используйте встроенные функции REPLACE, TRANSLATE и RPAD:

```
1 select replace(
2        replace(
3        translate(replace('Stewie Griffin', '.', ''),
4                  'abcdefghijklmnopqrstuvwxyz',
5                  rpad('#',26,'#') ), '#','' ),' ','.' ) ||'.'
6   from t1
```

SQL Server[1]

На момент написания данной книги SQL Server не поддерживает ни функцию TRANSLATE, ни функцию CONCAT_WS.

Обсуждение

Получить инициалы можно, выбирая из имени заглавные[2] буквы. В следующих разделах подробно описаны решения для всех рассматриваемых СУБД.

---

[1] Решение для SQL Server аналогично решению для MySQL, только вместо функции SUBSTRING_INDEX используется функция CHARINDEX, а для конкатенации символ «+». – *Примеч. науч. ред.*

[2] В решениях для DB2, Oracle, PostgreSQL делается предположение, что первые буквы имен – прописные. Если использовать это же условие для SQL Server, то при помощи функции PATINDEX с шаблоном [^A-Z] можно найти позиции всех заглавных букв и вырезать затем эти буквы, используя функцию SUBSTRING. – *Примеч. науч. ред.*

DB2

Функция REPLACE удалит из имени все точки (чтобы обработать второй инициал), а функция TRANSLATE заменит все строчные буквы символом #.

```
select translate(replace('Stewie Griffin', '.', ''),
                 repeat('#',26),
                 'abcdefghijklmnopqrstuvwxyz')
  from t1
```

```
TRANSLATE('STE
--------------
S##### G######
```

На данном этапе инициалы – это символы, отличные от #. С помощью функции REPLACE удаляются все символы #:

```
select replace(
       translate(replace('Stewie Griffin', '.', ''),
                  repeat('#',26),
                  'abcdefghijklmnopqrstuvwxyz'),'#','')
  from t1
```

```
REP
---
S G
```

Следующий шаг – заменить пробелы точками опять же с помощью функции REPLACE:

```
select replace(
       replace(
       translate(replace('Stewie Griffin', '.', ''),
                 repeat('#',26),
                'abcdefghijklmnopqrstuvwxyz'),'#',''),' ','.') || '.'
  from t1
```

```
REPLA
-----
S.G
```

Завершающий шаг – поставить точку в конце.

Oracle и PostgreSQL

Функция REPLACE удалит все точки из имени (чтобы обработать второй инициал), а функция TRANSLATE заменит все строчные буквы символом #.

```
select translate(replace('Stewie Griffin','.',''),
                 'abcdefghijklmnopqrstuvwxyz',
                 rpad('#',26,'#'))
  from t1
```

```
TRANSLATE('STE
```

```
--------------
S##### G######
```

На данном этапе инициалы – это символы, отличные от #. С помощью функции REPLACE удаляются все символы #:

```
select replace(
       translate(replace('Stewie Griffin','.',''),
                 'abcdefghijklmnopqrstuvwxyz',
                  rpad('#',26,'#')),'#','')
  from t1
```

```
REP
---
S G
```

Следующий шаг – заменить пробелы точками опять же с помощью функции REPLACE:

```
select replace(
       replace(
     translate(replace('Stewie Griffin','.',''),
               'abcdefghijklmnopqrstuvwxyz',
                rpad('#',26,'#') ),'#',''),' ','.') || '.'
  from t1
```

```
REPLA
-----
S.G
```

Завершающий шаг – поставить точку в конце.

MySQL

Для удаления точек из имени используется вложенный запрос Y. Вложенный запрос X находит число пробелов в имени, чтобы определить, сколько раз необходимо вызвать функцию SUBSTR для извлечения инициалов. В результате трехкратного вызова SUBSTRING_INDEX строка имени разбивается на составляющие соответственно местоположению пробелов. Поскольку имя состоит только из имени и фамилии, выполняется код конструкции ELSE выражения CASE:

```
select substr(substring_index(name, ' ',1),1,1) as a,
       substr(substring_index(name,' ',-1),1,1) as b
  from (select 'Stewie Griffin' as name from t1) x
```

```
A B
- -
S G
```

Если в рассматриваемом имени есть отчество или второй инициал, то соответствующий инициал будет возвращен за счет выполнения выражения

```
substr(name,length(substring_index(name, ' ',1))+2,1)
```

которое находит конец имени, перемещается на две позиции на начало отчества или второго инициала, т. е. находит начальную позицию для SUBSTR. Поскольку выбирается только один символ, мы получаем инициал, соответствующий отчеству, или второй инициал. После этого инициалы передаются в функцию CONCAT_WS, которая разделяет их точками:

```
select concat_ws('.',
                 substr(substring_index(name, ' ',1),1,1),
                 substr(substring_index(name,' ',-1),1,1),
                 '.' ) a
  from (select 'Stewie Griffin' as name from t1) x
```

```
A
-----
S.G..
```

Последний шаг – убрать из инициалов лишнюю точку.

Упорядочивание по частям строки

Задача

Требуется упорядочить результирующее множество по подстроке. Возьмем следующие записи:

```
ENAME
----------
SMITH
ALLEN
WARD
JONES
MARTIN
BLAKE
CLARK
SCOTT
KING
TURNER
ADAMS
JAMES
FORD
MILLER
```

Требуется упорядочить строки по *последним* двум символам имени:

```
ENAME
---------
ALLEN
TURNER
MILLER
JONES
JAMES
MARTIN
BLAKE
```

```
ADAMS
KING
WARD
FORD
CLARK
SMITH
SCOTT
```

Решение

Ключ к решению – найти и использовать встроенную функцию СУБД для извлечения подстроки, по которой требуется провести сортировку. Обычно эту операцию выполняет функция SUBSTR.

DB2, Oracle, MySQL и PostgreSQL

Чтобы сортировать строки по определенной части строки, используйте сочетание встроенных функций LENGTH и SUBSTR:

```
1 select ename
2   from emp
3  order by substr(ename,length(ename)-1,2)
```

SQL Server

Чтобы сортировать строки по определенной части строки, используйте функции SUBSTRING и LEN:

```
1 select ename
2   from emp
3  order by substring(ename,len(ename)-1,2)
```

Обсуждение

Применяя выражение SUBSTR в конструкции ORDER BY, в качестве параметра сортировки результирующего множества можно выбирать любую часть строки. Кроме того, вы не ограничены только функцией SUBSTR. Упорядочивать строки можно по результату практически любого выражения.

Упорядочивание по числу в строке

Задача

Требуется упорядочить результирующее множество, используя число, входящее в строку. Рассмотрим следующее представление:

```
create view V as
select e.ename ||' '||
        cast(e.empno as char(4))||' '||
        d.dname as data
  from emp e, dept d
 where e.deptno=d.deptno
```

Это представление возвращает следующие данные:

```
DATA
----------------------------
CLARK   7782 ACCOUNTING
KING    7839 ACCOUNTING
MILLER  7934 ACCOUNTING
SMITH   7369 RESEARCH
JONES   7566 RESEARCH
SCOTT   7788 RESEARCH
ADAMS   7876 RESEARCH
FORD    7902 RESEARCH
ALLEN   7499 SALES
WARD    7521 SALES
MARTIN  7654 SALES
BLAKE   7698 SALES
TURNER  7844 SALES
JAMES   7900 SALES
```

Необходимо упорядочить результаты по номеру служащего, который располагается между именем служащего и названием соответствующего отдела:

```
DATA
----------------------------
SMITH   7369 RESEARCH
ALLEN   7499 SALES
WARD    7521 SALES
JONES   7566 RESEARCH
MARTIN  7654 SALES
BLAKE   7698 SALES
CLARK   7782 ACCOUNTING
SCOTT   7788 RESEARCH
KING    7839 ACCOUNTING
TURNER  7844 SALES
ADAMS   7876 RESEARCH
JAMES   7900 SALES
FORD    7902 RESEARCH
MILLER  7934 ACCOUNTING
```

Решение

В каждом решении используются функции и синтаксис, характерные для той или иной СУБД, но методика (применение встроенных функций REPLACE и TRANSLATE) во всех случаях одна и та же. Суть в том, чтобы с помощью REPLACE и TRANSLATE удалить из строк символы, не являющиеся цифрами, оставив только числовые значения, по которым и провести сортировку.

DB2

Чтобы упорядочить результирующее множество по числовым символам строки, используйте встроенные функции REPLACE и TRANSLATE:

```
1  select data
2    from V
3   order by
4          cast(
5       replace(
6     translate(data,repeat('#',length(data)),
7       replace(
8     translate(data,'##########','0123456789'),
9               '#',''')),'#','') as integer)
```

Oracle

Чтобы упорядочить результирующее множество по числовым символам строки, используйте встроенные функции REPLACE и TRANSLATE:

```
1  select data
2    from V
3   order by
4          to_number(
5            replace(
6          translate(data,
7            replace(
8          translate(data,'0123456789','##########'),
9                   '#'),rpad('#',20,'#')),'#'))
```

PostgreSQL

Чтобы упорядочить результирующее множество по числовым символам строки, используйте встроенные функции REPLACE и TRANSLATE:

```
1 select data
2   from V
3  order by
4         cast(
5      replace(
6    translate(data,
7      replace(
8    translate(data,'0123456789','##########'),
9             '#',''),rpad('#',20,'#')),'#','') as integer)
```

MySQL и SQL Server

На момент написания данной книги ни один из данных производителей не поддерживает функцию TRANSLATE.

Обсуждение

Единственное назначение представления V – предоставление строк для демонстрации решения данного рецепта. Это представление просто объединяет несколько столбцов таблицы EMP. В решениях показано, как сортировать такие объединенные текстовые данные по входящему в них номеру служащего.

Используемый в решениях оператор ORDER BY, возможно, выглядит немного устрашающе, но он прекрасно работает и становится довольно понятным, если разобрать его по частям. Чтобы провести сортировку по числам, входящим в строку, проще всего удалить из строки все символы, не являющиеся числами. Как только нечисловые символы будут удалены, останется лишь привести строку чисел к числовому типу данных и сортировать, как душе угодно. Прежде чем анализировать каждую функцию, важно понять порядок их вызова. Начиная с самого внутреннего вызова, функции TRANSLATE (строка 8 во всех решениях), мы видим, что:

1. Вызывается TRANSLATE (строка 8) и результаты возвращаются в
2. REPLACE (строка 7), результаты ее выполнения возвращаются в
3. TRANSLATE (строка 6), результаты ее выполнения возвращаются в
4. REPLACE (строка 5), результаты ее выполнения возвращаются и наконец-то
5. приводятся к числовому типу данных.

Первый шаг – преобразовать числа в символы, которых нет в строке. Для данного примера я выбрал символ «#» и использовал функцию TRANSLATE, чтобы заменить им все числовые символы. После выполнения следующего запроса, например, можно увидеть слева исходные данные, а справа результаты первой подстановки:

```
select data,
       translate(data,'0123456789','##########') as tmp
  from V

DATA                           TMP
 ------------------------------ ----------------------
 CLARK   7782 ACCOUNTING        CLARK   #### ACCOUNTING
 KING    7839 ACCOUNTING        KING    #### ACCOUNTING
 MILLER  7934 ACCOUNTING        MILLER  #### ACCOUNTING
 SMITH   7369 RESEARCH          SMITH   #### RESEARCH
 JONES   7566 RESEARCH          JONES   #### RESEARCH
 SCOTT   7788 RESEARCH          SCOTT   #### RESEARCH
 ADAMS   7876 RESEARCH          ADAMS   #### RESEARCH
 FORD    7902 RESEARCH          FORD    #### RESEARCH
 ALLEN   7499 SALES             ALLEN   #### SALES
 WARD    7521 SALES             WARD    #### SALES
 MARTIN  7654 SALES             MARTIN  #### SALES
 BLAKE   7698 SALES             BLAKE   #### SALES
 TURNER  7844 SALES             TURNER  #### SALES
 JAMES   7900 SALES             JAMES   #### SALES
```

TRANSLATE находит числа в строках и заменяет каждое символом «#». После этого измененные строки возвращаются в REPLACE (строка 11), которая удаляет все экземпляры «#»:

```
select data,
replace(
```

```
translate(data,'0123456789','##########'),'#') as tmp
  from V
```

```
DATA                           TMP
------------------------------ --------------------
CLARK   7782 ACCOUNTING        CLARK    ACCOUNTING
KING    7839 ACCOUNTING        KING     ACCOUNTING
MILLER  7934 ACCOUNTING        MILLER   ACCOUNTING
SMITH   7369 RESEARCH          SMITH    RESEARCH
JONES   7566 RESEARCH          JONES    RESEARCH
SCOTT   7788 RESEARCH          SCOTT    RESEARCH
ADAMS   7876 RESEARCH          ADAMS    RESEARCH
FORD    7902 RESEARCH          FORD     RESEARCH
ALLEN   7499 SALES             ALLEN    SALES
WARD    7521 SALES             WARD     SALES
MARTIN  7654 SALES             MARTIN   SALES
BLAKE   7698 SALES             BLAKE    SALES
TURNER  7844 SALES             TURNER   SALES
JAMES   7900 SALES             JAMES    SALES
```

Затем строки опять попадают в TRANSLATE, но теперь это вторая (самая внешняя) функция TRANSLATE решения. TRANSLATE ведет поиск исходной строки с символами, соответствующими символам в TMP. Если соответствие найдено, то все эти символы также заменяются символами «#». Это преобразование позволяет рассматривать все нечисловые символы как один символ (поскольку все они заменяются одним и тем же символом):

```
select data, translate(data,
             replace(
             translate(data,'0123456789','##########'),
                       '#'),
                       rpad('#',length(data),'#')) as tmp
  from V
```

```
DATA                           TMP
------------------------------ --------------------------
CLARK   7782 ACCOUNTING        ########7782###########
KING    7839 ACCOUNTING        ########7839###########
MILLER  7934 ACCOUNTING        ########7934###########
SMITH   7369 RESEARCH          ########7369#########
JONES   7566 RESEARCH          ########7566#########
SCOTT   7788 RESEARCH          ########7788#########
ADAMS   7876 RESEARCH          ########7876#########
FORD    7902 RESEARCH          ########7902#########
ALLEN   7499 SALES             ########7499######
WARD    7521 SALES             ########7521######
MARTIN  7654 SALES             ########7654######
BLAKE   7698 SALES             ########7698######
TURNER  7844 SALES             ########7844######
JAMES   7900 SALES             ########7900######
```

Следующий шаг – удалить все символы «#» посредством вызова REPLACE (строка 8), оставив только числа:

```
select data, replace(
             translate(data,
             replace(
           translate(data,'0123456789','##########'),
                     '#'),
                     rpad('#',length(data),'#')),'#') as tmp
  from V
```

```
DATA                           TMP
------------------------------ -----------
CLARK   7782 ACCOUNTING        7782
KING    7839 ACCOUNTING        7839
MILLER  7934 ACCOUNTING        7934
SMITH   7369 RESEARCH          7369
JONES   7566 RESEARCH          7566
SCOTT   7788 RESEARCH          7788
ADAMS   7876 RESEARCH          7876
FORD    7902 RESEARCH          7902
ALLEN   7499 SALES             7499
WARD    7521 SALES             7521
MARTIN  7654 SALES             7654
BLAKE   7698 SALES             7698
TURNER  7844 SALES             7844
JAMES   7900 SALES             7900
```

Наконец, приводим TMP к числовому типу данных (строка 4), используя для этого соответствующую функцию СУБД (часто это функция CAST):

```
select data, to_number(
              replace(
             translate(data,
             replace(
       translate(data,'0123456789','##########'),
                 '#'),
                 rpad('#',length(data),'#')),'#')) as tmp
  from V
```

```
DATA                                TMP
------------------------------ ----------
CLARK   7782 ACCOUNTING             7782
KING    7839 ACCOUNTING             7839
MILLER  7934 ACCOUNTING             7934
SMITH   7369 RESEARCH               7369
JONES   7566 RESEARCH               7566
SCOTT   7788 RESEARCH               7788
ADAMS   7876 RESEARCH               7876
FORD    7902 RESEARCH               7902
ALLEN   7499 SALES                  7499
WARD    7521 SALES                  7521
```

```
MARTIN  7654 SALES                    7654
BLAKE   7698 SALES                    7698
TURNER  7844 SALES                    7844
JAMES   7900 SALES                    7900
```

При создании подобных запросов полезно располагать используемые выражения в списке SELECT. Тогда можно без труда просматривать промежуточные результаты в ходе продвижения к окончательному решению. Поскольку целью данного рецепта является упорядочивание результатов, в итоге необходимо поместить все вызовы функций в оператор ORDER BY:

```
select data
  from V
 order by
        to_number(
          replace(
        translate( data,
          replace(
        translate( data,'0123456789','##########'),
                  '#'),rpad('#',length(data),'#')),'#'))
```

```
DATA
---------------------------
SMITH   7369 RESEARCH
ALLEN   7499 SALES
WARD    7521 SALES
JONES   7566 RESEARCH
MARTIN  7654 SALES
BLAKE   7698 SALES
CLARK   7782 ACCOUNTING
SCOTT   7788 RESEARCH
KING    7839 ACCOUNTING
TURNER  7844 SALES
ADAMS   7876 RESEARCH
JAMES   7900 SALES
FORD    7902 RESEARCH
MILLER  7934 ACCOUNTING
```

И последнее замечание: данные представления располагаются в трех столбцах, и только один из них является числовым. Помните, что при наличии нескольких числовых полей они должны быть объединены в одно перед сортировкой строк.

Создание списка с разделителями из строк таблицы

Задача

Требуется представить строки таблицы в виде списка с разделителями, возможно, через запятую, а не через вертикальную черту, как они обычно отображаются. Стоит задача из исходной таблицы:

```
DEPTNO EMPS
------ ----------
    10 CLARK
    10 KING
    10 MILLER
    20 SMITH
    20 ADAMS
    20 FORD
    20 SCOTT
    20 JONES
    30 ALLEN
    30 BLAKE
    30 MARTIN
    30 JAMES
    30 TURNER
    30 WARD
```

получить следующее результирующее множество:

```
 DEPTNO EMPS
------- -----------------------------------
     10 CLARK,KING,MILLER
     20 SMITH,JONES,SCOTT,ADAMS,FORD
     30 ALLEN,WARD,MARTIN,BLAKE,TURNER,JAMES
```

Решение

Каждая СУБД требует индивидуального подхода к этой проблеме. Ключ к решению – применение встроенных функций СУБД. Знание доступных возможностей позволит, используя функциональность СУБД и творческий подход, решить задачу, которая обычно не ставится в SQL.

DB2

Для построения списка с разделителями используйте рекурсивный WITH:

```
 1  with x (deptno, cnt, list, empno, len)
 2    as (
 3 select deptno, count(*) over (partition by deptno),
 4       cast(ename as varchar(100)), empno, 1
 5  from emp
 6 union all
 7 select x.deptno, x.cnt, x.list ||','|| e.ename, e.empno, x.len+1
 8   from emp e, x
 9  where e.deptno = x.deptno
10    and e.empno > x. empno
11        )
12 select deptno,list
13   from x
14  where len = cnt
```

MySQL

Для построения списка с разделителями используйте встроенную функцию GROUP_CONCAT:

```
select deptno,
       group_concat(ename order by empno separator ',') as emps
  from emp
 group by deptno
```

Oracle

Для построения списка с разделителями используйте встроенную функцию SYS_CONNECT_BY_PATH:

```
 select deptno,
        ltrim(sys_connect_by_path(ename,','),',') emps
   from (
 select deptno,
        ename,
        row_number() over
                (partition by deptno order by empno) rn,
        count(*) over
                (partition by deptno) cnt
   from emp
        )
  where level   = cnt
  start with rn = 1
 connect by prior deptno = deptno and prior rn = rn-1
```

PostgreSQL

PostgreSQL не предлагает стандартной встроенной функции для создания списка с разделителями, поэтому количество значений в списке должно быть известно заранее. Зная предполагаемый максимальный размер списка, можно определить, сколько значений требуется добавить, чтобы создать список, используя стандартные перестановку и конкатенацию:

```
select deptno,
       rtrim(
           max(case when pos=1 then emps else '' end)||
           max(case when pos=2 then emps else '' end)||
           max(case when pos=3 then emps else '' end)||
           max(case when pos=4 then emps else '' end)||
           max(case when pos=5 then emps else '' end)||
           max(case when pos=6 then emps else '' end),','
       ) as emps
  from (
select a.deptno,
       a.ename||',' as emps,
       d.cnt,
       (select count(*) from emp b
```

```
15          where a.deptno=b.deptno and b.empno <= a.empno) as pos
16   from emp a,
17        (select deptno, count(ename) as cnt
18          from emp
19         group by deptno) d
20  where d.deptno=a.deptno
21        ) x
22  group by deptno
23  order by 1
```

SQL Server

Для построения списка с разделителями используйте рекурсивный WITH:

```
 1   with x (deptno, cnt, list, empno, len)
 2     as (
 3 select deptno, count(*) over (partition by deptno),
 4        cast(ename as varchar(100)),
 5        empno,
 6        1
 7   from emp
 9  union all
 9 select x.deptno, x.cnt,
10        cast(x.list + ',' + e.ename as varchar(100)),
11        e.empno, x.len+1
12   from emp e, x
13  where e.deptno = x.deptno
14    and e.empno > x. empno
15        )
16 select deptno,list
17   from x
18  where len = cnt
19  order by 1
```

Обсуждение

Умение создавать списки с разделителями в SQL пригодится, поскольку необходимость в этом возникает довольно часто. Пока что для построения таких списков в SQL каждая СУБД предлагает собственный уникальный метод. Между решениями разных производителей мало общего; техники варьируются от использования рекурсии до иерархических функций, классической перестановки и агрегации.

DB2 и SQL Server

Решения для этих двух баз данных немного различаются в синтаксисе (для DB2 оператор конкатенации – «||», для SQL Server – «+»), но методика одна. Первый запрос в конструкции WITH (верхняя часть UNION ALL) возвращает следующие данные для каждого служащего: отдел, количество служащих в отделе, имя, ID и константу 1 (которая в данный момент никак не используется). Во втором запросе (нижняя

половиная UNION ALL) посредством рекурсии создается список. Чтобы понять, как это происходит, проанализируем некоторые фрагменты решения: сначала третий элемент списка SELECT второго запроса операции UNION ALL:

```
x.list ||','|| e.ename
```

и затем конструкцию WHERE того же запроса:

```
  where e.deptno = x.deptno
    and e.empno > x.empno
```

Сначала в решении проверяется, что все служащие относятся к одному отделу. Затем к каждому служащему, возвращенному верхней частью операции UNION ALL, добавляются имена служащих, которым соответствуют большие значения в столбце EMPNO. Таким образом, гарантируется, что к данному служащему не будет добавлено его собственное имя. Выражение

```
x.len+1
```

увеличивает LEN (изначально равное 1) на единицу при обработке каждого последующего служащего. Если это значение равно количеству служащих в отделе:

```
where len = cnt
```

следовательно, мы обработали данные для всех служащих и завершили создание списка. Это значение является ключевым для запроса, потому что не только сигнализирует о завершении создания списка, но также прекращает дальнейшее рекурсивное выполнение в надлежащий момент.

MySQL

Всю работу выполняет функция GROUP_CONCAT. Она осуществляет конкатенацию значений столбца, переданного в нее, в данном случае – ENAME. Эта функция является агрегатной; таким образом, в запросе необходим оператор GROUP BY.

Oracle

Первый шаг к пониманию запроса Oracle – разложить его на части. Отдельно выполняя вложенный запрос (строки 4–10), формируем результирующее множество, включающее следующие данные для каждого служащего: отдел, имя, ранг в соответствующем отделе, получаемый путем сортировки по возрастанию по столбцу EMPNO, и количество служащих в отделе. Например:

```
select deptno,
       ename,
       row_number() over
                 (partition by deptno order by empno) rn,
```

```
       count(*) over (partition by deptno) cnt
  from emp

DEPTNO ENAME      RN CNT
------ ---------- -- ---
    10 CLARK       1   3
    10 KING        2   3
    10 MILLER      3   3
    20 SMITH       1   5
    20 JONES       2   5
    20 SCOTT       3   5
    20 ADAMS       4   5
    20 FORD        5   5
    30 ALLEN       1   6
    30 WARD        2   6
    30 MARTIN      3   6
    30 BLAKE       4   6
    30 TURNER      5   6
    30 JAMES       6   6
```

Назначение столбца ранга (которому в запросе присвоен псевдоним RN) – обеспечить возможность обхода дерева. Поскольку функция ROW_NUMBER формирует список, начиная с единицы, без повторов или пропусков, для того чтобы обратиться к предыдущей (или родительской) строке, необходимо просто вычесть единицу (из текущего значения). Например, предыдущим числом для 3 является 3 минус 1, что равняется 2. В данном контексте, 2 – родитель 3; наблюдать это можно в строке 12. Кроме того, строки

```
start with rn = 1
connect by prior deptno = deptno
```

определяют корень для каждого DEPTNO как запись, в которой значение RN равно 1, и создают новый список для каждого нового отдела (при каждом обнаружении в столбце RN значения 1).

Здесь важно подробнее остановиться на части ORDER BY функции ROW_NUMBER. Помните, что имена ранжированы по EMPNO и список будет создаваться в соответствующем порядке. Вычисляется количество служащих в отделе (представление под псевдонимом CNT), и это значение обеспечивает, что запрос возвратит список, включающий имена всех служащих отдела. Это необходимо, потому что SYS_CONNECT_BY_PATH создает список путем многократных итераций, и никому не нужен в итоге неполный список. Для иерархических запросов значения псевдостолбца LEVEL (уровень) начинаются с 1 (для запросов, не использующих CONNECT BY, LEVEL равен 0; для версий 10*g* и более поздних LEVEL доступен только в сочетании с CONNECT BY) и увеличиваются на единицу после добавления каждого служащего отдела (для каждого уровня вложенности иерархии). Поэтому, как только LEVEL становится равным CNT, мы знаем, что достигли последнего EMPNO и получим полный список.

Функция SYS_CONNECT_BY_PATH ставит выбранный разделитель (в данном случае запятую) первым элементом списка. Такое поведение может быть нежелательным. В решении данного рецепта вызов функции LTRIM удаляет запятую, стоящую в начале списка.

PostgreSQL

В решении PostgreSQL необходимо заранее знать максимальное число служащих в каждом отделе. Если выполнить отдельно вложенный запрос (строки 11–18), сформируется результирующее множество, включающее (для каждого служащего) отдел, имя с запятой в конце, количество служащих в отделе и количество служащих, значение EMPNO которых меньше, чем у данного служащего:

```
 deptno |  emps   | cnt | pos
--------+---------+-----+-----
     20 | SMITH,  |   5 |   1
     30 | ALLEN,  |   6 |   1
     30 | WARD,   |   6 |   2
     20 | JONES,  |   5 |   2
     30 | MARTIN, |   6 |   3
     30 | BLAKE,  |   6 |   4
     10 | CLARK,  |   3 |   1
     20 | SCOTT,  |   5 |   3
     10 | KING,   |   3 |   2
     30 | TURNER, |   6 |   5
     20 | ADAMS,  |   5 |   4
     30 | JAMES,  |   6 |   6
     20 | FORD,   |   5 |   5
     10 | MILLER, |   3 |   3
```

Скалярный подзапрос, POS (строки 14–15), используется для ранжирования служащих по EMPNO. Например, строка

```
max(case when pos = 1 then ename else '' end)||
```

проверяет значение столбца POS на равенство 1. Выражение CASE возвращает имя служащего, если значение POS равно 1, и NULL в противном случае.

Сначала надо создать запрос для определения, какое максимальное число значений может содержаться в одном списке. Исходя из таблицы EMP, наибольшее число служащих в отделе – шесть, поэтому в любом списке может быть не более шести элементов.

Следующий шаг – создание списка. Это делается посредством применения некоторой условной логики (в форме выражения CASE) к строкам, возвращаемым вложенным запросом. Выражений CASE должно быть столько же, сколько существует значений, подлежащих конкатенации.

Если значение POS равно 1, текущее имя добавляется в список. Второе выражение CASE оценивает, равно ли значение POS двум; если да, то

второе имя присоединяется к первому. Если второго имени нет, к первому имени добавляется еще одна запятая (этот процесс повторяется для каждого значения POS, пока не будет достигнута последняя строка).

Функция MAX необходима, потому что требуется создать по одному списку на отдел (можно также использовать MIN; в данном случае разницы нет, поскольку POS возвращает по одному значению при каждом вычислении выражения CASE). Если используется агрегатная функция, все элементы списка SELECT, не участвующие в агрегации, должны быть заданы в конструкции GROUP BY. Это гарантирует, что мы получим по одной строке на каждый такой элемент списка SELECT.

Обратите внимание, что замыкающие запятые необходимо удалить функцией RTRIM. Количество запятых будет всегда равно максимальному числу значений, которые потенциально могут присутствовать в списке (в данном случае шести).

Преобразование данных с разделителями в список оператора IN со множеством значений

Задача

Имеется список с разделителями, который требуется передать в итератор значений списка оператора IN конструкции WHERE. Рассмотрим следующую строку:

```
7654,7698,7782,7788
```

Хотелось бы применить эту строку в конструкции WHERE, но следующий фрагмент SQL приводит к ошибке, потому что EMPNO – числовой столбец:

```
select ename,sal,deptno
  from emp
 where empno in ( '7654,7698,7782,7788' )
```

Ошибка возникает, поскольку EMPNO является числовым столбцом, а список оператора IN образован одним строковым значением. Необходимо, чтобы эта строка трактовалась как список разделенных запятыми числовых значений.

Решение

На первый взгляд может показаться, что SQL должен самостоятельно интерпретировать список с разделителями как список значений, но это не так. Встречая запятые между кавычками, SQL не может знать, что это признак списка со множеством значений. SQL трактует все, что заключено в кавычки, как одно строковое значение. Необходимо разложить строку на отдельные значения EMPNO. Ключ к этому решению – обойти строку, но не посимвольно, а разбить ее на корректные значения EMPNO.

DB2

Обходя строку, переданную в список оператора IN, можно без труда преобразовать ее в строки таблицы. Функции ROW_NUMBER, LOCATE и SUBSTR будут здесь особенно полезны:

```
 1  select empno,ename,sal,deptno
 2    from emp
 3   where empno in (
 4  select cast(substr(c,2,locate(',',c,2)-2) as integer) empno
 5    from (
 6  select substr(csv.emps,cast(iter.pos as integer)) as c
 7    from (select ','||'7654,7698,7782,7788'||',' emps
 8            from t1) csv,
 9         (select id as pos
10            from t100 ) iter
11   where iter.pos <= length(csv.emps)
12         )  x
13    where length(c) > 1
14      and substr(c,1,1) = ','
15         )  y
```

MySQL

Обходя строку, переданную в список оператора IN, можно без труда преобразовать ее в строки таблицы:

```
 1 select empno, ename, sal, deptno
 2   from emp
 3  where empno in
 4        (
 5 select substring_index(
 6        substring_index(list.vals,',',iter.pos),',',-1) empno
 6   from (select id pos from t10) as iter,
 7        (select '7654,7698,7782,7788' as vals
 8           from t1) list
 9  where iter.pos <=
10        (length(list.vals)-length(replace(list.vals,',',''))+1
11        )  x
```

Oracle

Обходя строку, переданную в список оператора IN, можно без труда преобразовать ее в строки таблицы. Функции ROWNUM, SUBSTR и INSTR будут здесь особенно полезны:

```
1 select empno,ename,sal,deptno
2   from emp
3  where empno in (
4        select to_number(
5                   rtrim(
6                  substr(emps,
7                   instr(emps,',',1,iter.pos)+1,
8                   instr(emps,',',1,iter.pos+1) -
```

```
 9                  instr(emps,',',1,iter.pos)),',')) emps
10          from (select ','||'7654,7698,7782,7788'||',' emps from t1) csv,
11               (select rownum pos from emp) iter
12         where iter.pos <= ((length(csv.emps)-
13                  length(replace(csv.emps,',',''))/length(','))-1
14  )
```

PostgreSQL

Обходя строку, переданную в список оператора IN, можно без труда преобразовать ее в строки таблицы. Функция SPLIT_PART упрощает задачу синтаксического разбора строки на отдельные числовые значения:

```
 1 select ename,sal,deptno
 2   from emp
 3  where empno in (
 4 select cast(empno as integer) as empno
 5   from (
 6 select split_part(list.vals,',',iter.pos) as empno
 7   from (select id as pos from t10) iter,
 8        (select ','||'7654,7698,7782,7788'||',' as vals
 9           from t1) list
10  where iter.pos <=
11        length(list.vals)-length(replace(list.vals,',',''))
12        )  z
13  where length(empno) > 0
14        )  x
```

SQL Server

Обходя строку, переданную в список оператора IN, можно без труда преобразовать ее в строки таблицы. Функции ROW_NUMBER, CHARINDEX и SUBSTRING будут здесь особенно полезны:

```
 1 select empno,ename,sal,deptno
 2   from emp
 3  where empno in (select substring(c,2,charindex(',',c,2)-2) as empno
 4   from (
 5 select substring(csv.emps,iter.pos,len(csv.emps)) as c
 6   from (select ','+'7654,7698,7782,7788'+',' as emps
 7           from t1) csv,
 8        (select id as pos
 9          from t100) iter
10  where iter.pos <= len(csv.emps)
11        )  x
12  where len(c) > 1
13    and substring(c,1,1) = ','
14        )  y
```

Обсуждение

Первый и самый важный шаг в данном решении – обход строки. После этого остается только провести синтаксический разбор строки и раз-

бить ее на отдельные числовые значения, используя предоставленные СУБД функции.

DB2 и SQL Server

Обход строки выполняет вложенный запрос X (строки 6–11). Основная идея решения – «пройти по» строке так, чтобы в каждой следующей строке было на один символ меньше, чем в предыдущей:

```
,7654,7698,7782,7788,
7654,7698,7782,7788,
654,7698,7782,7788,
54,7698,7782,7788,
4,7698,7782,7788,
,7698,7782,7788,
7698,7782,7788,
698,7782,7788,
98,7782,7788,
8,7782,7788,
,7782,7788,
7782,7788,
782,7788,
82,7788,
2,7788,
,7788,
7788,
788,
88,
8,
,
```

Обратите внимание, благодаря тому, что строка заключена в запятые (разделитель), нет необходимости в специальных проверках для определения начала или конца строки.

Следующий шаг – выбрать только те значения, которые войдут в список оператора IN. Это значения, начинающиеся с запятой, за исключением последней строки, в которой запятая является единственным символом. С помощью функции SUBSTR или SUBSTRING находим такие строки и затем для каждой из них сохраняем все символы до следующей запятой. Когда это сделано, приводим строки к числовому типу, чтобы они могли быть помещены в числовой столбец EMPNO (строки 4–14):

```
 EMPNO
------
  7654
  7698
  7782
  7788
```

Последний шаг – использовать результаты в подзапросе для возвращения необходимых строк.

MySQL

Вложенный запрос (строки 5–9) осуществляет обход строки. Выражение в строке 10 определяет количество значений в строке, подсчитывая число запятых (разделителей) и добавляя к нему единицу. Функция SUBSTRING_INDEX (строка 6) возвращает все символы строки до (слева) *n*-ной запятой (разделителя):

```
+--------------------+
| empno              |
+--------------------+
| 7654               |
| 7654,7698          |
| 7654,7698,7782     |
| 7654,7698,7782,7788 |
+--------------------+
```

Затем эти строки передаются в еще одну SUBSTRING_INDEX (строка 5); на этот раз *n*-ным разделителем является первая запятая, поэтому сохранены будут все значения справа от *n*-ного разделителя:

```
+-------+
| empno |
+-------+
| 7654  |
| 7698  |
| 7782  |
| 7788  |
+-------+
```

Последний шаг – вставить результаты в подзапрос.

Oracle

Первый шаг – обход строки:

```
select emps,pos
  from (select ','||'7654,7698,7782,7788'||',' emps
          from t1) csv,
       (select rownum pos from emp) iter
 where iter.pos <=
 ((length(csv.emps)-length(replace(csv.emps,',')))/length(','))-1

EMPS                          POS
--------------------- ----------
,7654,7698,7782,7788,          1
,7654,7698,7782,7788,          2
,7654,7698,7782,7788,          3
,7654,7698,7782,7788,          4
```

Число возвращенных строк представляет количество значений в списке. Значения POS очень важны, потому что они необходимы в запросе для проведения синтаксического разбора строки. Синтаксический разбор строки осуществляется с помощью функций SUBSTR и INSTR.

POS используется для определения местонахождения *n*-ного разделителя в каждой строке. Поскольку строки окружены запятыми, дополнительных проверок для определения начала или конца строки не требуется. Значения передаются в SUBSTR, INSTR (строки 7–9) находит *n*-й и *n*+1-й разделители. Вычитая значение, возвращенное для текущей запятой (местоположение текущей запятой в строке), из значения, возвращенного для следующей запятой (местоположение следующей запятой в строке), можно извлечь все значения строки:

```
select substr(emps,
       instr(emps,',',1,iter.pos)+1,
       instr(emps,',',1,iter.pos+1) -
       instr(emps,',',1,iter.pos)) emps
  from (select ','||'7654,7698,7782,7788'||',' emps
          from t1) csv,
       (select rownum pos from emp) iter
 where iter.pos <=
  ((length(csv.emps)-length(replace(csv.emps,',',')))/length(','))-1

EMPS
-----------
7654,
7698,
7782,
7788,
```

Последний шаг – удалить из каждого значения завершающую запятую, привести значение к числовому типу данных и поместить в подзапрос.

PostgreSQL

Обход строки осуществляет вложенный запрос Z (строки 6–9). Количество возвращаемых строк соответствует количеству значений в строке. Чтобы определить, сколько значений в строке, находим разность между размером строки с разделителями и ее размером без них (строка 9). Синтаксическим разбором строки занимается функция SPLIT_PART. Она ищет значение, располагающееся перед *n*-ным разделителем:

```
select list.vals,
       split_part(list.vals,',',iter.pos) as empno,
       iter.pos
  from (select id as pos from t10) iter,
       (select ','||'7654,7698,7782,7788'||',' as vals
          from t1) list
 where iter.pos <=
       length(list.vals)-length(replace(list.vals,',',''))

         vals          | empno | pos
-----------------------+-------+-----
 ,7654,7698,7782,7788, |       |   1
 ,7654,7698,7782,7788, | 7654  |   2
 ,7654,7698,7782,7788, | 7698  |   3
```

```
,7654,7698,7782,7788, | 7782  |   4
,7654,7698,7782,7788, | 7788  |   5
```

Заключительный шаг – привести значения (EMPNO) к числовому типу и поместить их в подзапрос.

Упорядочение строки в алфавитном порядке

Задача

Требуется расположить символы строк таблицы в алфавитном порядке. Рассмотрим следующее множество:

```
ENAME
----------
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD
```

Желаемый результат:

```
OLD_NAME   NEW_NAME
---------- --------
ADAMS      AADMS
ALLEN      AELLN
BLAKE      ABEKL
CLARK      ACKLR
FORD       DFOR
JAMES      AEJMS
JONES      EJNOS
KING       GIKN
MARTIN     AIMNRT
MILLER     EILLMR
SCOTT      COSTT
SMITH      HIMST
TURNER     ENRRTU
WARD       ADRW
```

Решение

Эта задача – идеальный пример того, почему так исключительно важно понимать СУБД, с которой работаешь, и знать, какая функциональ-

ность доступна. В тех ситуациях, когда используемая СУБД не предоставляет встроенные функции для решения задачи, необходимо применять творческий подход. Сравните решение MySQL с остальными.

DB2

Чтобы расположить строковые значения в алфавитном порядке, необходимо обойти каждую строку и упорядочить ее символы:

```
select ename,
       max(case when pos=1 then c else '' end)||
       max(case when pos=2 then c else '' end)||
       max(case when pos=3 then c else '' end)||
       max(case when pos=4 then c else '' end)||
       max(case when pos=5 then c else '' end)||
       max(case when pos=6 then c else '' end)
  from (
select e.ename,
       cast(substr(e.ename,iter.pos,1) as varchar(100)) c,
       cast(row_number()over(partition by e.ename
                             order by substr(e.ename,iter.pos,1))
            as integer) pos
  from emp e,
       (select cast(row_number()over() as integer) pos
          from emp) iter
 where iter.pos <= length(e.ename)
       ) x
 group by ename
```

MySQL

Ключ к решению в данном случае – функция GROUP_CONCAT, которая не только осуществляет конкатенацию символов, составляющих каждое имя, но и упорядочивает их:

```
select ename, group_concat(c order by c separator '')
  from (
select ename, substr(a.ename,iter.pos,1) c
  from emp a,
       ( select id pos from t10 ) iter
 where iter.pos <= length(a.ename)
       ) x
 group by ename
```

Oracle

Функция SYS_CONNECT_BY_PATH обеспечивает возможность создать список посредством итераций:

```
select old_name, new_name
  from (
select old_name, replace(sys_connect_by_path(c,' '),' ') new_name
  from (
```

```
 5  select e.ename old_name,
 6         row_number() over(partition by e.ename
 7                           order by substr(e.ename,iter.pos,1)) rn,
 8         substr(e.ename,iter.pos,1) c
 9    from emp e,
10         ( select rownum pos from emp ) iter
11   where iter.pos <= length(e.ename)
12   order by 1
13         ) x
14   start with rn = 1
15 connect by prior rn = rn-1 and prior old_name = old_name
16         )
17   where length(old_name) = length(new_name)
```

PostgreSQL

PostgreSQL не предлагает никаких встроенных функций для упрощения сортировки символов в строке, поэтому необходимо не только обойти все строки, но также заранее знать максимально возможную длину имени. В данном решении для удобства чтения кода используется представление V:

```
create or replace view V as
select x.*
  from (
select a.ename,
       substr(a.ename,iter.pos,1) as c
  from emp a,
       (select id as pos from t10) iter
 where iter.pos <= length(a.ename)
 order by 1,2
       ) x
```

Следующее выражение SELECT использует это представление:

```
 1 select ename,
 2        max(case when pos=1 then
 3                 case when cnt=1 then c
 4                      else rpad(c,cast(cnt as integer),c)
 5                 end
 6                 else ''
 7             end)||
 8        max(case when pos=2 then
 9                 case when cnt=1 then c
10                      else rpad(c,cast(cnt as integer),c)
11                 end
12                 else ''
13            end)||
14        max(case when pos=3 then
15                 case when cnt=1 then c
16                      else rpad(c,cast(cnt as integer),c)
17                 end
```

```
18                else ''
19            end)||
20        max(case when pos=4 then
21                case when cnt=1 then c
22                     else rpad(c,cast(cnt as integer),c)
23                end
24                else ''
25            end)||
26        max(case when pos=5 then
27                case when cnt=1 then c
28                     else rpad(c,cast(cnt as integer),c)
29                end
30                else ''
31            end)||
32        max(case when pos=6 then
33                case when cnt=1 then c
34                     else rpad(c,cast(cnt as integer),c)
35                end
36                else ''
37            end)
38   from (
39 select a.ename, a.c,
40        (select count(*)
41           from v b
42          where a.ename=b.ename and a.c=b.c ) as cnt,
43        (select count(*)+1
44           from v b
45          where a.ename=b.ename and b.c<a.c) as pos
46   from v a
47        ) x
48  group by ename
```

SQL Server

Чтобы расположить символы строк в алфавитном порядке, необходимо обойти все строки и упорядочить их символы:

```
 1  select ename,
 2            max(case when pos=1 then c else '' end)+
 3            max(case when pos=2 then c else '' end)+
 4            max(case when pos=3 then c else '' end)+
 5            max(case when pos=4 then c else '' end)+
 6            max(case when pos=5 then c else '' end)+
 7            max(case when pos=6 then c else '' end)
 8       from (
 9     select e.ename,
10           substring(e.ename,iter.pos,1) as c,
11           row_number() over (
12            partition by e.ename
13                order by substring(e.ename,iter.pos,1)) as pos
14      from emp e,
15           (select row_number()over(order by ename) as pos
```

```
16              from emp) iter
17     where iter.pos <= len(e.ename)
18           ) x
19     group by ename
```

Обсуждение

DB2 и SQL Server

Вложенный запрос X возвращает каждый символ каждого имени в отдельной строке. Синтаксический разбор имени осуществляется функцией SUBSTR или SUBSTRING, а функция ROW_NUMBER располагает символы в алфавитном порядке:

```
ENAME    C    POS
-----    -    ---
ADAMS    A    1
ADAMS    A    2
ADAMS    D    3
ADAMS    M    4
ADAMS    S    5
...
```

Чтобы возвратить каждую букву каждого имени в отдельной строке, необходимо выполнить обход строки. Это осуществляет вложенный запрос ITER.

Теперь, когда буквы всех имен выстроены в алфавитном порядке, осталось опять собрать их вместе в одну строку, сохранив этот порядок. Позиция каждой буквы определяется выражениями CASE (строки 2–7). Если найден символ в определенной позиции, то осуществляется его конкатенация с результатом следующего вычисления (следующего выражения CASE). Благодаря использованию агрегатной функции MAX в каждую позицию POS возвращается только один символ, таким образом, для каждого имени возвращается только одна строка. Вычисление CASE продолжается до числа 6 – максимального количества символов в любом имени таблицы EMP.

MySQL

Вложенный запрос X (строки 3–6) возвращает каждый символ каждого имени в отдельной строке. Извлечением символов из имен занимается функция SUBSTR:

```
ENAME    C
-----    -
ADAMS    A
ADAMS    A
ADAMS    D
ADAMS    M
ADAMS    S
...
```

Вложенный запрос ITER используется для обхода строки. Все остальное выполняет функция GROUP_CONCAT. Она не только производит конкатенацию букв, но и располагает их в заданном порядке (в данном случае в алфавитном).

Oracle

Основную работу осуществляет вложенный запрос X (строки 5–11), в котором происходит разложение каждого имени на символы и расположение их в алфавитном порядке. Выполняется это путем обхода строки с последующим их упорядочиванием. Остальная часть запроса просто опять объединяет символы в имена.

Имена в разложенном виде можно увидеть, выполнив отдельно вложенный запрос X:

```
OLD_NAME           RN C
---------- ---------- -
ADAMS               1 A
ADAMS               2 A
ADAMS               3 D
ADAMS               4 M
ADAMS               5 S
...
```

Следующий шаг – взять выстроенные в алфавитном порядке символы и повторно объединить их в имена. Это выполняет функция SYS_CONNECT_BY_PATH, добавляя каждый последующий символ в конец строки, полученной в результате объединения предыдущих:

```
OLD_NAME   NEW_NAME
---------- ----------
ADAMS      A
ADAMS      AA
ADAMS      AAD
ADAMS      AADM
ADAMS      AADMS
...
```

Заключительный шаг – оставить только те строки, длина которых равна длине имен, из которых они были построены.

PostgreSQL

Для удобства чтения в данном решении используется представление V для обхода строки. Функция SUBSTR в описании представления осуществляет посимвольный разбор каждого имени, так что представление возвращает следующее:

```
ENAME    C
-----    -
ADAMS    A
ADAMS    A
ADAMS    D
```

```
 ADAMS    M
 ADAMS    S
...
```

Представление также упорядочивает результаты по столбцу ENAME и по столбцу C в алфавитном порядке. Вложенный запрос X (строки 15–18) возвращает имена и символы, полученные в представлении V, число раз, сколько каждый символ встречается в каждом имени, и его позицию (в алфавитном порядке):

```
 ename | c | cnt | pos
-------+---+-----+-----
 ADAMS | A |   2 |   1
 ADAMS | A |   2 |   1
 ADAMS | D |   1 |   3
 ADAMS | M |   1 |   4
 ADAMS | S |   1 |   5
```

Дополнительные столбцы CNT и POS, возвращенные вложенным запросом X, исключительно важны для решения. POS используется для ранжирования каждого символа, а CNT – для определения того, сколько раз символ встречается в данном имени. Окончательный шаг – определить позицию и реконструировать имя. Как видите, каждое выражение CASE, на самом деле, является двумя выражениями CASE. Сначала выясняется, не встречается ли данный символ в имени более одного раза. Если это так, то возвращается не один символ, а строка, в которой данный символ повторяется столько раз, сколько указано значением столбца CNT. Агрегатная функция MAX обеспечивает вывод по одной строке для каждого имени.

Выявление строк, которые могут быть интерпретированы как числа

Задача

Имеется столбец символьного типа. К несчастью, в строках содержатся и числовые, и символьные данные. Рассмотрим представление V:

```
create view V as
select replace(mixed,' ','') as mixed
  from (
select substr(ename,1,2)||
       cast(deptno as char(4))||
       substr(ename,3,2) as mixed
  from emp
 where deptno = 10
 union all
select cast(empno as char(4)) as mixed
  from emp
 where deptno = 20
 union all
```

```
select ename as mixed
  from emp
 where deptno = 30
       ) x

select * from v
```

```
MIXED
---------------
CL10AR
KI10NG
MI10LL
7369
7566
7788
7876
7902
ALLEN
WARD
MARTIN
BLAKE
TURNER
JAMES
```

Требуется выбрать строки, содержащие только числа или, по крайней мере, одно число. Если в строке есть и числа, и символьные данные, необходимо удалить символы и возвратить только числа. Для приведенных выше данных должно быть получено следующее результирующее множество:

```
   MIXED
--------
      10
      10
      10
    7369
    7566
    7788
    7876
    7902
```

Решение

Для работы со строками и отдельными символами удобно использовать функции REPLACE и TRANSLATE. Суть решения – заменить все числа одним символом, что позволит затем без труда выделить и выбрать каждое число, ссылаясь лишь на один символ.

DB2

Для выбора чисел из строк используются функции TRANSLATE, REPLACE и POSSTR. Функция CAST является обязательной в представ-

лении V, поскольку в противном случае представление не сможет быть создано из-за ошибок несоответствия типов. Функция REPLACE понадобится для удаления лишних пробелов, возникающих в результате приведения к типу фиксированной длины CHAR:

```
 1 select mixed old,
 2        cast(
 3          case
 4          when
 5             replace(
 6           translate(mixed,'9999999999','0123456789'),'9','') = ''
 7          then
 8             mixed
 9          else replace(
10             translate(mixed,
11                repeat('#',length(mixed)),
12              replace(
13               translate(mixed,'9999999999','0123456789'),'9',''')),
14                       '#','')
15          end as integer ) mixed
16   from V
17  where posstr(translate(mixed,'9999999999','0123456789'),'9') > 0
```

MySQL

Синтаксис для MySQL немного отличается. Представление V определяется как:

```
create view V as
select concat(
         substr(ename,1,2),
         replace(cast(deptno as char(4)),' ',''),
         substr(ename,3,2)
       ) as mixed
  from emp
 where deptno = 10
 union all
select replace(cast(empno as char(4)), ' ', '')
  from emp where deptno = 20
 union all
select ename from emp where deptno = 30
```

Поскольку MySQL не поддерживает функцию TRANSLATE, придется выполнять обход каждой строки и последовательно обрабатывать все ее символы.

```
 1 select cast(group_concat(c order by pos separator '') as unsigned)
 2        as MIXED1
 3   from (
 4 select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
 5   from V,
 6        ( select id pos from t10 ) iter
 7  where iter.pos <= length(v.mixed)
```

```
 8    and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
 9        ) y
10  group by mixed
11  order by 1
```

Oracle

С помощью функций TRANSLATE, REPLACE и INSTR выделяем числовые значения в каждой строке. В представлении V вызывать CAST необязательно. Функция REPLACE используется для удаления лишних пробелов, возникающих в результате приведения к типу фиксированной длины CHAR. По желанию можно выполнять явное приведение типа в описании представления, предлагается приводить к типу VARCHAR2:

```
 1 select to_number (
 2        case
 3        when
 4           replace(translate(mixed,'0123456789','9999999999'),'9')
 5          is not null
 6        then
 7             replace(
 8           translate(mixed,
 9             replace(
10           translate(mixed,'0123456789','9999999999'),'9'),
11                     rpad('#',length(mixed),'#')),'#')
12        else
13             mixed
14        end
15        ) mixed
16   from V
17  where instr(translate(mixed,'0123456789','9999999999'),'9') > 0
```

PostgreSQL

С помощью функций TRANSLATE, REPLACE и INSTR выделяем числовые значения в каждой строке. В представлении V вызывать CAST необязательно. Функция REPLACE используется для удаления лишних пробелов, возникающих в результате приведения к типу фиксированной длины CHAR. По желанию можно выполнять явное приведение в описании представления, предлагается приводить к типу VARCHAR:

```
 1 select cast(
 2        case
 3        when
 4         replace(translate(mixed,'0123456789','9999999999'),'9','')
 5         is not null
 6        then
 7           replace(
 8         translate(mixed,
 9           replace(
10         translate(mixed,'0123456789','9999999999'),'9',''),
```

```
11                  rpad('#',length(mixed),'#')),'#','')
12       else
13         mixed
14       end as integer ) as mixed
15    from V
16  where strpos(translate(mixed,'0123456789','9999999999'),'9') > 0
```

SQL Server

Без труда найти строки, содержащие числа, позволяет встроенная функция ISNUMERIC с групповым символом. Но извлечь числовые символы из строки не так просто, потому что функция TRANSLATE не поддерживается.

Обсуждение

Функция TRANSLATE здесь очень полезна, поскольку позволяет выделять и идентифицировать числа и символы. Хитрость состоит в замене всех чисел одним символом, в результате чего ведется поиск не разных чисел, а всего одного символа.

DB2, Oracle и PostgreSQL

Синтаксис для этих СУБД немного отличается, но прием один. Обсудим на примере решения для PostgreSQL.

Основную работу выполняют функции TRANSLATE и REPLACE. Для получения окончательного результирующего множества необходимо выполнить несколько вызовов функций, как показано в запросе ниже:

```
select mixed as orig,
translate(mixed,'0123456789','9999999999') as mixed1,
replace(translate(mixed,'0123456789','9999999999'),'9','') as mixed2,
 translate(mixed,
 replace(
 translate(mixed,'0123456789','9999999999'),'9',''),
          rpad('#',length(mixed),'#')) as mixed3,

 replace(
 translate(mixed,
 replace(
translate(mixed,'0123456789','9999999999'),'9',''),
         rpad('#',length(mixed),'#')),'#','') as mixed4
  from V
 where strpos(translate(mixed,'0123456789','9999999999'),'9') > 0
```

```
 ORIG  | MIXED1 | MIXED2 | MIXED3 | MIXED4 | MIXED5
-------+--------+--------+--------+--------+--------
 CL1OAR | CL99AR | CLAR   | ##1O## | 10     |     10
 KI1ONG | KI99NG | KING   | ##1O## | 10     |     10
 MI1OLL | MI99LL | MILL   | ##1O## | 10     |     10
 7369   | 9999   |        | 7369   | 7369   |   7369
 7566   | 9999   |        | 7566   | 7566   |   7566
```

```
7788   | 9999   |        | 7788   | 7788   |   7788
7876   | 9999   |        | 7876   | 7876   |   7876
7902   | 9999   |        | 7902   | 7902   |   7902
```

Во-первых, обратите внимание, что удалены все строки, в которых нет чисел. Как это было сделано, станет понятным после рассмотрения каждого из столбцов приведенного выше результирующего множества. Строки, которые были оставлены, представлены в столбце ORIG, и они, в конечном счете, образуют результирующее множество. Первый шаг для извлечения чисел – применение функции TRANSLATE, которая заменяет каждое число числом 9 (может использоваться любое число; 9 выбрано произвольно). Результаты этой операции приведены в столбце MIXED1. Теперь, когда все числа представлены 9, с ними можно работать как с одним элементом. Следующий шаг – удалить все числа с помощью функции REPLACE. Поскольку все числа теперь представлены 9, REPLACE просто ищет числа 9 и удаляет их. Результаты данной операции показаны в столбце MIXED2. Далее эти значения используются для получения значений MIXED3: они сравниваются со значениями столбца ORIG; все символы значения MIXED2, найденные в значении ORIG, функция TRANSLATE заменяет символом #. Результаты, выведенные в MIXED3, показывают, что теперь выделены и заменены одним символом буквы, а не числа, и они могут обрабатываться как один элемент. Следующий шаг – использовать REPLACE для поиска и удаления из строк всех символов #. В строках сохраняются только числа, что показано в столбце MIXED4. Теперь остается только привести числовые символы к числовому типу данных. После поэтапного разбора всего процесса становится понятным, что делает предикат WHERE. Результаты из столбца MIXED1 передаются в STRPOS, и если обнаружена 9 (местоположение первой 9 в строке), результат должен быть больше 0. Если для строки возвращается значение больше нуля, значит, в этой строке есть, по крайней мере, одно число, и строка должна быть сохранена.

MySQL

Первый шаг – выполнить обход каждой строки, провести ее синтаксический разбор и определить, содержатся ли в ней числа:

```
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
      ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
 order by 1,2
```

```
+--------+------+------+
| mixed  | pos  | c    |
+--------+------+------+
| 7369   |    1 | 7    |
| 7369   |    2 | 3    |
| 7369   |    3 | 6    |
```

```
| 7369   |    4 | 9    |
...
| ALLEN  |    1 | A    |
| ALLEN  |    2 | L    |
| ALLEN  |    3 | L    |
| ALLEN  |    4 | E    |
| ALLEN  |    5 | N    |
...
| CL10AR |    1 | C    |
| CL10AR |    2 | L    |
| CL10AR |    3 | 1    |
| CL10AR |    4 | 0    |
| CL10AR |    5 | A    |
| CL10AR |    6 | R    |
+--------+------+------+
```

Теперь каждый символ строки может обрабатываться индивидуально. Следующий шаг – выбрать только строки, содержащие число в столбце C:

```
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
      ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
  and ascii(substr(v.mixed,iter.pos,1)) between 48 and 57
 order by 1,2
```

```
+--------+------+------+
| mixed  | pos  | c    |
+--------+------+------+
| 7369   |    1 | 7    |
| 7369   |    2 | 3    |
| 7369   |    3 | 6    |
| 7369   |    4 | 9    |
...
| CL10AR |    3 | 1    |
| CL10AR |    4 | 0    |
...
+--------+------+------+
```

На этом этапе в столбце C находятся только числа. Следующий шаг – с помощью функции GROUP_CONCAT выполнить конкатенацию чисел и получить соответствующие значениям столбца MIXED целые числа. Полученный результат приводится к числовому типу:

```
select cast(group_concat(c order by pos separator '') as unsigned)
         as MIXED1
  from (
select v.mixed, iter.pos, substr(v.mixed,iter.pos,1) as c
  from V,
      ( select id pos from t10 ) iter
 where iter.pos <= length(v.mixed)
  and ascii(substr(x.mixed,iter.pos,1)) between 48 and 57
       ) y
```

```
  group by mixed
  order by 1
```

```
+--------+
| MIXED1 |
+--------+
|    10  |
|    10  |
|    10  |
|  7369  |
|  7566  |
|  7788  |
|  7876  |
|  7902  |
+--------+
```

И последнее замечание – помните, что конкатенации подвергаются все цифры строки, и в результате получается одно числовое значение. Например, возьмем исходное значение «99Gennick87». В результате будет получено значение 9987. Об этом следует помнить, особенно при работе с сериализованными данными.

Извлечение n-ной подстроки с разделителями

Задача

Требуется извлечь из строки заданную подстроку с разделителями. Рассмотрим следующее представление V, формирующее исходные данные для этой задачи:

```
create view V as
select 'mo,larry,curly' as name
  from t1
 union all
select 'tina,gina,jaunita,regina,leena' as name
  from t1
```

В результате получаем следующее:

```
select * from v
```

```
NAME
--------------------
mo,larry,curly
tina,gina,jaunita,regina,leena
```

Из каждой строки необходимо выбрать второе имя. Результирующее множество должно быть таким:

```
  SUB
-----
larry
gina
```

Решение

Суть решения данной задачи – возвратить каждое имя в отдельной строке, сохраняя порядок, в котором имена представлены в списке. Конкретная реализация этого зависит от используемой СУБД.

DB2

После обхода значений столбца NAME, возвращенных представлением V, с помощью функции ROW_NUMBER выбираем только второе имя каждой строки:

```
 1 select substr(c,2,locate(',',c,2)-2)
 2   from (
 3 select pos, name, substr(name, pos) c,
 4        row_number() over(partition by name
 5                          order by length(substr(name,pos)) desc) rn
 6   from (
 7 select ',' ||csv.name|| ',' as name,
 8        cast(iter.pos as integer) as pos
 9   from V csv,
10        (select row_number() over() pos from t100 ) iter
11  where iter.pos <= length(csv.name)+2
12        )  x
13  where length(substr(name,pos)) > 1
14    and substr(substr(name,pos),1,1) = ','
15        ) y
16  where rn = 2
```

MySQL

После обхода значений столбца NAME, возвращенных представлением V, выбираем из каждой строки второе имя, ориентируясь на позиции запятых:

```
 1 select name
 2   from (
 3 select iter.pos,
 4        substring_index(
 5        substring_index(src.name, ',',iter.pos), ',',-1) name
 6   from V src,
 7        (select id pos from t10) iter,
 8  where iter.pos <=
 9        length(src.name)-length(replace(src.name, ',',''))
10         ) x
11 where pos = 2
```

Oracle

После обхода значений столбца NAME, возвращенных представлением V, второе имя каждого списка извлекаем с помощью функций SUBSTR и INSTR:

```
 1 select sub
 2   from (
 3 select iter.pos,
 4        src.name,
 5        substr( src.name,
 6         instr( src.name,',',1,iter.pos )+1,
 7         instr( src.name,',',1,iter.pos+1 ) -
 8         instr( src.name,',',1,iter.pos )-1) sub
 9   from (select ','||name||',' as name from V) src,
10        (select rownum pos from emp) iter
11  where iter.pos < length(src.name)-length(replace(src.name,','))
12        )
13  where pos = 2
```

PostgreSQL

С помощью функции SPLIT_PART возвращаем все имена в отдельных строках:

```
 1 select name
 2   from (
 3 select iter.pos, split_part(src.name,',',iter.pos) as name
 4   from (select id as pos from t10) iter,
 5        (select cast(name as text) as name from v) src
 7  where iter.pos <=
 8         length(src.name)-length(replace(src.name,',',''))+1
 9        ) x
10  where pos = 2
```

SQL Server

После обхода значений столбца NAME, возвращенных представлением V, с помощью функции ROW_NUMBER выбираем из каждой строки только второе имя:

```
 1  select substring(c,2,charindex(',',c,2)-2)
 2    from (
 3  select pos, name, substring(name, pos, len(name)) as c,
 4         row_number() over(
 5          partition by name
 6          order by len(substring(name,pos,len(name))) desc) rn
 7    from (
 8  select ',' + csv.name + ',' as name,
 9         iter.pos
10    from V csv,
11         (select id as pos from t100 ) iter
12   where iter.pos <= len(csv.name)+2
13         ) x
14   where len(substring(name,pos,len(name))) > 1
15     and substring(substring(name,pos,len(name)),1,1) = ','
16         ) y
17   where rn = 2
```

Обсуждение

DB2 и SQL Server

Синтаксис для этих двух СУБД немного отличается, но техника – одна. Обсудим ее на примере решения для DB2. Выполняется обход строки, результаты представляются вложенным запросом X:

```
select  ','||csv.name|| ',' as name,
         iter.pos
  from v csv,
       (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
```

```
EMPS                              POS
--------------------------------  ----
,tina,gina,jaunita,regina,leena,     1
,tina,gina,jaunita,regina,leena,     2
,tina,gina,jaunita,regina,leena,     3
...
```

Следующий шаг – перебрать все элементы каждой строки:

```
select pos, name, substr(name, pos) c,
       row_number() over(partition by name
                         order by length(substr(name, pos)) desc) rn
  from (
select ','||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
      (select row_number() over() pos from t100 ) iter
 where iter.pos <= length(csv.name)+2
       )  x
 where length(substr(name,pos)) > 1
```

```
POS  EMPS              C                 RN
---  ---------------   ----------------  --
  1  ,mo,larry,curly,  ,mo,larry,curly,   1
  2  ,mo,larry,curly,  mo,larry,curly,    2
  3  ,mo,larry,curly,  o,larry,curly,     3
  4  ,mo,larry,curly,  ,larry,curly,      4
  ...
```

Теперь, имея доступ ко всем частям строки, просто выбираем необходимые строки представления. Нас интересуют строки, начинающиеся с запятой; остальные можно отбросить:

```
select pos, name, substr(name,pos) c,
       row_number() over(partition by name
                         order by length(substr(name, pos)) desc) rn
  from (
select ','||csv.name||',' as name,
       cast(iter.pos as integer) as pos
  from v csv,
       (select row_number() over() pos from t100 ) iter
```

```
 where iter.pos <= length(csv.name)+2
       )  x
 where length(substr(name,pos)) > 1
    and substr(substr(name,pos),1,1) = ','
```

```
POS   EMPS                              C                                  RN
 ---   --------------                    ----------------                   --
   1   ,mo,larry,curly,                  ,mo,larry,curly,                    1
   4   ,mo,larry,curly,                  ,larry,curly,                       2
  10   ,mo,larry,curly,                  ,curly,                             3
   1   ,tina,gina,jaunita,regina,leena,  ,tina,gina,jaunita,regina,leena,    1
   6   ,tina,gina,jaunita,regina,leena,  ,gina,jaunita,regina,leena,         2
  11   ,tina,gina,jaunita,regina,leena,  ,jaunita,regina,leena,              3
  19   ,tina,gina,jaunita,regina,leena,  ,regina,leena,                      4
  26   ,tina,gina,jaunita,regina,leena,  ,leena,                             5
```

Это важный шаг, потому что он определяет, как будет получена *n*-ная подстрока. Обратите внимание, что многие строки были исключены из этого запроса согласно следующему условию WHERE:

```
substr(substr(name,pos),1,1) = ','
```

Заметим, что строка `,larry,curly,` имела ранг 4, а сейчас он равен 2. Вспомним, что предикат WHERE обрабатывается раньше SELECT. Таким образом, сохраняются строки, первым символом которых является запятая, *после этого* ROW_NUMBER ранжирует их. В данный момент легко увидеть, что чтобы получить *n*-ную подстроку, необходимо выбрать строки, где значение RN равно *n*. Последний шаг – выбрать интересующие нас строки (в данном случае те, в которых RN равно 2) и с помощью SUBSTR извлечь из них имя. Выбрать требуется первое имя строки: `larry` из `,larry,curly,` и `gina` из `,gina,jaunita,regina,leena,`.

MySQL

Вложенный запрос X выполняет обход всех строк. Определить количество значений в строке можно, подсчитав число разделителей в ней:

```
select iter.pos, src.name
  from (select id pos from t10) iter,
       V src
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))
```

```
+------+--------------------------------+
| pos  | name                           |
+------+--------------------------------+
|    1 | mo,larry,curly                 |
|    2 | mo,larry,curly                 |
|    1 | tina,gina,jaunita,regina,leena |
|    2 | tina,gina,jaunita,regina,leena |
|    3 | tina,gina,jaunita,regina,leena |
|    4 | tina,gina,jaunita,regina,leena |
+------+--------------------------------+
```

В данном случае строк в таблице на одну меньше, чем значений в строке. Выбор необходимых значений обеспечит функция SUBSTRING_INDEX:

```
select iter.pos,src.name name1,
       substring_index(src.name,',',iter.pos) name2,
       substring_index(
        substring_index(src.name,',',iter.pos),',',-1) name3
  from (select id pos from t10) iter,
       V src
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))
```

```
+-----+--------------------------------+-------------------------+---------+
| pos | name1                          | name2                   | name3   |
+-----+--------------------------------+-------------------------+---------+
|   1 | mo,larry,curly                 | mo                      | mo      |
|   2 | mo,larry,curly                 | mo,larry                | larry   |
|   1 | tina,gina,jaunita,regina,leena | tina                    | tina    |
|   2 | tina,gina,jaunita,regina,leena | tina,gina               | gina    |
|   3 | tina,gina,jaunita,regina,leena | tina,gina,jaunita       | jaunita |
|   4 | tina,gina,jaunita,regina,leena | tina,gina,jaunita,regina | regina  |
+-----+--------------------------------+-------------------------+---------+
```

Чтобы продемонстрировать работу функции SUBSTRING_INDEX, показаны три поля имен. Внутренний вызов возвращает все символы, расположенные левее n-ной запятой. Внешний вызов возвращает все, что находится правее первой обнаруженной запятой (начиная с конца строки). Последний шаг – выбрать значения, для которых POS равно n, в данном случае нас интересует 2, и поместить их в столбец NAME3.

Oracle

Вложенный запрос осуществляет обход каждой строки. То, сколько раз будет возвращена строка, зависит от количества значений в ней. Количество значений в строке определяется посредством подсчета числа разделителей в ней. Поскольку каждая строка заключена в запятые, количество значений в строке равно числу запятых минус один. Потом строки объединяются оператором UNION и добавляются в таблицу с кардинальностью, не меньшей количества значений в самой длинной строке. Функции SUBSTR и INSTR используют значение POS для синтаксического разбора каждой строки:

```
select iter.pos, src.name,
       substr( src.name,
        instr( src.name,',',1,iter.pos )+1,
        instr( src.name,',',1,iter.pos+1 ) -
        instr( src.name,',',1,iter.pos )-1) sub
  from (select ','||name||',' as name from v) src,
       (select rownum pos from emp) iter
 where iter.pos < length(src.name)-length(replace(src.name,','))
```

```
POS NAME                              SUB
--- --------------------------------- -------------
  1 ,mo,larry,curly,                  mo
  1 , tina,gina,jaunita,regina,leena,  tina
  2 ,mo,larry,curly,                  larry
  2 , tina,gina,jaunita,regina,leena, gina
  3 ,mo,larry,curly,                  curly
  3 , tina,gina,jaunita,regina,leena, jaunita
  4 , tina,gina,jaunita,regina,leena, regina
  5 , tina,gina,jaunita,regina,leena, leena
```

Первый вызов INSTR в SUBSTR определяет начальную позицию необходимой нам подстроки. Следующий вызов INSTR в SUBSTR находит позицию *n*-ной запятой (совпадает с начальной позицией), а также *n*-ной+1 запятой. Разность этих значений соответствует длине извлекаемой подстроки. Поскольку в результате синтаксического разбора каждое значение занимает отдельную строку, для получения *n*-ной подстроки просто задаем WHERE POS = *n* (в данном случае where POS = 2, т. е. получаем вторую подстроку списка).

PostgreSQL

Вложенный запрос X осуществляет обход всех строк. Число возвращенных строк зависит от количества значений в каждой строке. Чтобы найти, сколько значений в каждой строке, находим число разделителей в ней и добавляем единицу. Функция SPLIT_PART использует значения столбца POS для поиска *n*-ного разделителя и раскладывает строку на значения:

```
select iter.pos, src.name as name1,
       split_part(src.name,',',iter.pos) as name2
  from (select id as pos from t10) iter,
       (select cast(name as text) as name from v) src
 where iter.pos <=
       length(src.name)-length(replace(src.name,',',''))+1
```

```
 pos |             name1              |  name2
-----+--------------------------------+---------
   1 | mo,larry,curly                 | mo
   2 | mo,larry,curly                 | larry
   3 | mo,larry,curly                 | curly
   1 | tina,gina,jaunita,regina,leena | tina
   2 | tina,gina,jaunita,regina,leena | gina
   3 | tina,gina,jaunita,regina,leena | jaunita
   4 | tina,gina,jaunita,regina,leena | regina
   5 | tina,gina,jaunita,regina,leena | leena
```

Здесь показаны два столбца имен, чтобы можно было увидеть, как SPLIT_PART проводит синтаксический разбор строк, используя значение POS. После того как разбор всех строк завершен, заключительный шаг – выбрать строки, в которых значение POS равно *n*-ной подстроке, в данном случае 2.

Синтаксический разбор IP-адреса

Задача

Требуется провести синтаксический разбор IP-адреса и разместить его поля в отдельные столбцы. Рассмотрим следующий IP-адрес:

```
111.22.3.4
```

В результате запроса должен быть получен такой результат:

```
A     B     C     D
----- ----- ----- ---
111   22    3     4
```

Решение

Решение опирается на встроенные функции, предоставляемые СУБД. Ключом к решению, независимо от СУБД, является выявление точек и выбора чисел, их окружающих.

DB2

С помощью рекурсивного блока WITH моделируйте перебор IP-адреса и используйте функцию SUBSTR для синтаксического разбора. Добавьте в начало IP-адреса точку с целью обеспечения единообразия при обработке каждой группы чисел.

```
 1 with x (pos,ip) as (
 2   values (1,'.92.111.0.222')
 3   union all
 4  select pos+1,ip from x where pos+1 <= 20
 5 )
 6 select max(case when rn=1 then e end) a,
 7        max(case when rn=2 then e end) b,
 8        max(case when rn=3 then e end) c,
 9        max(case when rn=4 then e end) d
10   from (
11 select pos,c,d,
12        case when posstr(d,'.') > 0 then substr(d,1,posstr(d,'.')-1)
13             else d
14        end as e,
15        row_number() over(order by pos desc) rn
16   from (
17 select pos, ip,right(ip,pos) as c, substr(right(ip,pos),2) as d
18   from x
19  where pos <= length(ip)
20    and substr(right(ip,pos),1,1) = '.'
21       ) x
22       ) y
```

MySQL

Функция SUBSTR_INDEX упрощает задачу синтаксического разбора IP-адреса:

```
select substring_index(substring_index(y.ip,'.',1),'.',-1) a,
       substring_index(substring_index(y.ip,'.',2),'.',-1) b,
       substring_index(substring_index(y.ip,'.',3),'.',-1) c,
       substring_index(substring_index(y.ip,'.',4),'.',-1) d
  from (select '92.111.0.2' as ip from t1) y
```

Oracle

Для проведения синтаксического разбора и обхода IP-адреса используйте встроенные функции SUBSTR и INSTR:

```
select ip,
       substr(ip, 1, instr(ip,'.')-1 ) a,
       substr(ip, instr(ip,'.')+1,
                   instr(ip,'.',1,2)-instr(ip,'.')-1 ) b,
       substr(ip, instr(ip,'.',1,2)+1,
                   instr(ip,'.',1,3)-instr(ip,'.',1,2)-1 ) c,
       substr(ip, instr(ip,'.',1,3)+1 ) d
  from (select '92.111.0.2' as ip from t1)
```

PostgreSQL

Для проведения синтаксического разбора IP-адреса используйте встроенную функцию SPLIT_PART:

```
select split_part(y.ip,'.',1) as a,
       split_part(y.ip,'.',2) as b,
       split_part(y.ip,'.',3) as c,
       split_part(y.ip,'.',4) as d
  from (select cast('92.111.0.2' as text) as ip from t1) as y
```

SQL Server

С помощью рекурсивного оператора WITH моделируйте перебор символов IP-адреса, для синтаксического разбора используйте функцию SUBSTRING. Добавьте в начало IP-адреса точку с целью обеспечения единообразия при обработке каждой группы чисел:

```
 with x (pos,ip) as (
   select 1 as pos,'.92.111.0.222' as ip from t1
   union all
  select pos+1,ip from x where pos+1 <= 20
 )
 select max(case when rn=1 then e end) a,
        max(case when rn=2 then e end) b,
        max(case when rn=3 then e end) c,
        max(case when rn=4 then e end) d
   from (
select pos,c,d,
```

```
12          case when charindex('.',d) > 0
13               then substring(d,1,charindex('.',d)-1)
14               else d
15          end as e,
16          row_number() over(order by pos desc) rn
17     from (
18   select pos, ip,right(ip,pos) as c,
19          substring(right(ip,pos),2,len(ip)) as d
20     from x
21    where pos <= len(ip)
22      and substring(right(ip,pos),1,1) = '.'
23         ) x
24         ) y
```

Обсуждение

Встроенные функции базы данных позволяют без труда реализовать обход частей строки. Главное – найти все точки, после этого можно проводить синтаксический разбор чисел, располагающихся между ними.

7

Работа с числами

Данная глава посвящена операциям над числами, в том числе различным вычислениям. Хотя SQL не тот инструмент, который первым приходит на ум при необходимости выполнить сложные вычисления, он очень эффективен для повседневных рутинных операций с числами.

В некоторых рецептах данной главы используются агрегатные функции и оператор GROUP BY. Если вы не знакомы с группировкой, пожалуйста, прочитайте хотя бы первый основной раздел приложения А «Группировка».

Вычисление среднего

Задача

Требуется вычислить среднее значение столбца либо для всех строк таблицы, либо для некоторого подмножества строк. Например, поставлена задача найти среднюю заработную плату для всех служащих предприятия, а также среднее ее значение в каждом отделе.

Решение

Для вычисления средней заработной платы для всех служащих просто применяем функцию AVG к столбцу, содержащему эти данные. Поскольку предикат WHERE не используется, среднее вычисляется для всех определенных (не-NULL) значений:

```
1 select avg(sal) as avg_sal
2   from emp

   AVG_SAL
----------
2073.21429
```

Чтобы вычислить среднюю заработную плату для каждого отдела, посредством оператора GROUP BY создаем соответствующие группы:

```
1 select deptno, avg(sal) as avg_sal
2   from emp
3  group by deptno
```

```
    DEPTNO    AVG_SAL
---------- ----------
        10 2916.66667
        20       2175
        30 1566.66667
```

Обсуждение

Для вычисления среднего значения для всей таблицы просто применяем к соответствующему столбцу функцию AVG без использования оператора GROUP BY. Важно понимать, что функция AVG игнорирует значения NULL. Результат такого поведения можно увидеть ниже:

```
create table t2(sal integer)
insert into t2 values (10)
insert into t2 values (20)
insert into t2 values (null)
```

```
select avg(sal)
  from t2
```

```
  AVG(SAL)
----------
        15
```

```
select distinct 30/2
  from t2
```

```
      30/2
----------
        15
```

```
select avg(coalesce(sal,0))
  from t2
```

```
AVG(COALESCE(SAL,0))
--------------------
                  10
```

```
select distinct 30/3
  from t2
```

```
      30/3
----------
        10
```

Функция COALESCE возвратит первое не-NULL значение переданного в нее списка значений. Если значение столбца SAL преобразовать в нуль, среднее изменится. При работе с агрегатными функциями всегда следует продумать, как должны обрабатываться значения NULL.

Вторая часть решения использует оператор GROUP BY (строка 3) для разделения записей служащих на группы на основании принадлежности к тому или иному отделу. GROUP BY обусловливает выполнение агрегатных функций, таких как AVG, и возвращение результата для каждой группы. В данном примере AVG выполняется по одному разу для каждой группы записей служащих каждого отдела.

Кстати, необязательно включать столбцы, используемые в предложении GROUP BY, в список оператора SELECT. Например:

```
 select avg(sal)
   from emp
  group by deptno

  AVG(SAL)
----------
2916.66667
      2175
1566.66667
```

По-прежнему выполняется группировка по столбцу DEPTNO, хотя он не указан в списке SELECT. Включение столбца, по которому проводится группировка, в оператор SELECT делает код более понятным, но не является обязательным. Однако необходимо, чтобы все столбцы списка SELECT запроса GROUP BY были указаны в операторе GROUP BY.

См. также

Приложение А для повторения функциональных возможностей GROUP BY.

Поиск минимального/максимального значения столбца

Задача

Требуется найти наибольшее и наименьшее значения заданного столбца; например, наибольшую и наименьшую заработные платы среди всех служащих предприятия, а также для каждого отдела.

Решение

При поиске самой низкой и самой высокой заработных плат среди всех служащих просто используются функции MIN и MAX соответственно:

```
1 select min(sal) as min_sal, max(sal) as max_sal
2   from emp

   MIN_SAL    MAX_SAL
---------- ----------
       800       5000
```

При поиске самой низкой и самой высокой заработных плат для каждого отдела используются функции MIN и MAX в сочетании с оператором GROUP BY:

```
1 select deptno, min(sal) as min_sal, max(sal) as max_sal
2   from emp
3  group by deptno

    DEPTNO    MIN_SAL    MAX_SAL
---------- ---------- ----------
```

```
        10        1300        5000
        20         800        3000
        30         950        2850
```

Обсуждение

При поиске наибольшего или наименьшего значений во всей таблице к соответствующему столбцу просто применяется функция MIN или MAX без использования оператора GROUP BY.

Не забывайте, что функции MIN и MAX игнорируют значения NULL. Для заданной группы в рассматриваемом столбце могут содержаться как отдельные значения NULL, так и только значения NULL. Далее представлены примеры, в последнем из которых используется оператор GROUP BY, возвращающий значения NULL для двух групп (DEPTNO 10 и 20):

```
select deptno, comm
  from emp
 where deptno in (10,30)
 order by 1
```

```
    DEPTNO       COMM
---------- ----------
        10
        10
        10
        30        300
        30        500
        30
        30          0
        30       1300
        30
```

```
select min(comm), max(comm)
  from emp
```

```
 MIN(COMM)  MAX(COMM)
---------- ----------
         0       1300
```

```
select deptno, min(comm), max(comm)
  from emp
 group by deptno
```

```
    DEPTNO  MIN(COMM)  MAX(COMM)
---------- ---------- ----------
        10
        20
        30          0       1300
```

Как указано в Приложении А, даже если в операторе SELECT, кроме агрегатной функции, не указано никаких столбцов таблицы, группировку можно осуществлять по другим столбцам. Например:

```
select min(comm), max(comm)
  from emp
 group by deptno

 MIN(COMM)  MAX(COMM)
---------- ----------
         0       1300
```

Здесь группировка проводится по столбцу DEPTNO, хотя его нет в списке оператора SELECT. Включение в список SELECT столбца, по которому осуществляется группировка, делает код более понятным, но не является обязательным. Однако все столбцы списка SELECT запроса GROUP BY должны быть указаны в операторе GROUP BY.

См. также

Приложение А для повторения функциональных возможностей GROUP BY.

Вычисление суммы значений столбца

Задача

Требуется вычислить сумму всех значений столбца, например заработных плат всех служащих.

Решение

При вычислении суммы для всей таблицы просто применяем функцию SUM к соответствующим столбцам без использования оператора GROUP BY:

```
1 select sum(sal)
2   from emp

  SUM(SAL)
----------
     29025
```

При создании нескольких групп или «окон» данных используется функция SUM в сочетании с оператором GROUP BY. В следующем примере вычисляются суммы заработных плат служащих по отделам:

```
1 select deptno, sum(sal) as total_for_dept
2   from emp
3  group by deptno

    DEPTNO TOTAL_FOR_DEPT
---------- --------------
        10           8750
        20          10875
        30           9400
```

Обсуждение

При вычислении суммы всех заработных плат для каждого отдела создаются группы или «окна» данных. Заработная плата всех служащих одного отдела складывается и получается общая сумма для этого отдела. Это пример агрегации в SQL, поскольку предметом рассмотрения является не детальная информация, такая как заработная плата каждого отдельно взятого служащего, а конечный результат для всего отдела. Важно отметить, что функция SUM игнорирует значения NULL, но для группы SUM может возвращать NULL, что показано ниже. Служащие 10-го отдела (DEPTNO 10) не получают комиссионных, таким образом, при группировке по DEPTNO 10 и суммировании значений столбца COMM получаем группу, для которой SUM возвращает значение NULL:

```
select deptno, comm
  from emp
 where deptno in (10,30)
 order by 1

    DEPTNO       COMM
---------- ----------
        10
        10
        10
        30        300
        30        500
        30
        30          0
        30       1300
        30

select sum(comm)
  from emp

 SUM(COMM)
----------
      2100

select deptno, sum(comm)
  from emp
 where deptno in (10,30)
 group by deptno

    DEPTNO  SUM(COMM)
---------- ----------
        10
        30       2100
```

См. также

Приложение A для повторения функциональных возможностей GROUP BY.

Подсчет строк в таблице

Задача

Требуется подсчитать число строк в таблице или количество значений в столбце. Например, необходимо найти общее число служащих и количество служащих в каждом отделе.

Решение

При подсчете строк всей таблицы просто используем функцию COUNT с символом «*»:

```
1 select count(*)
2   from emp
```

```
  COUNT(*)
----------
        14
```

При создании нескольких групп или «окон» данных используем функцию COUNT с оператором GROUP BY:

```
1 select deptno, count(*)
2   from emp
3  group by deptno
```

```
    DEPTNO   COUNT(*)
---------- ----------
        10          3
        20          5
        30          6
```

Обсуждение

При подсчете количества служащих в каждом отделе мы создаем группы или «окна» данных. При выявлении каждого нового служащего значение счетчика увеличивается на единицу, таким образом, вычисляется общее значение для соответствующего отдела. Это пример агрегации в SQL, поскольку рассматриваются не заработная плата или должность каждого отдельно взятого служащего, а конечный результат для каждого отдела. Важно отметить, что функция COUNT игнорирует значения NULL для столбца, имя которого передается в нее в качестве аргумента, но включает значения NULL при использовании символа «*» или любой константы. Рассмотрим:

```
select deptno, comm
  from emp
```

```
    DEPTNO       COMM
---------- ----------
        20
        30        300
```

```
        30         500
        20
        30        1300
        30
        10
        20
        10
        30           0
        20
        30
        20
        10
```

```
select count(*), count(deptno), count(comm), count('hello')
  from emp
```

```
  COUNT(*) COUNT(DEPTNO) COUNT(COMM) COUNT('HELLO')
---------- ------------- ----------- --------------
        14            14           4             14
```

```
select deptno, count(*), count(comm), count('hello')
  from emp
 group by deptno
```

```
    DEPTNO   COUNT(*) COUNT(COMM) COUNT('HELLO')
---------- ---------- ----------- --------------
        10          3           0              3
        20          5           0              5
        30          6           4              6
```

Если во всех строках переданного в COUNT столбца содержатся значения NULL или если таблица пуста, COUNT возвратит нуль. Следует также заметить, что даже если в операторе SELECT определена только агрегатная функция, по-прежнему можно осуществлять группировку по другим столбцам таблицы; например:

```
select count(*)
  from emp
 group by deptno
```

```
  COUNT(*)
----------
         3
         5
         6
```

Обратите внимание, что группировка осуществляется по DEPTNO, несмотря на то, что этот столбец не указан в SELECT. Включение в список оператора SELECT столбца, по которому осуществляется группировка, делает код более понятным, но не является обязательным. Если все-таки столбец включен (в список SELECT), он обязательно должен быть указан в операторе GROUP BY.

См. также

Приложение A для повторения функциональных возможностей GROUP BY.

Подсчет значений в столбце

Задача

Требуется подсчитать количество определенных (не-NULL) значений в столбце. Например, стоит задача выяснить, сколько служащих получают комиссионные.

Решение

Подсчитываем число не-NULL значений в столбце COMM таблицы EMP, используя функцию COUNT и имя столбца в качестве ее аргумента:

```
select count(comm)
  from emp

COUNT(COMM)
-----------
          4
```

Обсуждение

Выражение COUNT(*) обеспечивает подсчет всех строк (независимо от их значений; поэтому учитываются как строки со значениями NULL, так и строки с определенными значениями). Но когда в функции COUNT задан конкретный столбец, подсчитывается количество не-NULL значений в этом столбце. В обсуждении предыдущего рецепта затрагивалось это различие. В данном рецепте функция COUNT(COMM) возвращает число не-NULL значений в столбце COMM. Поскольку комиссионные получают только служащие, занимающие определенную должность, в результате выполнения COUNT(COMM) возвращает количество таких служащих.

Вычисление текущей суммы

Задача

Требуется вычислить текущую сумму значений столбца.

Решение

В качестве примера следующие решения показывают, как вычисляется текущая сумма заработных плат всех служащих. Для удобства чтения результаты по возможности упорядочены по столбцу SAL, чтобы можно было увидеть процесс вычисления текущих сумм.

DB2 и Oracle

Для вычисления текущей суммы используется аналитическая версия функции SUM:

```
1 select ename, sal,
2        sum(sal) over (order by sal,empno) as running_total
3   from emp
4   order by 2
```

```
ENAME             SAL RUNNING_TOTAL
---------- ---------- -------------
SMITH             800           800
JAMES             950          1750
ADAMS            1100          2850
WARD             1250          4100
MARTIN           1250          5350
MILLER           1300          6650
TURNER           1500          8150
ALLEN            1600          9750
CLARK            2450         12200
BLAKE            2850         15050
JONES            2975         18025
SCOTT            3000         21025
FORD             3000         24025
KING             5000         29025
```

MySQL, PostgreSQL и SQL Server

Для вычисления текущей суммы используется скалярный подзапрос (без оконной функции, такой как SUM OVER, нельзя так просто упорядочить результирующее множество по столбцу SAL, как в решении для DB2 и Oracle). В конечном счете сумма вычисляется правильно (окончательное значение совпадает с полученным в предыдущем рецепте), но промежуточные значения отличаются из-за отсутствия упорядоченности:

```
1 select e.ename, e.sal,
2        (select sum(d.sal) from emp d
3          where d.empno <= e.empno) as running_total
4   from emp e
5  order by 3
```

```
ENAME             SAL RUNNING_TOTAL
---------- ---------- -------------
SMITH             800           800
ALLEN            1600          2400
WARD             1250          3650
JONES            2975          6625
MARTIN           1250          7875
BLAKE            2850         10725
CLARK            2450         13175
SCOTT            3000         16175
KING             5000         21175
```

```
TURNER          1500          22675
ADAMS           1100          23775
JAMES            950          24725
FORD            3000          27725
MILLER          1300          29025
```

Обсуждение

Получение текущих сумм – одна из задач, решение которых упростили новые оконные функции ANSI. Для СУБД, не поддерживающих пока что эти функции, необходимо использовать скалярный подзапрос (объединение по полю с уникальными значениями).

DB2 и Oracle

Оконная функция SUM OVER упрощает задачу по вычислению текущей суммы. В операторе ORDER BY решения указан не только столбец SAL, но и столбец EMPNO (первичный ключ), чтобы исключить дублирование значений при вычислении текущей суммы. В противном случае возникает проблема с дубликатами, что иллюстрирует в следующем примере столбец RUNNING_TOTAL2:

```
select empno, sal,
       sum(sal)over(order by sal,empno) as running_total1,
       sum(sal)over(order by sal) as running_total2
  from emp
 order by 2
```

```
ENAME             SAL RUNNING_TOTAL1 RUNNING_TOTAL2
---------- ---------- -------------- --------------
SMITH             800            800            800
JAMES             950           1750           1750
ADAMS            1100           2850           2850
WARD             1250           4100           5350
MARTIN           1250           5350           5350
MILLER           1300           6650           6650
TURNER           1500           8150           8150
ALLEN            1600           9750           9750
CLARK            2450          12200          12200
BLAKE            2850          15050          15050
JONES            2975          18025          18025
SCOTT            3000          21025          24025
FORD             3000          24025          24025
KING             5000          29025          29025
```

Значения столбца RUNNING_TOTAL2 для служащих WARD, MARTIN, SCOTT и FORD неверны. Их заработные платы встречаются несколько раз, и дубликаты также вошли в текущую сумму. Вот почему для формирования (правильных) результатов, которые показаны в столбце RUNNING_TOTAL1, необходимо включать в оператор ORDER BY и столбец EMPNO (значения которого уникальны). Рассмотрим следующее: для служащего ADAMS в столбцах RUNNING_TOTAL1

и RUNNING_TOTAL2 указано значение 2850, добавляем к нему заработную плату служащего WARD, равную 1250, и получаем 4100, а в столбце RUNNING_TOTAL2 возвращено 5350. Почему? Поскольку WARD и MARTIN имеют одинаковые значения в столбце SAL, их две зарплаты размером по 1250 складываются, образуя в сумме 2500, а затем это значение добавляется к 2850, что в итоге и дает 5350 как для WARD, так и для MARTIN. Задавая сочетание столбцов, которое не может иметь дублирующиеся значения (например, любое сочетание SAL и EMPNO уникально), мы гарантируем правильное вычисление текущей суммы.

MySQL, PostgreSQL и SQL Server

Пока данные СУБД не обеспечивают полной поддержки оконных функций, для вычисления текущей суммы используется скалярный подзапрос. Необходимо провести объединение по столбцу с уникальными значениями, в противном случае, если в столбце имеются дублирующиеся значения, текущие суммы будут неверны. Ключ к решению данного рецепта – объединение по D.EMPNO с E.EMPNO, в результате чего возвращаются (суммируются) все значения D.SAL, для которых D.EMPNO меньше или равно E.EMPNO. Разобраться в этом можно, переписав скалярный подзапрос как объединение для небольшого числа служащих:

```
select e.ename as ename1, e.empno as empno1, e.sal as sal1,
       d.ename as ename2, d.empno as empno2, d.sal as sal2
  from emp e, emp d
 where d.empno <= e.empno
   and e.empno = 7566
```

```
ENAME          EMPNO1       SAL1 ENAME          EMPNO2       SAL2
---------- ---------- ---------- ---------- ---------- ----------
JONES            7566       2975 SMITH            7369        800
JONES            7566       2975 ALLEN            7499       1600
JONES            7566       2975 WARD             7521       1250
JONES            7566       2975 JONES            7566       2975
```

Каждое значение EMPNO2 сравнивается с каждым значением EMPNO1. В сумму включается значение столбца SAL2 каждой строки, для которой значение столбца EMPNO2 меньше или равно значению столбца EMPNO1. В этом фрагменте значения EMPNO для служащих SMITH, ALLEN, WARD и JONES сравниваются со значением EMPNO для JONES. Поскольку значения EMPNO для всех четырех служащих удовлетворяют условию (меньше или равны EMPNO для JONES), их заработные платы суммируются. Заработная плата любого служащего, значение EMPNO которого больше, чем значение для JONES, не будет включена в SUM (в данном фрагменте). Полный запрос суммирует заработные платы всех служащих, для которых соответствующее значение EMPNO меньше или равно 7934 (EMPNO служащего MILLER), которое является наибольшим значением EMPNO в таблице.

Вычисление текущего произведения

Задача

Требуется найти текущее произведение для числового столбца. Эта операция аналогична «Вычислению текущей суммы», но значения не складываются, а перемножаются.

Решение

В качестве примера во всех решениях вычисляются текущие произведения заработных плат служащих. Хотя практической пользы в этом нет, используемая техника может быть применена в других прикладных задачах.

DB2 и Oracle

Примените оконную функцию SUM OVER и воспользуйтесь возможностью производить умножение путем суммирования логарифмов:

```
1 select empno,ename,sal,
2        exp(sum(ln(sal))over(order by sal,empno)) as running_prod
3   from emp
4  where deptno = 10
```

```
EMPNO ENAME       SAL         RUNNING_PROD
----- ---------- ---- --------------------
 7934 MILLER     1300                 1300
 7782 CLARK      2450              3185000
 7839 KING       5000          15925000000
```

В SQL вычисление логарифмов отрицательных чисел и нуля является недопустимой операцией. Если в таблицах содержатся такие значения, необходимо предупредить их передачу в SQL-функцию LN. В данном решении такие меры предосторожности не предусмотрены в целях удобства чтения кода, но они должны быть предприняты в реальных запросах. Если в таблицах представлены исключительно отрицательные значения, ноль либо NULL, приведенное выше решение не подходит.

В качестве альтернативы в Oracle можно использовать оператор MODEL, который был введен в Oracle Database 10*g*. В следующем примере все значения SAL возвращены как отрицательные числа, чтобы показать, что отрицательные значения не представляют проблемы для вычисления текущих произведений:

```
1 select empno, ename, sal, tmp as running_prod
2   from (
3 select empno,ename,-sal as sal
4   from emp
5  where deptno=10
6        )
```

```
7  model
8    dimension by(row_number()over(order by sal desc) rn )
9    measures(sal, 0 tmp, empno, ename)
10   rules (
11     tmp[any] = case when sal[cv()-1] is null then sal[cv()]
12                     else tmp[cv()-1]*sal[cv()]
13                end
14   )
```

```
EMPNO ENAME       SAL         RUNNING_PROD
----- ---------- ---- --------------------
 7934 MILLER     -1300               -1300
 7782 CLARK      -2450             3185000
 7839 KING       -5000        -15925000000
```

MySQL, PostgreSQL и SQL Server

По-прежнему применяется подход с суммированием логарифмов, но поскольку данные платформы не поддерживают оконные функции, вместо них используется скалярный подзапрос:

```
1 select e.empno,e.ename,e.sal,
2        (select exp(sum(ln(d.sal)))
3           from emp d
4          where d.empno <= e.empno
5            and e.deptno=d.deptno) as running_prod
6  from emp e
7  where e.deptno=10
```

```
EMPNO ENAME       SAL         RUNNING_PROD
----- ---------- ---- --------------------
 7782 CLARK      2450                 2450
 7839 KING       5000             12250000
 7934 MILLER     1300          15925000000
```

Для SQL Server вместо LN используется функция LOG.

Обсуждение

Кроме решения с использованием оператора MODEL, которое применимо только для Oracle 10*g* Database и более поздних версий, все решения основываются на том, что сумму двух чисел можно найти:

1. Вычисляя соответствующие натуральные логарифмы
2. Суммируя эти логарифмы
3. Возводя результат в степень математической константы *e* (используя функцию EXP)

Единственный недостаток данного подхода в его непригодности для суммирования отрицательных или нулевых значений, потому что они выходят за рамки допустимых значений для логарифмов в SQL.

DB2 и Oracle

Принцип работы оконной функции SUM OVER описан в предыдущем рецепте «Вычисление текущей суммы».

В Oracle 10*g* Database и более поздних версиях вычисление текущего произведения можно реализовать с помощью оператора MODEL. Используя оператор MODEL и ранжирующую функцию ROW_NUMBER, мы без труда организуем доступ к предыдущим строкам. С каждым элементом списка оператора MEASURES работаем, как с массивом. Обращаться к массивам можно посредством элементов списка DIMENSION (которые являются значениями, возвращенными ROW_ NUMBER под псевдонимом RN):

```
select empno, ename, sal, tmp as running_prod,rn
  from (
select empno,ename,-sal as sal
  from emp
 where deptno=10
       )
 model
   dimension by(row_number()over(order by sal desc) rn )
   measures(sal, 0 tmp, empno, ename)
  rules ()
```

```
EMPNO ENAME              SAL RUNNING_PROD         RN
----- ---------- ---------- ------------ ----------
 7934 MILLER          -1300            0          1
 7782 CLARK           -2450            0          2
 7839 KING            -5000            0          3
```

Обратите внимание, что SAL[1] имеет значение -1300. Поскольку строки нумеруются последовательно, без пропусков, сослаться на предыдущую строку можно, вычитая 1 из порядкового номера текущей строки. Конструкция RULES:

```
rules (
   tmp[any] = case when  sal[cv()-1] is null then sal[cv()]
                  else tmp[cv()-1]*sal[cv()]
             end
 )
```

использует встроенный оператор ANY, чтобы можно было обрабатывать строки без точного указания их номеров в коде. В данном случае ANY принимает значения 1, 2 и 3. Исходное значение TMP[*n*] – нуль. Затем TMP[*n*] присваивается текущее значение (функция CV возвращает текущее значение), вычисляемое для соответствующей строки столбца SAL. TMP[1] изначально равно нулю, SAL[1] равно -1300. Значения для SAL[0] нет, поэтому TMP[1] присваивается значение SAL[1]. После того как TMP[1] задано, берем следующую строку, TMP[2]. Сначала вычисляется SAL[1] (SAL[CV()–1] равно SAL[1], потому что текущее значение ANY равно 2). SAL[1] не NULL, оно равно –

1300, поэтому TMP[2] присваивается результат произведения TMP[1] и SAL[2]. И так далее для всех строк.

MySQL, PostgreSQL и SQL Server

Описание подхода с использованием подзапроса, применяемого в решениях MySQL, PostgreSQL и SQL Server, можно найти в данной главе выше в разделе «Вычисление текущей суммы».

Результат решения, использующего подзапрос, будет немного отличаться от результата решений для Oracle и DB2 из-за сортировки по значениям EMPNO (текущее произведение вычисляется в другой последовательности). Как и при вычислении текущей суммы, суммированием управляет предикат скалярного подзапроса. В данном решении строки упорядочиваются по столбцу EMPNO, тогда как в решении для Oracle/DB2 упорядочивание осуществляется по SAL.

Вычисление текущей разности

Задача

Требуется вычислить текущие разности для значений числового столбца. Например, стоит задача найти текущую разность заработных плат служащих 10-го отдела (DEPTNO 10). Должно быть получено следующее результирующее множество:

```
ENAME             SAL RUNNING_DIFF
---------- ---------- ------------
MILLER           1300         1300
CLARK            2450        -1150
KING             5000        -6150
```

Решение

DB2 и Oracle

Для вычисления текущей разности используйте оконную функцию SUM OVER:

```
1  select ename,sal,
2         sum(case when rn = 1 then sal else -sal end)
3          over(order by sal,empno) as running_diff
4    from (
5  select empno,ename,sal,
6         row_number()over(order by sal,empno) as rn
7    from emp
8   where deptno = 10
9         ) x
```

MySQL, PostgreSQL и SQL Server

Для вычисления текущей разности используйте скалярный подзапрос:

```
1 select a.empno, a.ename, a.sal,
2        (select case when a.empno = min(b.empno) then sum(b.sal)
3                     else sum(-b.sal)
4                end
5           from emp b
6          where b.empno <= a.empno
7            and b.deptno = a.deptno ) as rnk
8   from emp a
9  where a.deptno = 10
```

Обсуждение

Решения аналогичны рассматриваемым в рецепте «Вычисление текущей суммы». Единственное отличие в том, что все, кроме первого (точкой отсчета должно быть первое значение SAL в DEPTNO 10), значения столбца SAL возвращаются как отрицательные значения.

Вычисление моды

Задача

Требуется найти моду (для тех, кто забыл, *мода* (*mode*) в математике – это наиболее часто встречающийся элемент рассматриваемого множества данных) столбца значений. Например, поставлена задача найти моду заработных плат 20-го отдела (DEPTNO 20). Для следующего набора заработных плат:

```
select sal
  from emp
 where deptno = 20
 order by sal
```

```
       SAL
----------
       800
      1100
      2975
      3000
      3000
```

мода равна 3000.

Решение

DB2 и SQL Server

С помощью ранжирующей функции DENSE_RANK ранжируйте счетчики заработных плат, что поможет найти моду:

```
1  select sal
2    from (
3  select sal,
4         dense_rank()over(order by cnt desc) as rnk
```

```
 5    from (
 6  select sal, count(*) as cnt
 8    from emp
 9   where deptno = 20
10   group by sal
11         ) x
12         ) y
13   where rnk = 1
```

Oracle

Пользователям Oracle 8*i* Database подойдет решение для DB2. В Oracle 9*i* Database и более поздних версиях для поиска моды столбца SAL можно применять расширение KEEP к агрегатной функции MAX. Одно важное замечание: если есть одинаковые значения счетчиков, т. е. модой являются несколько строк, решение с использованием KEEP возвратит только одну из них, и это будет строка с наибольшей заработной платой. Если необходимо увидеть все моды (если их несколько), придется изменить это решение или просто использовать представленное выше решение для DB2. В данном случае, поскольку 3000 – мода столбца SAL для DEPTNO 20, а также наибольшее значение SAL, это решение подходит:

```
1  select max(sal)
2         keep(dense_rank first order by cnt desc) sal
3    from (
4  select sal, count(*) cnt
5    from emp
6   where deptno=20
7   group by sal
8         )
```

MySQL и PostgreSQL

Для поиска моды используйте подзапрос:

```
1  select sal
2    from emp
3   where deptno = 20
4   group by sal
5  having count(*) >= all ( select count(*)
6                             from emp
7                            where deptno = 20
8                            group by sal )
```

Обсуждение

DB2 и SQL Server

Вложенное представление X возвращает каждое значение столбца SAL и число раз, сколько это значение встречается в столбце. Вложенное представление Y использует ранжирующую функцию DENSE_RANK

(которая допускает одинаковые значения счетчиков) для сортировки результатов. Результаты ранжируются на основании того, сколько раз встречается каждое из значений SAL, как показано ниже:

```
1 select sal,
2        dense_rank()over(order by cnt desc) as rnk
3   from (
4 select sal,count(*) as cnt
5   from emp
6  where deptno = 20
7  group by sal
8        ) x
```

```
  SAL        RNK
----- ----------
 3000          1
  800          2
 1100          2
 2975          2
```

Самая внешняя часть запроса просто возвращает строку(и), для которых значение RNK равно 1.

Oracle

Вложенное представление возвращает каждое значение столбца SAL и число раз, сколько это значение встречается в столбце, как показано ниже:

```
select sal, count(*) cnt
  from emp
 where deptno=20
 group by sal
```

```
  SAL        CNT
----- ----------
  800          1
 1100          1
 2975          1
 3000          2
```

Следующий шаг – использовать расширение KEEP агрегатной функции MAX для поиска моды. Если проанализировать приведенный ниже оператор KEEP, можно заметить три подоператора, DENSE_RANK, FIRST и ORDER BY CNT DESC:

```
keep(dense_rank first order by cnt desc)
```

С их помощью очень удобно искать моду. Оператор KEEP определяет, какое значение SAL будет возвращено функцией MAX, по значению CNT, возвращенному вложенным представлением. Выражение выполняется справа налево. Сначала значения для CNT выстраиваются по убыванию, затем выбирается первое из них и возвращается в порядке, установленном функцией DENSE_RANK. Посмотрев на результирую-

щее множество вложенного представления, можно увидеть, что заработная плата 3000 имеет максимальное значение CNT, 2. MAX(SAL) возвращает самое большое значение SAL, которому соответствует наибольшое значение CNT, в данном случае это 3000.

См. также

Главу 11 раздел «Ход конем», в котором расширение агрегатных функций KEEP Oracle обсуждается более подробно.

MySQL и PostgreSQL

Подзапрос подсчитывает, сколько раз встречается каждое из значений столбца SAL. Внешний запрос возвращает каждое значение SAL, которое встречается чаще других или столько же раз, сколько все остальные значения (или иначе говоря, внешний запрос возвращает самые распространенные заработные платы для DEPTNO 20).

Вычисление медианы

Задача

Требуется найти медиану столбца числовых значений (для тех, кто не помнит, *медиана* (*median*) – это значение среднего члена множества упорядоченных элементов). Например, необходимо найти медиану заработных плат служащих 20-го отдела (DEPTNO 20). Для следующего набора зарплат:

```
select sal
  from emp
 where deptno = 20
 order by sal
```

```
       SAL
----------
       800
      1100
      2975
      3000
      3000
```

медианой является 2975.

Решение

Кроме решения Oracle (в котором для вычисления медианы используются предоставляемые СУБД функции), все решения основаны на методе, описанном Розенштейном, Абрамовичем и Бёргером в книге «Optimizing Transact-SQL: Advanced Programming Techniques» (SQL Forum Press, 1997). Введение ранжирующих функций обеспечивает более эффективное решение по сравнению с традиционным рефлексивным объединением.

DB2

Для поиска медианы используйте ранжирующие функции COUNT(*) OVER и ROW_NUMBER:

```
select avg(sal)
  from (
select sal,
       count(*) over() total,
       cast(count(*) over() as decimal)/2 mid,
       ceil(cast(count(*) over() as decimal)/2) next,
       row_number() over (order by sal) rn
  from emp
 where deptno = 20
       ) x
 where ( mod(total,2) = 0
         and rn in ( mid, mid+1 )
       )
    or ( mod(total,2) = 1
         and rn = next
       )
```

MySQL и PostgreSQL

Для поиска медианы используйте рефлексивное объединение:

```
select avg(sal)
  from (
select e.sal
  from emp e, emp d
 where e.deptno = d.deptno
   and e.deptno = 20
 group by e.sal
having sum(case when e.sal = d.sal then 1 else 0 end)
                           >= abs(sum(sign(e.sal - d.sal)))
       )
```

Oracle

Используйте функции MEDIAN (Oracle10*g* Database) или PERCENTILE_CONT (Oracle 9*i* Database):

```
select median(sal)
  from emp
 where deptno=20
```

```
select percentile_cont(0.5)
       within group(order by sal)
  from emp
 where deptno=20
```

Для Oracle 8*i* Database используйте решение для DB2. Для версий ранее Oracle 8*i* Database можно применять решение PostgreSQL/MySQL.

SQL Server

Для поиска медианы используйте ранжирующие функции COUNT(*) OVER и ROW_NUMBER:

```
 1  select avg(sal)
 2    from (
 3  select sal,
 4         count(*)over() total,
 5         cast(count(*)over() as decimal)/2 mid,
 6         ceiling(cast(count(*)over() as decimal)/2) next,
 7         row_number()over(order by sal) rn
 8    from emp
 9   where deptno = 20
10         ) x
11   where ( total%2 = 0
12           and rn in ( mid, mid+1 )
13         )
14      or ( total%2 = 1
15           and rn = next
16         )
```

Обсуждение

DB2 и SQL Server

Решения для DB2 и SQL Server лишь немного отличаются синтаксисом: SQL Server использует для вычисления остатка от деления оператор «%», а DB2 – функцию MOD. Во всем остальном решения аналогичны. Вложенное представление X возвращает три разных счетчика, TOTAL, MID и NEXT, вместе с номером строки (RN), генерируемым функцией ROW_NUMBER. Эти дополнительные столбцы помогают в поиске медианы. Рассмотрим результирующее множество вложенного представления X, чтобы понять, что находится в этих столбцах:

```
select sal,
       count(*)over() total,
       cast(count(*)over() as decimal)/2 mid,
       ceil(cast(count(*)over() as decimal)/2) next,
       row_number()over(order by sal) rn
  from emp
 where deptno = 20

SAL TOTAL  MID NEXT   RN
---- ----- ---- ---- ----
 800     5  2.5    3    1
1100     5  2.5    3    2
2975     5  2.5    3    3
3000     5  2.5    3    4
3000     5  2.5    3    5
```

Чтобы найти медиану, значения столбца SAL должны быть упорядочены от наименьшего к наибольшему. Поскольку в 20-м отделе (DEPT-

NO 20) нечетное количество сотрудников, медианой будет просто значение SAL, находящееся в позиции, где значения столбцов RN и NEXT равны (позиция, представляющая наименьшее целое, которое больше частного от деления общего числа служащих на два).

Если в результирующем множестве возвращается нечетное число строк, первая часть предиката WHERE (строки 11–13) не выполняется. Если известно, что количество строк в результирующем множестве всегда будет нечетным, запрос можно упростить до:

```
select avg(sal)
  from (
select sal,
       count(*)over() total,
       ceil(cast(count(*)over() as decimal)/2) next,
       row_number()over(order by sal) rn
  from emp
 where deptno = 20
       ) x
 where rn = next
```

К сожалению, если в результирующем множестве четное количество строк, упрощенное решение не годится. В исходном решении четное количество строк обрабатывается с помощью значений столбца MID. Рассмотрим результаты вложенного представления X для 30-го отдела (DEPTNO 30), в котором шесть служащих:

```
select sal,
       count(*)over() total,
       cast(count(*)over() as decimal)/2 mid,
       ceil(cast(count(*)over() as decimal)/2) next,
       row_number()over(order by sal) rn
  from emp
 where deptno = 30
```

```
SAL TOTAL  MID NEXT   RN
---- ----- ---- ---- ----
 950     6    3    3    1
1250     6    3    3    2
1250     6    3    3    3
1500     6    3    3    4
1600     6    3    3    5
2850     6    3    3    6
```

Поскольку возвращается четное число строк, медиана вычисляется путем нахождения среднего из значений двух строк: строки, где RN равно MID, и строки, где RN равно MID + 1.

MySQL и PostgreSQL

Вычисление медианы начинается с рефлексивного объединения таблицы EMP, в результате которого возвращается декартово произведение всех заработных плат (группировка по столбцу E.SAL предотвратит

возвращение дубликатов). В конструкции HAVING с помощью функции SUM подсчитывается, сколько в столбцах E.SAL и D.SAL равных значений. Если это количество больше или равно числу раз, когда значение E.SAL больше значения D.SAL, значит, данная строка является медианой. Увидеть это можно, переместив SUM в список SELECT:

```
select e.sal,
       sum(case when e.sal=d.sal
                then 1 else 0 end) as cnt1,
       abs(sum(sign(e.sal - d.sal))) as cnt2
  from emp e, emp d
 where e.deptno = d.deptno
   and e.deptno = 20
 group by e.sal
```

```
 SAL CNT1 CNT2
---- ---- ----
 800    1    4
1100    1    2
2975    1    0
3000    4    6
```

Oracle

При работе с Oracle10*g* Database или Oracle 9*i* Database задачу по вычислению медианы можно переложить на плечи функций, предоставляемых Oracle. Для Oracle 8*i* Database можно использовать решение DB2. Для всех других версий используется решение PostgreSQL. Очевидно, что функция MEDIAN вычисляет медиану, тогда как с функцией PERCENTILE_CONT все не так явно, хотя она делает то же самое. Передаваемый в PERCENTILE_CONT аргумент, 0,5 – это процентиль. Конструкция WITHIN GROUP (ORDER BY SAL) определяет множество сортированных строк, с которым будет работать PERCENTILE_CONT (помним, что медиана – это середина множества упорядоченных значений). Возвращаемое значение соответствует заданной процентили в сортированном множестве строк (в данном случае, это 0,5; т. е. середина множества, поскольку граничными значениями являются 0 и 1).

Вычисление доли от целого в процентном выражении

Задача

Требуется определить, какую долю от целого в процентном выражении для определенного столбца составляет та или иная группа значений. Например, стоит задача вычислить, какой процент от всех заработных плат составляют заработные платы служащих 10-го отдела (процентный вклад зарплат DEPTNO 10 в общую сумму заработных плат).

Решение

В общем, вычисление процента от целого в SQL ничем не отличается от того, как это делается на бумаге: делим, затем умножаем. В данном примере требуется найти, какой процент от всех заработных плат таблицы EMP составляют заработные платы служащих 10-го отдела (DEPTNO 10). Для этого просто находим заработные платы служащих 10-го отдела и делим их сумму на общую сумму заработных плат в таблице. В качестве заключительного шага умножаем полученное значение на 100, чтобы представить результат в процентном выражении.

MySQL и PostgreSQL

Делим сумму заработных плат 10-го отдела (DEPTNO 10) на сумму всех заработных плат:

```
1 select (sum(
2          case when deptno = 10 then sal end)/sum(sal)
3         )*100 as pct
4   from emp
```

DB2, Oracle и SQL Server

С помощью вложенного запроса и оконной функции SUM OVER находим суммы всех заработных плат и заработных плат 10-го отдела. Затем во внешнем запросе выполняем деление и умножение:

```
1  select distinct (d10/total)*100 as pct
2    from (
3  select deptno,
4         sum(sal)over() total,
5         sum(sal)over(partition by deptno) d10
6    from emp
7         ) x
8   where deptno=10
```

Обсуждение

MySQL и PostgreSQL

Выражение CASE позволяет без труда выбрать заработные платы служащих 10-го отдела (DEPTNO 10). После этого они суммируются и делятся на сумму всех заработных плат. Агрегатные функции игнорируют значения NULL, поэтому конструкция ELSE в выражении CASE не нужна. Чтобы увидеть, какие именно значения участвуют в делении, выполните запрос без этой операции:

```
select sum(case when deptno = 10 then sal end) as d10,
       sum(sal)
  from emp

D10  SUM(SAL)
---- ---------
8750     29025
```

В зависимости от того, как определен SAL, при делении могут понадобиться явные приведения типов. Например, для DB2, SQL Server и PostgreSQL, если значения SAL хранятся как целые, их можно привести к десятичному типу для получения более точного результата, как показано ниже:

```
select (cast(
         sum(case when deptno = 10 then sal end)
             as decimal)/sum(sal)
        )*100 as pct
  from emp
```

DB2, Oracle и SQL Server

В качестве альтернативы традиционному подходу данное решение для вычисления процента от целого использует оконные функции. Для DB2 и SQL Server, если значения SAL хранятся как целые, перед делением понадобится выполнить приведение типов:

```
select distinct
       cast(d10 as decimal)/total*100 as pct
  from (
select deptno,
       sum(sal)over() total,
       sum(sal)over(partition by deptno) d10
  from emp
       ) x
 where deptno=10
```

Важно помнить, что оконные функции обрабатываются после предиката WHERE. Таким образом, сортировка по DEPTNO не может осуществляться во вложенном запросе X. Рассмотрим результаты вложенного запроса X без и с фильтром по DEPTNO. Сначала без:

```
select deptno,
       sum(sal)over() total,
       sum(sal)over(partition by deptno) d10
  from emp
```

```
DEPTNO     TOTAL       D10
------- --------- ---------
     10     29025      8750
     10     29025      8750
     10     29025      8750
     20     29025     10875
     20     29025     10875
     20     29025     10875
     20     29025     10875
     20     29025     10875
     30     29025      9400
     30     29025      9400
     30     29025      9400
     30     29025      9400
```

```
        30      29025       9400
        30      29025       9400
```

а теперь с фильтром:

```
select deptno,
       sum(sal)over() total,
       sum(sal)over(partition by deptno) d10
  from emp
 where deptno=10

DEPTNO     TOTAL       D10
------ --------- ---------
    10      8750      8750
    10      8750      8750
    10      8750      8750
```

Поскольку оконные функции обрабатываются после предиката WHERE, значение столбца TOTAL представляет сумму всех зарплат только 10-го отдела. Но для решения задачи TOTAL должен представлять сумму всех заработных плат. Вот почему фильтр по DEPTNO должен располагаться вне вложенного запроса X.

Агрегация столбцов, которые могут содержать NULL–значения

Задача

Требуется выполнить агрегацию столбца, но в нем могут содержаться NULL-значения. Хочется сохранить точность, но тревожит то, что агрегатные функции игнорируют NULL-значения. Например, поставлена задача определить среднюю сумму комиссионных для служащих 30-го отдела (DEPTNO 30), но некоторые из них не получают комиссионных (для этих служащих в столбце COMM располагаются NULL-значения). Агрегатные функции игнорируют NULL-значения, поэтому точность результата под вопросом. Хотелось бы каким-то образом учесть NULL-значения при агрегации.

Решение

С помощью функции COALESCE преобразуйте NULL-значения в 0, тогда они будут включены в агрегацию:

```
1  select avg(coalesce(comm,0)) as avg_comm
2    from emp
3   where deptno=30
```

Обсуждение

При работе с агрегатными функциями следует помнить об игнорировании NULL-значений. Рассмотрим результат выполнения решения без функции COALESCE:

```
select avg(comm)
  from emp
 where deptno=30

AVG(COMM)
---------
      550
```

В результате данного запроса получаем, что средняя сумма комиссионных для 30-го отдела (DEPTNO 30) составляет 550, но беглый взгляд на строки:

```
select ename, comm
  from emp
 where deptno=30
order by comm desc

ENAME           COMM
---------- ---------
BLAKE
JAMES
MARTIN          1400
WARD             500
ALLEN            300
TURNER             0
```

показывает, что только четверо из шести служащих могут получать комиссионные. Сумма всех комиссионных 30-го отдела составляет 2200, среднее значение должно вычисляться как 2200/6, а не 2200/4. Исключая функцию COALESCE, мы отвечаем на вопрос «Каково среднее значение комиссионных служащих 30-го отдела, *которые могут получать комиссионные*?», а не на вопрос «Каково среднее значение комиссионных всех служащих 30-го отдела?» При работе с агрегатными функциями не забывайте обрабатывать NULL-значения соответствующим образом.

Вычисление среднего без учета наибольшего и наименьшего значений

Задача

Требуется вычислить среднее, но без учета наибольшего и наименьшего значений, чтобы (будем надеятся) уменьшить асимметрию распределения. Например, необходимо найти среднюю заработную плату служащих, исключая из расчета наибольшую и наименьшую из них.

Решение

MySQL и PostgreSQL

Исключаем наибольшее и наименьшее значения с помощью подзапросов:

```
1  select avg(sal)
2    from emp
3   where sal not in (
4      (select min(sal) from emp),
5      (select max(sal) from emp)
6   )
```

DB2, Oracle и SQL Server

С помощью вложенного запроса с оконными функциями MAX OVER и MIN OVER формируем результирующее множество, из которого можно без труда исключить наибольшее и наименьшее значения:

```
1  select avg(sal)
2    from (
3  select sal, min(sal)over() min_sal, max(sal)over() max_sal
4    from emp
5         ) x
6   where sal not in (min_sal,max_sal)
```

Обсуждение

MySQL и PostgreSQL

Подзапросы возвращают наибольшую и наименьшую заработные платы в таблице. Применяя к возвращенным значениям оператор NOT IN, мы исключаем их из вычисления среднего. Помните, что в определении среднего не будут участвовать и все дубликаты (если наибольшую или наименьшую заработную плату получают несколько служащих). Если целью является исключить только по одному экземпляру предельных значений, они просто вычитаются из SUM, и потом осуществляется деление:

```
select (sum(sal)-min(sal)-max(sal))/(count(*)-2)
  from emp
```

DB2, Oracle и SQL Server

Вложенный запрос X возвращает все заработные платы, а также наибольшую и наименьшую из них:

```
select sal, min(sal)over() min_sal, max(sal)over() max_sal
  from emp
```

```
       SAL   MIN_SAL   MAX_SAL
---------- --------- ---------
       800       800      5000
      1600       800      5000
      1250       800      5000
      2975       800      5000
      1250       800      5000
      2850       800      5000
      2450       800      5000
      3000       800      5000
```

```
5000        800        5000
1500        800        5000
1100        800        5000
 950        800        5000
3000        800        5000
1300        800        5000
```

Обратиться к наибольшей и наименьшей заработной плате можно в любой строке, поэтому доступ к ним не составляет труда. Внешний запрос фильтрует строки, возвращенные вложенным запросом X, так, чтобы все заработные платы, соответствующие MIN_SAL или MAX_SAL, были исключены из вычисления среднего.

Преобразование буквенно-цифровых строк в числа

Задача

Имеются буквенно-цифровые данные, и требуется выбрать из них только числа. Стоит задача получить число 123321 из строки «paul123f321».

Решение

DB2

Для извлечения числовых символов из буквенно-цифровой строки используются функции TRANSLATE и REPLACE:

```
1 select cast(
2        replace(
3      translate( 'paul123f321',
4                 repeat('#',26),
5                 'abcdefghijklmnopqrstuvwxyz'),'#','')
6        as integer ) as num
7   from t1
```

Oracle и PostgreSQL

Для извлечения числовых символов из буквенно-цифровой строки используются функции TRANSLATE и REPLACE:

```
1 select cast(
2        replace(
3      translate( 'paul123f321',
4                 'abcdefghijklmnopqrstuvwxyz',
5                 rpad('#',26,'#')),'#','')
6        as integer ) as num
7  from t1
```

MySQL and SQL Server

Решение не предоставляется, поскольку на момент написания данной книги ни один из этих производителей не поддерживает функцию TRANSLATE.

Обсуждение

Два решения отличаются только синтаксисом: для DB2 используется функция REPEAT, а не RPAD, и порядок параметров в списке функции TRANSLATE разный. Данное обсуждение ориентируется на пример решения для Oracle/PostgreSQL, но оно правомочно и для DB2. Если выполнить запрос «с изнанки» (начиная с TRANSLATE), станет видно, что все очень просто. Сначала TRANSLATE заменяет все нечисловые символы символом «#»:

```
select translate( 'paul123f321',
                  'abcdefghijklmnopqrstuvwxyz',
                  rpad('#',26,'#')) as num
  from t1

NUM
-----------
####123#321
```

Поскольку все нечисловые символы теперь представлены символом «#», просто удаляем их с помощью функции REPLACE. Затем приводим результат к числовому типу. Этот конкретный пример предельно прост, потому что представлены только буквенно-цифровые данные. Если могут храниться и другие символы, проще не вылавливать их, а подойти к решению этой задачи по-другому: вместо того, чтобы искать и удалять нечисловые символы, выбирайте все числовые символы и удаляйте все остальные. Следующий пример поясняет такой подход:

```
select replace(
     translate('paul123f321',
       replace(translate( 'paul123f321',
                          '0123456789',
                          rpad('#',10,'#')),'#',''),
               rpad('#',length('paul123f321'),'#')),'#','') as num
  from t1

NUM
------
123321
```

Это решение кажется немного более запутанным, чем первоначальное, но оно совсем не выглядит таковым, если разложить его на составляющие. Рассмотрим самый глубоко вложенный вызов TRANSLATE:

```
select translate( 'paul123f321',
                  '0123456789',
                  rpad('#',10,'#'))
  from t1

TRANSLATE('
-----------
paul###f###
```

Итак, исходный подход иной: вместо замены символом «#» всех нечисловых символов заменяем им все числовые символы. Следующий шаг – удаляем все экземпляры «#», таким образом оставляя только нечисловые символы:

```
select replace(translate( 'paul123f321',
                          '0123456789',
                          rpad('#',10,'#')),'#','')
  from t1

REPLA
-----
paulf
```

Далее опять вызывается TRANSLATE; на этот раз, чтобы заменить в исходной строке все нечисловые символы (возвращенные предыдущим запросом) экземпляром «#»:

```
select translate('paul123f321',
       replace(translate( 'paul123f321',
                          '0123456789',
                          rpad('#',10,'#')),'#',''),
               rpad('#',length('paul123f321'),'#'))
  from t1

TRANSLATE('
-----------
####123#321
```

Здесь остановимся и рассмотрим самый внешний вызов TRANSLATE. Второй параметр функции RPAD (или второй параметр функции REPEAT для DB2) – длина исходной строки. Этой величиной удобно пользоваться, потому что ни один символ не может повторяться количество раз, превышающее длину строки, частью которой он является. Теперь, когда мы заменили все нечисловые символы экземпляром «#», необходимо удалить все «#» с помощью функции REPLACE. В итоге остаются одни числа.

Изменение значений в текущей сумме

Задача

Требуется изменять значения текущей суммы в зависимости от значений другого столбца. Рассмотрим сценарий, в котором после каждой транзакции должны быть представлены история транзакций кредитной карты и текущий баланс. В этом примере будет использоваться следующее представление V:

```
create view V (id,amt,trx)
as
select 1, 100, 'PR' from t1 union all
select 2, 100, 'PR' from t1 union all
```

```
select 3, 50,  'PY' from t1 union all
select 4, 100, 'PR' from t1 union all
select 5, 200, 'PY' from t1 union all
select 6, 50,  'PY' from t1
```

```
select * from V
```

```
ID        AMT TRX
-- ---------- ---
 1        100 PR
 2        100 PR
 3         50 PY
 4        100 PR
 5        200 PY
 6         50 PY
```

Значения столбца ID уникально идентифицируют каждую транзакцию. Столбец AMT представляет размер денежных средств, участвующих в каждой транзакции (будь то покупка или платеж). Столбец TRX содержит информацию о типе транзакции: платеж обозначается «PY», покупка – «PR». Если в столбце TRX находится значение PY, текущее значение столбца AMT должно вычитаться из накопленной суммы. Если значение столбца – PR, текущее значение AMT добавляется к текущей сумме. В конечном счете требуется получить следующее результирующее множество:

```
TRX_TYPE        AMT    BALANCE
-------- ---------- ----------
PURCHASE        100        100
PURCHASE        100        200
PAYMENT          50        150
PURCHASE        100        250
PAYMENT         200         50
PAYMENT          50          0
```

Решение

DB2 и Oracle

Для вычисления текущей суммы используйте оконную функцию SUM OVER вместе с выражением CASE для определения типа транзакции:

```
 1  select case when trx = 'PY'
 2              then 'PAYMENT'
 3              else 'PURCHASE'
 4          end trx_type,
 5          amt,
 6          sum(
 7           case when trx = 'PY'
 8              then -amt else amt
 9           end
10         ) over (order by id,amt) as balance
11    from V
```

MySQL, PostgreSQL и SQL Server

Для вычисления текущей суммы используйте скалярный подзапрос вместе с выражением CASE для определения типа транзакции:

```
 1  select case when v1.trx = 'PY'
 2              then 'PAYMENT'
 3              else 'PURCHASE'
 4         end as trx_type,
 5         v1.amt,
 6         (select sum(
 7                  case when v2.trx = 'PY'
 8                       then -v2.amt else v2.amt
 9                  end
10                 )
11            from V v2
12           where v2.id <= v1.id) as balance
13    from V v1
```

Обсуждение

Выражение CASE определяет, как следует поступить с текущим значением AMT: добавить его или вычесть из текущей суммы. Если транзакция является платежом, то значение AMT преобразуется в отрицательное, таким образом, при суммировании текущая сумма уменьшается. Результат выполнения выражения CASE представлен ниже:

```
select case when trx = 'PY'
            then 'PAYMENT'
            else 'PURCHASE'
       end trx_type,
       case when trx = 'PY'
            then -amt else amt
       end as amt
  from V
```

```
TRX_TYPE       AMT
-------- ---------
PURCHASE       100
PURCHASE       100
PAYMENT        -50
PURCHASE       100
PAYMENT       -200
PAYMENT        -50
```

Когда тип транзакции известен, значения AMT добавляются или вычитаются из текущей суммы. О том, как оконная функция SUM OVER или скалярный подзапрос формируют текущую сумму, смотрите в рецепте «Вычисление текущей суммы».

8

Арифметика дат

В данной главе представлены методики выполнения простых операций с датами. Рецепты охватывают самые распространенные задачи, такие как добавление дней к датам, вычисление количества рабочих дней между датами и нахождение разности между датами в днях.

Умелое обращение со встроенными функциями используемой СУБД при работе с датами может существенно повысить вашу производительность. Во всех рецептах данной главы я пытался задействовать встроенные функции каждой СУБД. К тому же везде используется один формат даты, «DD-MON-YYYY». Я выбрал так, поскольку, полагаю, это будет удобно для тех, кто работает с одной СУБД и хочет изучить другие. Работа с одним стандартным форматом поможет сосредоточиться на разных техниках и функциях, предоставляемых каждой СУБД, и позволит не беспокоиться при этом о применяемых по умолчанию форматах дат.

Данная глава посвящена основам арифметики дат. Более сложные рецепты работы с датами вынесены в следующую главу. Рецепты, представленные здесь, используют простые типы дат. В случае использования более сложных типов дат решения должны быть скорректированы соответствующим образом.

Добавление и вычитание дней, месяцев и лет

Задача

Требуется добавить или вычесть некоторое количество дней, месяцев или лет из даты. Например, используя значение столбца HIREDATE (дата приема на работу) для служащего CLARK, необходимо получить шесть разных дат: пять дней до и после приема CLARK на работу, пять месяцев до и после приема CLARK на работу и, наконец, пять лет до

и после приема CLARK на работу. CLARK был нанят «09-JUN-1981», таким образом, должно быть получено следующее результирующее множество:

```
HD_MINUS_5D HD_PLUS_5D  HD_MINUS_5M HD_PLUS_5M  HD_MINUS_5Y HD_PLUS_5Y
----------- ----------- ----------- ----------- ----------- -----------
04-JUN-1981 14-JUN-1981 09-JAN-1981 09-NOV-1981 09-JUN-1976 09-JUN-1986
12-NOV-1981 22-NOV-1981 17-JUN-1981 17-APR-1982 17-NOV-1976 17-NOV-1986
18-JAN-1982 28-JAN-1982 23-AUG-1981 23-JUN-1982 23-JAN-1977 23-JAN-1987
```

Решение

DB2

При работе с данными допускается использование стандартных сложения и вычитания, но любое значение, добавляемое или вычитаемое из даты, должно сопровождаться единицей времени, которую оно представляет:

```
1 select hiredate -5 day   as hd_minus_5D,
2        hiredate +5 day   as hd_plus_5D,
3        hiredate -5 month as hd_minus_5M,
4        hiredate +5 month as hd_plus_5M,
5        hiredate -5 year  as hd_minus_5Y,
6        hiredate +5 year  as hd_plus_5Y
7   from emp
8  where deptno = 10
```

Oracle

Для вычисления дней используйте стандартные сложение и вычитание, а для операций над месяцами и годами – функцию ADD_MONTHS (добавить месяцы):

```
1 select hiredate-5                 as hd_minus_5D,
2        hiredate+5                 as hd_plus_5D,
3        add_months(hiredate,-5)    as hd_minus_5M,
4        add_months(hiredate,5)     as hd_plus_5M,
5        add_months(hiredate,-5*12) as hd_minus_5Y,
6        add_months(hiredate,5*12)  as hd_plus_5Y
7   from emp
8  where deptno = 10
```

PostgreSQL

Используйте стандартные сложение и вычитание с ключевым словом INTERVAL (интервал) с указанием периода, который нужно добавить или вычесть, и его единицы времени. Значения INTERVAL обязательно должны быть заключены в одинарные кавычки:

```
1 select hiredate - interval '5 day'   as hd_minus_5D,
2        hiredate + interval '5 day'   as hd_plus_5D,
3        hiredate - interval '5 month' as hd_minus_5M,
```

```
4        hiredate + interval '5 month' as hd_plus_5M,
5        hiredate - interval '5 year'  as hd_minus_5Y,
6        hiredate + interval '5 year'  as hd_plus_5Y
7   from emp
8  where deptno=10
```

MySQL

Используйте стандартные сложение и вычитание с ключевым словом INTERVAL (интервал) с указанием периода, который нужно добавить или вычесть, и его единицы времени. В отличие от решения PostgreSQL значения INTERVAL не заключаются в одинарные кавычки:

```
1 select hiredate - interval 5 day   as hd_minus_5D,
2        hiredate + interval 5 day   as hd_plus_5D,
3        hiredate - interval 5 month as hd_minus_5M,
4        hiredate + interval 5 month as hd_plus_5M,
5        hiredate - interval 5 year  as hd_minus_5Y,
6        hiredate + interval 5 year  as hd_plus_5Y
7   from emp
8  where deptno=10
```

В качестве альтернативы можно использовать функцию DATE_ADD, показанную ниже:

```
1 select date_add(hiredate,interval -5 day)   as hd_minus_5D,
2        date_add(hiredate,interval  5 day)   as hd_plus_5D,
3        date_add(hiredate,interval -5 month) as hd_minus_5M,
4        date_add(hiredate,interval  5 month) as hd_plus_5M,
5        date_add(hiredate,interval -5 year)  as hd_minus_5Y,
6        date_add(hiredate,interval  5 year)  as hd_plus_5DY
7  from emp
8  where deptno=10
```

SQL Server

Для добавления или вычитания различных единиц времени к/из даты используйте функцию DATEADD:

```
1 select dateadd(day,-5,hiredate)   as hd_minus_5D,
2        dateadd(day,5,hiredate)    as hd_plus_5D,
3        dateadd(month,-5,hiredate) as hd_minus_5M,
4        dateadd(month,5,hiredate)  as hd_plus_5M,
5        dateadd(year,-5,hiredate)  as hd_minus_5Y,
6        dateadd(year,5,hiredate)   as hd_plus_5Y
7   from emp
8  where deptno = 10
```

Обсуждение

Решение для Oracle при работе с датами использует то преимущество, что они хранятся как целое количество дней с определенной даты. Однако это верно только для операций с типом DATE (хранит только дату).

В Oracle 9*i* Database введен тип TIMESTAMP (хранит и дату, и время). Для данных такого типа следует использовать решение с INTERVAL, приведенное для PostgreSQL. Кроме того, остерегайтесь передавать данные типа TIMESTAMP в старые функции работы с датами, такие как ADD_MONTHS. При этом будет утеряна вся информация о дробных частях секунды, которая может храниться в значениях типа TIMESTAMP.

Ключевое слово INTERVAL и сопровождающие его строковые литералы представляют соответствующий стандарту ISO синтаксис SQL. Стандарт требует, чтобы значения интервала были заключены в одинарные кавычки. PostgreSQL (а также Oracle 9*i* Database и более поздние версии) соблюдает стандарт. MySQL несколько отклоняется от него и не поддерживает кавычки.

Определение количества дней между двумя датами

Задача

Требуется найти разность между двумя датами и представить результат в днях. Например, стоит задача вычислить разницу в днях между датами приема на работу (значения столбца HIREDATE) служащих ALLEN и WARD.

Решение

DB2

Используйте два вложенных запроса, чтобы найти значения HIREDATE для WARD и ALLEN. Затем найдите разность этих значений с помощью функции DAYS (дни):

```
1  select days(ward_hd) - days(allen_hd)
2    from (
3  select hiredate as ward_hd
4    from emp
5   where ename = 'WARD'
6         ) x,
7         (
8  select hiredate as allen_hd
9    from emp
10   where ename = 'ALLEN'
11        ) y
```

Oracle и PostgreSQL

Используйте два вложенных запроса, чтобы найти значения HIREDATE для WARD и ALLEN, и затем вычтите одну дату из другой:

```
1  select ward_hd - allen_hd
2    from (
3  select hiredate as ward_hd
```

```
 4    from emp
 5   where ename = 'WARD'
 6         ) x,
 7         (
 8  select hiredate as allen_hd
 9    from emp
10   where ename = 'ALLEN'
11         ) y
```

MySQL и SQL Server

Для определения количества дней между двумя датами используйте функцию DATEDIFF. В версии DATEDIFF для MySQL только два обязательных параметра (две даты, разницу между которыми необходимо найти), причем меньшая из двух дат должна быть передана в функцию первой (в SQL Server наоборот), чтобы избежать получения отрицательных значений. Версия этой функции для SQL Server позволяет задавать единицы измерения возвращаемого значения (в данном примере требуется получить разность в днях). В следующем решении используется SQL Server версия функции:

```
 1  select datediff(day,allen_hd,ward_hd)
 2    from (
 3  select hiredate as ward_hd
 4    from emp
 5   where ename = 'WARD'
 6         ) x,
 7         (
 8  select hiredate as allen_hd
 9    from emp
10   where ename = 'ALLEN'
11         ) y
```

Пользователи MySQL могут просто убрать первый аргумент функции DATEDIFF и поменять местами значения ALLEN_HD и WARD_HD.

Обсуждение

Во всех решениях вложенные запросы X и Y возвращают значения HIREDATE для служащих WARD и ALLEN соответственно. Например:

```
select ward_hd, allen_hd
    from (
select hiredate as ward_hd
  from emp
 where ename = 'WARD'
       ) y,
       (
select hiredate as allen_hd
  from emp
 where ename = 'ALLEN'
       ) x
```

```
WARD_HD     ALLEN_HD
----------- ---------
22-FEB-1981 20-FEB-1981
```

Как видите, формируется декартово произведение, поскольку объединение X и Y не задано. В данном случае отсутствие объединения не создает никакой опасности, поскольку кардинальности X и Y равны 1, и, таким образом, результирующее множество будет иметь одну строку (это очевидно, потому что 1×1=1). Чтобы получить разность в днях, просто вычитаем одно из возвращенных значений из другого с помощью метода, подходящего для используемой базы данных.

Определение количества рабочих дней между двумя датами

Задача

Даны две даты, требуется найти, сколько «рабочих» дней между ними, включая эти даты. Например, если 10 января – вторник, а 11 января – среда, то между этими двумя датами два рабочих дня, поскольку оба дня являются обычными рабочими днями. В данном рецепте «рабочими» днями считаются все дни, кроме субботы и воскресенья.

Решение

В примерах решения определяется число рабочих дней между датами приема на работу (значениями столбца HIREDATE) служащих BLAKE и JONES. Для поиска, сколько рабочих дней выпадает между двумя датами, можно использовать сводную таблицу, каждая строка которой будет соответствовать каждому отдельному дню между двумя датами (включая их самих). После создания такой таблицы определение количества рабочих дней заключается в простом подсчете возвращенных дат, не являющихся субботой или воскресеньем.

Если необходимо исключить и праздничные дни, то можно создать таблицу HOLIDAYS (праздники), а затем включить в запрос простой предикат NOT IN для исключения из результата дней, перечисленых в таблице HOLIDAYS.

DB2

Для формирования необходимого числа строк, представляющих дни между заданными датами, используйте сводную таблицу T500. Затем подсчитайте количество дней, не являющихся выходными. Воспользуйтесь функцией DAYNAME (название дня) для получения названия дня недели для каждой даты. Например:

```
1  select sum(case when dayname(jones_hd+t500.id day -1 day)
2                    in ( 'Saturday','Sunday' )
3                   then 0 else 1
```

```
            end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
            end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
            end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

MySQL

Для формирования необходимого числа строк (дней) между двумя датами используйте сводную таблицу T500. Затем подсчитайте количество дней, не являющихся выходными. Добавление дней к каждой дате реализуйте с помощью функции DATE_ADD. Воспользуйтесь функцией DATE_FORMAT (формат даты) для получения названия дня недели для каждой даты:

```
select sum(case when date_format(
                        date_add(jones_hd,
                                 interval t500.id-1 DAY),'%a')
                  in ( 'Sat','Sun' )
                then 0 else 1
            end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
            end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
            end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= datediff(blake_hd,jones_hd)+1
```

Oracle

Для формирования необходимого числа строк (дней) между двумя датами используйте сводную таблицу T500, и затем подсчитайте количество дней, не являющихся выходными. Используйте функцию TO_CHAR для получения названия дня недели для каждой даты:

```
select sum(case when to_char(jones_hd+t500.id-1,'DY')
                  in ( 'SAT','SUN' )
                then 0 else 1
```

```
           end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

PostgreSQL

Для формирования необходимого числа строк (дней) между двумя датами используйте сводную таблицу T500. Затем подсчитайте количество дней, не являющихся выходными. Используйте функцию TO_CHAR для получения названия дня недели для каждой даты:

```
select sum(case when trim(to_char(jones_hd+t500.id-1,'DAY'))
                  in ( 'SATURDAY','SUNDAY' )
                then 0 else 1
           end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

SQL Server

Для формирования необходимого числа строк (дней) между двумя датами используйте сводную таблицу T500, и затем подсчитайте количество дней, не являющихся выходными. Используйте функцию DATENAME (имя даты) для получения названия дня недели для каждой даты:

```
select sum(case when datename(dw,jones_hd+t500.id-1)
                  in ( 'SATURDAY','SUNDAY' )
                 then 0 else 1
           end) as days
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
```

```
 9         max(case when ename = 'JONES'
10                  then hiredate
11             end) as jones_hd
12    from emp
13   where ename in ( 'BLAKE','JONES' )
14         ) x,
15         t500
16   where t500.id <= datediff(day,jones_hd-blake_hd)+1
```

Обсуждение

Хотя в различных СУБД используются разные встроенные функции для определения названия дня недели, общий подход к решению одинаков для всех СУБД. Решение можно разбить на два этапа:

1. Возвращаем дни между начальной и конечной датами (включая их).
2. Подсчитываем количество дней (т. е. строк), исключая выходные.

Вложенный запрос X реализует первый этап. В нем можно заметить агрегатную функцию MAX, которая используется в рецепте для удаления значений NULL. Разобраться в работе функции MAX поможет следующий запрос. Он показывает результаты, возвращаемые вложенным запросом X без функции MAX:

```
select case when ename = 'BLAKE'
            then hiredate
       end as blake_hd,
       case when ename = 'JONES'
            then hiredate
       end as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
```

```
BLAKE_HD    JONES_HD
----------- -----------
            02-APR-1981
01-MAY-1981
```

Без MAX возвращаются две строки. При использовании функции MAX будет возвращена только одна строка, а не две, причем без значений NULL:

```
select max(case when ename = 'BLAKE'
            then hiredate
       end) as blake_hd,
       max(case when ename = 'JONES'
            then hiredate
       end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
```

```
BLAKE_HD    JONES_HD
----------- -----------
01-MAY-1981 02-APR-1981
```

Между заданными двумя датами (включая их) 30 дней. Расположив эти две даты в одной строке, можно перейти к формированию строк, каждая из которых будет соответствовать одному дню этого временного промежутка (всего 30 дней/строк). Для этого используем таблицу T500. Поскольку каждое значение столбца ID таблицы T500 просто на 1 больше предыдущего, то для получения последовательности дней (начиная от JONES_HD и до BLAKE_HD включительно) каждую строку, возвращаемую T500, добавляем к более ранней из двух дат (JONES_HD). Результат показан ниже (с использованием синтаксиса Oracle):

```
select x.*, t500.*, jones_hd+t500.id-1
  from (
select max(case when ename = 'BLAKE'
                then hiredate
           end) as blake_hd,
       max(case when ename = 'JONES'
                then hiredate
           end) as jones_hd
  from emp
 where ename in ( 'BLAKE','JONES' )
       ) x,
       t500
 where t500.id <= blake_hd-jones_hd+1
```

```
BLAKE_HD    JONES_HD             ID JONES_HD+T5
----------- ----------- ---------- -----------
01-MAY-1981 02-APR-1981          1 02-APR-1981
01-MAY-1981 02-APR-1981          2 03-APR-1981
01-MAY-1981 02-APR-1981          3 04-APR-1981
01-MAY-1981 02-APR-1981          4 05-APR-1981
01-MAY-1981 02-APR-1981          5 06-APR-1981
01-MAY-1981 02-APR-1981          6 07-APR-1981
01-MAY-1981 02-APR-1981          7 08-APR-1981
01-MAY-1981 02-APR-1981          8 09-APR-1981
01-MAY-1981 02-APR-1981          9 10-APR-1981
01-MAY-1981 02-APR-1981         10 11-APR-1981
01-MAY-1981 02-APR-1981         11 12-APR-1981
01-MAY-1981 02-APR-1981         12 13-APR-1981
01-MAY-1981 02-APR-1981         13 14-APR-1981
01-MAY-1981 02-APR-1981         14 15-APR-1981
01-MAY-1981 02-APR-1981         15 16-APR-1981
01-MAY-1981 02-APR-1981         16 17-APR-1981
01-MAY-1981 02-APR-1981         17 18-APR-1981
01-MAY-1981 02-APR-1981         18 19-APR-1981
01-MAY-1981 02-APR-1981         19 20-APR-1981
01-MAY-1981 02-APR-1981         20 21-APR-1981
01-MAY-1981 02-APR-1981         21 22-APR-1981
01-MAY-1981 02-APR-1981         22 23-APR-1981
01-MAY-1981 02-APR-1981         23 24-APR-1981
01-MAY-1981 02-APR-1981         24 25-APR-1981
01-MAY-1981 02-APR-1981         25 26-APR-1981
01-MAY-1981 02-APR-1981         26 27-APR-1981
```

```
01-MAY-1981 02-APR-1981          27 28-APR-1981
01-MAY-1981 02-APR-1981          28 29-APR-1981
01-MAY-1981 02-APR-1981          29 30-APR-1981
01-MAY-1981 02-APR-1981          30 01-MAY-1981
```

Проанализировав предикат WHERE, можно заметить, что для получения требуемых 30 строк мы добавляем 1 к разности между значениями BLAKE_HD и JONES_HD (в противном случае получилось бы 29 строк). Кроме того, видим, что в списке SELECT внешнего запроса из T500.ID вычитается 1, поскольку значения ID начинаются с 1, и добавление 1 к значению JONES_HD привело бы к тому, что оно не было бы включено в окончательный результат.

Сгенерировав строки, необходимые для формирования результирующего множества, с помощью выражения CASE отмечаем «1» строки, соответствующие рабочим дням, а «0» – выходные дни. Заключительный шаг – используя агрегатную функцию SUM, подсчитать количество «1» и получить окончательный ответ.

Определение количества месяцев или лет между двумя датами

Задача

Требуется найти разность между двумя датами, в месяцах или годах. Например, поставлена задача определить количество месяцев, прошедших между приемом на работу первого и последнего служащих, а также представить это значение в годах.

Решение

Поскольку в году всегда 12 месяцев, то чтобы получить результат в годах, можно найти количество месяцев между двумя датами и разделить это значение на 12. Определившись с решением, мы захотим округлить результаты с повышением или с понижением, в зависимости от предъявляемых требований. Например, первое значение столбца HIREDATE таблицы EMP – «17-DEC-1980», последнее – «12-JAN-1983». Если проводить вычисления по годам (1983 минус 1980), то получится три года, а разница в месяцах составит примерно 25 (немного больше двух лет). Решение должно отвечать поставленным требованиям. Приведенные ниже решения обеспечивают результат 25 месяцев и ~2 года.

DB2 и MySQL

С помощью функций YEAR (год) и MONTH (месяц) представьте заданные даты в таком формате: год – четырехзначным числом, месяц – двузначным числом:

```
1  select mnth, mnth/12
2    from (
```

```
3  select (year(max_hd) - year(min_hd))*12 +
4         (month(max_hd) - month(min_hd)) as mnth
5    from (
6  select min(hiredate) as min_hd, max(hiredate) as max_hd
7    from emp
8         ) x
9         ) y
```

Oracle

Чтобы найти разницу между двумя датами в месяцах, используйте функцию MONTHS_BETWEEN (месяцев между) (чтобы выразить результат в годах, просто разделите полученное значение на 12):

```
1  select months_between(max_hd,min_hd),
2         months_between(max_hd,min_hd)/12
3    from (
4  select min(hiredate) min_hd, max(hiredate) max_hd
5    from emp
6         ) x
```

PostgreSQL

С помощью функции EXTRACT (извлечь) представьте заданные даты в таком формате: год – четырехзначным числом, месяц – двузначным числом:

```
 1  select mnth, mnth/12
 2    from (
 3  select ( extract(year from max_hd) -
 4           extract(year from min_hd) ) * 12
 5         +
 6         ( extract(month from max_hd) -
 7           extract(month from min_hd) ) as mnth
 8    from (
 9  select min(hiredate) as min_hd, max(hiredate) as max_hd
10    from emp
11         ) x
12         ) y
```

SQL Server

Чтобы найти разницу между двумя датами в месяцах, используйте функцию DATEDIFF (чтобы выразить результат в годах, просто разделите полученное значение на 12):

```
1  select datediff(month,min_hd,max_hd),
2         datediff(month,min_hd,max_hd)/12
3    from (
4  select min(hiredate) min_hd, max(hiredate) max_hd
5    from emp
6         ) x
```

Обсуждение

DB2, MySQL и PostgreSQL

После того как в решении PostgreSQL из значений MIN_HD и MAX_HD извлечены соответствующие им годы и месяцы, методика определения количества месяцев и лет между датами MIN_HD и MAX_ HD для этих трех СУБД одинакова. Данное обсуждение охватывает все три решения. Вложенный запрос X возвращает граничные значения столбца HIREDATE таблицы EMP:

```
select min(hiredate) as min_hd,
       max(hiredate) as max_hd
  from emp
```

```
MIN_HD      MAX_HD
----------- -----------
17-DEC-1980 12-JAN-1983
```

Чтобы найти количество месяцев, прошедшее между датами в столбцах MAX_HD и MIN_HD, умножаем разницу между этими двумя датами в годах на 12 и добавляем разницу между ними в месяцах. Если не совсем понятно, как это работает, разложим каждую дату на составляющие. Числовые значения лет и месяцев показаны ниже:

```
select year(max_hd)  as max_yr,   year(min_hd) as min_yr,
       month(max_hd) as max_mon,  month(min_hd) as min_mon
  from (
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp
       ) x
```

```
MAX_YR     MIN_YR    MAX_MON    MIN_MON
------ ---------- ---------- ----------
  1983       1980          1         12
```

Имея перед глазами эти результаты, найти количество месяцев между датами MAX_HD и MIN_HD не составляет труда: (1983–1980)*12 + (1–12). Чтобы вычислить, сколько лет прошло между этими датами, делим количество месяцев на 12. Округления выполняются соответственно предъявляемым требованиям.

Oracle и SQL Server[1]

Вложенный запрос X возвращает граничные значения столбца HIREDATE таблицы EMP:

```
select min(hiredate) as min_hd, max(hiredate) as max_hd
  from emp
```

[1] В SQL Server для вычисления разницы в годах можно также использовать функцию DATEDIFF с указанием типа периода YEAR. Но такое решение приведет к получению значения 3, поскольку округление происходит вверх. – *Примеч. науч. ред.*

```
MIN_HD      MAX_HD
----------- -----------
17-DEC-1980 12-JAN-1983
```

Функции, поставляемые Oracle и SQL Server (MONTHS_BETWEEN и DATEDIFF соответственно), возвратят количество месяцев между двумя заданными датами. Чтобы выразить эту разницу в годах, делим результат на 12.

Определение количества секунд, минут или часов между двумя датами

Задача

Требуется возвратить разницу между двумя датами в секундах. Например, необходимо найти разницу между датами приема на работу (значениями столбца HIREDATE) служащих ALLEN и WARD в секундах, минутах и часах.

Решение[1]

Если можно найти количество дней между двумя датами, следовательно, полученное значение можно выразить в секундах, минутах и часах, поскольку это единицы измерения времени, составляющие день.

DB2

Чтобы найти разницу между значениями ALLEN_HD и WARD_HD в днях, используйте функцию DAYS. Затем посредством умножения выразите это значение в разных единицах времени:

```
 1  select dy*24 hr, dy*24*60 min, dy*24*60*60 sec
 2    from (
 3  select ( days(max(case when ename = 'WARD'
 4                         then hiredate
 5                     end)) -
 6           days(max(case when ename = 'ALLEN'
 7                         then hiredate
 8                     end))
 9         ) as dy
10    from emp
11         ) x
```

[1] Описанное ниже решение вычисляет разницу в секундах с учетом только целых дней. В СУБД, в которых для хранения даты используется тип данных, который хранит еще и время, такое решение является не совсем точным. В MySQL и SQL Server для точного вычисления можно использовать функцию DATEDIFF с аргументами диапазона HOUR, MINUTE, SECOND. – *Примеч. науч. ред.*

MySQL и SQL Server

Чтобы возвратить количество дней между датами ALLEN_HD и WARD_HD, используйте функцию DATEDIFF. Затем посредством умножения выразите это значение в разных единицах времени:

```
select datediff(day,allen_hd,ward_hd)*24 hr,
       datediff(day,allen_hd,ward_hd)*24*60 min,
       datediff(day,allen_hd,ward_hd)*24*60*60 sec
  from (
select max(case when ename = 'WARD'
                 then hiredate
            end) as ward_hd,
       max(case when ename = 'ALLEN'
                then hiredate
            end) as allen_hd
  from emp
       ) x
```

Oracle и PostgreSQL

Чтобы получить число дней между датами приема на работу ALLEN и WARD, найдите разность между значениями ALLEN_HD и WARD_HD. Затем посредством умножения выразите это значение в разных единицах времени:

```
select dy*24 as hr, dy*24*60 as min, dy*24*60*60 as sec
  from (
select (max(case when ename = 'WARD'
                then hiredate
            end) -
       max(case when ename = 'ALLEN'
                then hiredate
            end)) as dy
  from emp
       ) x
```

Обсуждение

Во всех решениях вложенный запрос X возвращает значения HIREDATE для служащих WARD и ALLEN, как показано ниже:

```
select max(case when ename = 'WARD'
                then hiredate
            end) as ward_hd,
       max(case when ename = 'ALLEN'
                then hiredate
            end) as allen_hd
  from emp

WARD_HD     ALLEN_HD
----------- -----------
22-FEB-1981 20-FEB-1981
```

Умножаем количество дней между значениями WARD_HD и ALLEN_HD на 24 (число часов в дне), 1440 (число минут в дне) и 86400 (число секунд в дне).

Как подсчитать, сколько раз в году повторяется каждый день недели

Задача

Требуется подсчитать, сколько раз в году повторяется каждый день недели.

Решение

Чтобы найти, сколько раз повторяется каждый день недели за год, необходимо:

1. Сгенерировать все возможные даты года.
2. Отформатировать даты таким образом, чтобы в них был указан соответствующий день недели.
3. Подсчитать, сколько раз за год встречается название каждого дня недели.

DB2

Используйте рекурсивный WITH, чтобы не делать выборку, состоящую, по меньшей мере, из 366 строк. Чтобы получить название дня недели для каждой даты, используйте функцию DAYNAME и затем подсчитайте количество дней с каждым именем:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (current_date -
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select dayname(start_date),count(*)
  from x
 group by dayname(start_date)
```

MySQL

Сначала сделайте запрос к таблице T500, чтобы получить достаточно строк для выбора всех дней года. Чтобы получить название дня недели для каждой даты, используйте функцию DATE_FORMAT и затем подсчитайте количество дней с каждым именем:

```
select date_format(
           date_add(
               cast(
             concat(year(current_date),'-01-01')
                    as date),
                    interval t500.id-1 day),
                    '%W') day,
        count(*)
   from t500
  where t500.id <= datediff(
                      cast(
                     concat(year(current_date)+1,'-01-01')
                            as date),
                      cast(
                     concat(year(current_date),'-01-01')
                            as date))
group by date_format(
             date_add(
                 cast(
               concat(year(current_date),'-01-01')
                      as date),
                      interval t500.id-1 day),
                      '%W')
```

Oracle

При работе с Oracle 9*i* Database или более поздними версиями для получения всех дней года можно использовать рекурсивный оператор CONNECT BY. В Oracle 8*i* Database и более ранних версиях выполняется запрос к таблице T500, чтобы сгенерировать достаточно строк для возвращения всех дней года. В любом случае название дня недели получаем с помощью функции TO_CHAR и затем подсчитываем количество дней с каждым именем.

Сначала решение с CONNECT BY:

```
with x as (
select level lvl
  from dual
 connect by level <= (
   add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
 )
)
select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
  from x
 group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')
```

и следующее решение для более старых версий Oracle:

```
1 select to_char(trunc(sysdate,'y')+rownum-1,'DAY'),
2        count(*)
3   from t500
4  where rownum <= (add_months(trunc(sysdate,'y'),12)
5                   - trunc(sysdate,'y'))
6  group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')
```

PostgreSQL

Посредством встроенной функции GENERATE_SERIES сформируйте количество строк, соответствующее количеству дней в году. Затем с помощью функции TO_CHAR получите название дня недели для каждой даты. Наконец, подсчитайте количество дней с каждым именем. Например:

```
 1  select to_char(
 2            cast(
 3      date_trunc('year',current_date)
 4                 as date) + gs.id-1,'DAY'),
 5         count(*)
 6    from generate_series(1,366) gs(id)
 7   where gs.id <= (cast
 8                     ( date_trunc('year',current_date) +
 9                          interval '12 month' as date) -
10  cast(date_trunc('year',current_date)
11                          as date))
12   group by to_char(
13               cast(
14         date_trunc('year',current_date)
15            as date) + gs.id-1,'DAY')
```

SQL Server

Используйте рекурсивный WITH, чтобы не делать выборку, состоящую, по меньшей мере, из 366 строк. Для версий SQL Server[1], не поддерживающих оператор WITH, используйте альтернативное решение Oracle как руководство по применению сводной таблицы. Чтобы получить название дня недели для каждой даты, используйте функцию DAYNAME и затем подсчитайте количество дней с каждым именем. Например:

```
1  with x (start_date,end_date)
2  as (
3  select start_date,
4         dateadd(year,1,start_date) end_date
5    from (
6  select cast(
7         cast(year(getdate()) as varchar) + '-01-01'
8              as datetime) start_date
```

[1] Для SQL Server 2000 и более ранних версий. – *Примеч. науч. ред.*

```
 9    from t1
10         ) tmp
11  union all
12  select dateadd(day,1,start_date), end_date
13    from x
14   where dateadd(day,1,start_date) < end_date
15  )
16  select datename(dw,start_date),count(*)
17    from x
18   group by datename(dw,start_date)
19 OPTION (MAXRECURSION 366)
```

Обсуждение

DB2

Вложенный запрос TMP в рекурсивном представлении X возвращает первый день текущего года, как показано ниже:

```
select (current_date -
        dayofyear(current_date) day)
        +1 day as start_date
  from t1

START_DATE
-----------
01-JAN-2005
```

Следующий шаг – добавить один год к значению START_DATE (начальная дата), чтобы иметь в своем распоряжении начальную и конечную даты. Необходимо знать обе граничные даты, потому что мы будем генерировать каждый день года. START_DATE и END_DATE (конечная дата) показаны ниже:

```
select start_date,
       start_date + 1 year end_date
  from (
select (current_date -
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp

START_DATE  END_DATE
----------- -----------
01-JAN-2005 01-JAN-2006
```

Далее рекурсивно добавляем к START_DATE по одному дню до тех пор, пока не получим значение, равное END_DATE. Ниже показаны некоторые строки, возвращенные рекурсивным представлением X:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
```

```
  from (
select (current_date -
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select * from x
```

```
START_DATE  END_DATE
----------- -----------
01-JAN-2005 01-JAN-2006
02-JAN-2005 01-JAN-2006
03-JAN-2005 01-JAN-2006
...
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
...
01-DEC-2005 01-JAN-2006
02-DEC-2005 01-JAN-2006
03-DEC-2005 01-JAN-2006
...
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006
```

Последний шаг – применить к строкам, возвращенным рекурсивным представлением X, функцию DAYNAME и подсчитать количество дней с каждым именем. Окончательный результат приведен ниже:

```
with x (start_date,end_date)
as (
select start_date,
       start_date + 1 year end_date
  from (
select (current_date -
        dayofyear(current_date) day)
        +1 day as start_date
  from t1
       ) tmp
 union all
select start_date + 1 day, end_date
  from x
 where start_date + 1 day < end_date
)
select dayname(start_date),count(*)
  from x
 group by dayname(start_date)
```

```
START_DATE   COUNT(*)
---------  ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

MySQL

В данном решении делаем запрос к таблице T500, чтобы получить количество строк, соответствующее числу дней в году. Команда в строке 4 возвращает первый день текущего года. Она извлекает год из даты, возвращенной функцией CURRENT_DATE (текущая дата), и добавляет к нему месяц и год (следуя стандартному формату дат MySQL). Результат показан ниже:

```
select concat(year(current_date),'-01-01')
  from t1
```

```
START_DATE
-----------
01-JAN-2005
```

Теперь, имея первый день текущего года, с помощью функции DATE_ADD добавляем к нему значения столбца T500.ID, чтобы получить каждый день года. Применяя функцию DATE_FORMAT, возвращаем для каждой даты соответствующий день недели. Чтобы выбрать необходимое число строк из таблицы T500, находим разницу в днях между первым днем текущего года и первым днем следующего года и возвращаем такое же количество строк (их будет 365 или 366). Результаты частично показаны ниже:

```
select date_format(
          date_add(
              cast(
            concat(year(current_date),'-01-01')
                  as date),
                  interval t500.id-1 day),
                  '%W') day
  from t500
 where t500.id <= datediff(
                     cast(
                   concat(year(current_date)+1,'-01-01')
                          as date),
                     cast(
                   concat(year(current_date),'-01-01')
                          as date))
```

```
DAY
-----------
```

```
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

Получив все дни текущего года, подсчитаем, сколько раз в году встречается каждый из дней недели, возвращенных функцией DAYNAME. Окончательный результат приведен ниже:

```
select date_format(
          date_add(
              cast(
            concat(year(current_date),'-01-01')
                  as date),
                  interval t500.id-1 day),
                  '%W') day,
       count(*)
  from t500
 where t500.id <= datediff(
                     cast(
                   concat(year(current_date)+1,'-01-01')
                          as date),
                     cast(
                   concat(year(current_date),'-01-01')
                          as date))
group by date_format(
            date_add(
                cast(
              concat(year(current_date),'-01-01')
                     as date),
                     interval t500.id-1 day),
                     '%W')
```

```
DAY       COUNT(*)
--------- ----------
FRIDAY            52
MONDAY            52
SATURDAY          53
SUNDAY            52
THURSDAY          52
TUESDAY           52
WEDNESDAY         52
```

Oracle

В представленных решениях для формирования строк соответственно количеству дней в году используются или запрос к таблице T500 (сводной таблице), или рекурсивные операторы CONNECT BY и WITH. Вызывая функцию TRUNC, получаем первый день текущего года.

При использовании решения CONNECT BY/WITH генерировать порядковые номера, начиная с 1, можно с помощью псевдостолбца LEVEL (уровень). Чтобы создать необходимое для данного решения число строк, фильтруем столбцы ROWNUM или LEVEL по разнице в днях между первым днем текущего года и первым днем следующего года (365 или 366 дней). Следующий шаг – получить все даты года, добавляя значения ROWNUM или LEVEL к первому дню текущего года. Результаты частично представлены ниже:

```
/* Oracle 9i и последующие версии */
with x as (
select level lvl
  from dual
 connect by level <= (
   add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
 )
)
select trunc(sysdate,'y')+lvl-1
  from x
```

Для решения с использованием сводной таблицы подойдут любая таблица или представление, содержащие, по крайней мере, 366 строк. А поскольку в Oracle есть ROWNUM, нет необходимости в таблице с начинающимися с 1 и последовательно возрастающими значениями. Рассмотрим следующий пример, в котором для получения всех дней текущего года используется сводная таблица T500:

```
/* Oracle 8i и предыдущие версии */
select trunc(sysdate,'y')+rownum-1 start_date
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                  - trunc(sysdate,'y'))
```

```
START_DATE
-----------
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
```

```
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

В любом подходе названия дней недели для каждой даты получаем с помощью функции TO_CHAR и затем подсчитываем количество каждого из дней недели в году. Ниже приведены окончательные результаты:

```
/* Oracle 9i и последующие версии */
with x as (
select level lvl
  from dual
 connect by level <= (
   add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
 )
)
select to_char(trunc(sysdate,'y')+lvl-1,'DAY'), count(*)
  from x
 group by to_char(trunc(sysdate,'y')+lvl-1,'DAY')

/* Oracle 8i и предыдущие версии */
select to_char(trunc(sysdate,'y')+rownum-1,'DAY') start_date,
       count(*)
  from t500
 where rownum <= (add_months(trunc(sysdate,'y'),12)
                  - trunc(sysdate,'y'))
 group by to_char(trunc(sysdate,'y')+rownum-1,'DAY')

START_DATE   COUNT(*)
---------- ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

PostgreSQL

Первый шаг – с помощью функции DATE_TRUNC выбрать год из текущей даты (показана ниже; запрос обращен к таблице T1, поэтому возвращается всего одна строка):

```
select cast(
         date_trunc('year',current_date)
       as date) as start_date
  from t1

START_DATE
-----------
01-JAN-2005
```

Следующий шаг – обратиться к источнику строк (на самом деле, любому табличному выражению) с по крайней мере 366 строками. В качестве источника строк в данном решении используется функция GENERATE_SERIES, но, безусловно, источником может быть и таблица T500. Затем добавляем по дню к первому дню года до тех пор, пока не будут возвращены все дни года (показано ниже):

```
select cast( date_trunc('year',current_date)
               as date) + gs.id-1 as start_date
  from generate_series (1,366) gs(id)
 where gs.id <= (cast
                  ( date_trunc('year',current_date) +
                       interval '12 month' as date) -
     cast(date_trunc('year',current_date)
                     as date))
```

```
START_DATE
-----------
01-JAN-2005
02-JAN-2005
03-JAN-2005
...
29-JAN-2005
30-JAN-2005
31-JAN-2005
...
01-DEC-2005
02-DEC-2005
03-DEC-2005
...
29-DEC-2005
30-DEC-2005
31-DEC-2005
```

Последний шаг – посредством функции TO_CHAR получить день недели для каждой даты и подсчитать количество каждого дня недели в году. Конечные результаты представлены ниже:

```
select to_char(
          cast(
    date_trunc('year',current_date)
               as date) + gs.id-1,'DAY') as start_dates,
       count(*)
  from generate_series(1,366) gs(id)
 where gs.id <= (cast
                  ( date_trunc('year',current_date) +
                       interval '12 month' as date) -
     cast(date_trunc('year',current_date)
                     as date))
 group by to_char(
             cast(
       date_trunc('year',current_date)
          as date) + gs.id-1,'DAY')
```

```
START_DATE   COUNT(*)
---------- ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

SQL Server

Вложенный запрос TMP в рекурсивном представлении X возвращает первый день текущего года:

```
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
            as datetime) start_date
  from t1
```

```
START_DATE
-----------
01-JAN-2005
```

Получив первый день текущего года, добавляем один год к значению START_DATE, чтобы иметь начальную и конечную даты. Необходимо знать обе даты, потому что мы хотим сгенерировать все дни года. START_DATE и END_DATE показаны ниже:

```
select start_date,
        dateadd(year,1,start_date) end_date
  from (
select cast(
       cast(year(getdate()) as varchar) + '-01-01'
            as datetime) start_date
  from t1
       ) tmp
```

```
START_DATE  END_DATE
----------- -----------
01-JAN-2005 01-JAN-2006
```

Далее рекурсивно увеличиваем START_DATE с шагом в один день и останавливаемся на значении, соответствующем дню перед END_DATE. Строки, возвращенные рекурсивным представлением X, частично показаны ниже:

```
with x (start_date,end_date)
 as (
 select start_date,
        dateadd(year,1,start_date) end_date
   from (
 select cast(
        cast(year(getdate()) as varchar) + '-01-01'
```

```
            as datetime) start_date
  from t1
       ) tmp
union all
select dateadd(day,1,start_date), end_date
  from x
 where dateadd(day,1,start_date) < end_date
)
select * from x
OPTION (MAXRECURSION 366)
```

```
START_DATE  END_DATE
----------- -----------
01-JAN-2005 01-JAN-2006
02-JAN-2005 01-JAN-2006
03-JAN-2005 01-JAN-2006
...
29-JAN-2005 01-JAN-2006
30-JAN-2005 01-JAN-2006
31-JAN-2005 01-JAN-2006
...
01-DEC-2005 01-JAN-2006
02-DEC-2005 01-JAN-2006
03-DEC-2005 01-JAN-2006
...
29-DEC-2005 01-JAN-2006
30-DEC-2005 01-JAN-2006
31-DEC-2005 01-JAN-2006
```

Последний шаг – применяем к возвращенным рекурсивным представлением X строкам функцию DATENAME и подсчитываем количество каждого дня недели в году. Окончательные результаты показаны ниже:

```
with x(start_date,end_date)
 as (
 select start_date,
        dateadd(year,1,start_date) end_date
   from (
 select cast(
        cast(year(getdate()) as varchar) + '-01-01'
             as datetime) start_date
   from t1
        ) tmp
 union all
 select dateadd(day,1,start_date), end_date
   from x
  where dateadd(day,1,start_date) < end_date
 )
 select datename(dw,start_date), count(*)
   from x
  group by datename(dw,start_date)
 OPTION (MAXRECURSION 366)
```

```
START_DATE   COUNT(*)
---------  ----------
FRIDAY             52
MONDAY             52
SATURDAY           53
SUNDAY             52
THURSDAY           52
TUESDAY            52
WEDNESDAY          52
```

Определение интервала времени в днях между текущей и следующей записями

Задача

Требуется определить интервал времени в днях между двумя датами (в частности, датами, хранящимися в двух разных строках). Например, для каждого служащего 10-го отдела (DEPTNO 10) необходимо найти, сколько дней прошло между датой его приема на работу и датой приема на работу следующего служащего (может быть из другого отдела).

Решение

Ключ к решению данной задачи – найти ближайшую дату приема на работу (значение HIREDATE) после даты приема на работу текущего служащего. Затем вычислить разницу в днях, используя технику рецепта «Определение количества дней между двумя датами».

DB2

Чтобы найти следующее значение HIREDATE относительно текущего HIREDATE, используйте скалярный подзапрос. Затем с помощью функции DAYS определите интервал между ними в днях:

```
1  select x.*,
2         days(x.next_hd) - days(x.hiredate) diff
3    from (
4  select e.deptno, e.ename, e.hiredate,
5         (select min(d.hiredate) from emp d
6           where d.hiredate > e.hiredate) next_hd
7    from emp e
8   where e.deptno = 10
9         ) x
```

MySQL и SQL Server

Чтобы найти следующее значение HIREDATE относительно текущего HIREDATE, используйте скалярный подзапрос. Затем с помощью функции DATEDIFF вычислите разницу между ними в днях. Ниже показана версия DATEDIFF для SQL Server:

```
select x.*,
       datediff(day,x.hiredate,x.next_hd) diff
  from (
select e.deptno, e.ename, e.hiredate,
       (select min(d.hiredate) from emp d
         where d.hiredate > e.hiredate) next_hd
  from emp e
 where e.deptno = 10
       ) x
```

Пользователи MySQL могут исключить первый аргумент («day») и поменять порядок оставшихся двух аргументов:

```
       datediff(x.next_hd, x.hiredate) diff
```

Oracle

При работе с Oracle 8*i* Database или более новыми версиями для поиска следующего значения HIREDATE относительно текущего используйте оконную функцию LEAD OVER:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate)over(order by hiredate) next_hd
  from emp
       )
 where deptno=10
```

Для Oracle 8 Database и более ранних версий можно использовать в качестве альтернативы решение PostgreSQL.

PostgreSQL

С помощью скалярного подзапроса найдите следующее значение HIREDATE относительно текущего HIREDATE. Затем, чтобы вычислить разницу в днях, используйте простое вычитание:

```
select x.*,
       x.next_hd - x.hiredate as diff
  from (
select e.deptno, e.ename, e.hiredate,
       (select min(d.hiredate) from emp d
         where d.hiredate > e.hiredate) as next_hd
  from emp e
 where e.deptno = 10
       ) x
```

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

Несмотря на различия в синтаксисе, подход во всех этих решениях одинаковый: с помощью скалярного подзапроса находим следующее

относительно текущего значение HIREDATE и затем вычисляем разницу в днях между этими двумя датами, используя технику, описанную ранее в данной главе в разделе «Определение количества дней между двумя датами».

Oracle

Здесь оконная функция LEAD OVER исключительно полезна, поскольку обеспечивает возможность доступа к «последующим» (относительно текущей строки согласно порядку, установленному оператором ORDER BY) строкам. Возможность доступа к близлежащим строкам без дополнительных объединений делает код более понятным и эффективным. При работе с оконными функциями необходимо помнить о том, что они обрабатываются после предиката WHERE, следовательно, в решении должен использоваться вложенный запрос. Если бы во вложенном запросе осуществлялась сортировка по столбцу DEPTNO, результаты были бы другими (рассматривались бы значения HIREDATE только для DEPTNO 10). Важно рассмотреть поведение Oracle-функций LEAD и LAG в присутствии дубликатов. В предисловии я говорил, что код данных рецептов «незащищенный», потому что существует слишком много условий, которые разработчик должен предвидеть, чтобы сохранить целостность и работоспособность своего кода. А иногда, после того как все проблемы учтены, результирующий SQL становится абсолютно нечитабельным. Поэтому целью предлагаемого решения является представить технику, которая может использоваться при создании систем, но должна быть протестирована и многократно скорректирована для работы с реальными данными. Здесь мы обсудим ситуацию, подход к решению которой может быть не так очевиден, особенно для тех, кто работает не в системах Oracle. В данном примере в столбце HIREDATE таблицы EMP нет дублирующихся значений, но такой вариант, безусловно, возможен. Рассмотрим служащих 10-го отдела (DEPTNO 10) и их значения HIREDATE:

```
select ename, hiredate
  from emp
 where deptno=10
 order by 2
```

```
ENAME  HIREDATE
------ -----------
CLARK  09-JUN-1981
KING   17-NOV-1981
MILLER 23-JAN-1982
```

Для данного примера вставим четыре дубликата, так чтобы в таблице было пять служащих (включая KING), которые были приняты на работу 17 ноября:

```
insert into emp (empno,ename,deptno,hiredate)
values (1,'ant',10,to_date('17-NOV-1981'))
```

```
insert into emp (empno,ename,deptno,hiredate)
values (2,'joe',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (3,'jim',10,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,hiredate)
values (4,'choi',10,to_date('17-NOV-1981'))

select ename, hiredate
  from emp
 where deptno=10
 order by 2

ENAME  HIREDATE
------ -----------
CLARK  09-JUN-1981
ant    17-NOV-1981
joe    17-NOV-1981
KING   17-NOV-1981
jim    17-NOV-1981
choi   17-NOV-1981
MILLER 23-JAN-1982
```

Теперь в 10-м отделе несколько служащих, которые были приняты на работу в один день. Если к данному результирующему множеству применить предложенное решение (с переносом фильтра во вложенный запрос, чтобы рассматривались только служащие с DEPTNO 10 и их значения HIREDATE), то будет получено следующее:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate)over(order by hiredate) next_hd
  from emp
 where deptno=10
       )

ENAME  HIREDATE    NEXT_HD           DIFF
------ ----------- ----------- ----------
CLARK  09-JUN-1981 17-NOV-1981        161
ant    17-NOV-1981 17-NOV-1981          0
joe    17-NOV-1981 17-NOV-1981          0
KING   17-NOV-1981 17-NOV-1981          0
jim    17-NOV-1981 17-NOV-1981          0
choi   17-NOV-1981 23-JAN-1982         67
MILLER 23-JAN-1982 (null)          (null)
```

Значения столбца DIFF для четырех из пяти служащих, поступивших на работу в один день, равны нулю. Это неправильно. Даты найма на работу всех служащих, принятых в один день, должны сравниваться с одним следующим по отношению к ним значением HIREDATE, т. е. для всех служащих, нанятых 17 ноября, вычисления должны проводиться относительно значения HIREDATE служащего MILLER. В дан-

ном случае проблема в том, что функция LEAD упорядочивает строки по столбцу HIREDATE, но не пропускает дубликаты. Таким образом, например, при сравнении значений HIREDATE служащих ANT и JOE разница получается равной нулю, т. е. значение столбца DIFF для ANT – нуль. К счастью, Oracle предоставил простой выход из подобных ситуаций. При вызове функции LEAD в нее можно передать аргумент, точно определяющий местоположение строки для сравнения (т. е. будет ли это следующая строка, 10-я после рассматриваемой строка и т. д.). Таким образом, для служащего ANT мы проводим сравнение не со следующей строкой, а заглядываем на пять строк вперед (перешагиваем через все остальные дубликаты), на строку служащего MILLER. Строка MILLER отстоит от строки служащего JOE на четыре строки; от JIM – на три строки; от KING – на две; и красавчик CHOI находится на предыдущей относительно MILLER строке. Чтобы получить правильный ответ, просто передаем в функцию LEAD в качестве аргумента расстояние от строки каждого служащего до строки служащего MILLER. Решение показано ниже:

```
select ename, hiredate, next_hd,
       next_hd - hiredate diff
  from (
select deptno, ename, hiredate,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
        )
        )
```

```
ENAME  HIREDATE    NEXT_HD           DIFF
------ ----------- ----------- ----------
CLARK  09-JUN-1981 17-NOV-1981        161
ant    17-NOV-1981 23-JAN-1982         67
joe    17-NOV-1981 23-JAN-1982         67
jim    17-NOV-1981 23-JAN-1982         67
choi   17-NOV-1981 23-JAN-1982         67
KING   17-NOV-1981 23-JAN-1982         67
MILLER 23-JAN-1982 (null)          (null)
```

Теперь получен правильный результат. Даты приема на работу (значения столбца HIREDATE) всех служащих, принятых в один день, сравниваются со следующей по хронологии датой (другим значением HIREDATE), а не с таким же значением HIREDATE. Если решение этой задачи не вполне понятно, просто разложите запрос на составляющие. Начнем с вложенного запроса:

```
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
```

```
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10

DEPTNO ENAME  HIREDATE           CNT         RN
------ ------ ----------- ---------- ----------
    10 CLARK  09-JUN-1981          1          1
    10 ant    17-NOV-1981          5          1
    10 joe    17-NOV-1981          5          2
    10 jim    17-NOV-1981          5          3
    10 choi   17-NOV-1981          5          4
    10 KING   17-NOV-1981          5          5
    10 MILLER 23-JAN-1982          1          1
```

Оконная функция COUNT OVER (пересчитать) подсчитывает, сколько раз встречается каждое из значений HIREDATE, и возвращает это количество для каждой строки. Для строк с дублирующимися значениями HIREDATE возвращается значение 5. Ранжирующая функция ROW_NUMBER OVER ранжирует каждого служащего по столбцу EMPNO. Ранги разделяются по HIREDATE, т. е. если значение HIREDATE не дублируется, служащий получает ранг 1. Если значение HIREDATE дублируется, все дубликаты пересчитываются и получают соответствующий ранг. Их ранг может использоваться для вычисления расстояния до следующего значения HIREDATE (HIREDATE служащего MILLER). Это можно увидеть, вычитая RN из CNT и прибавляя 1 для каждой строки при вызове LEAD:

```
select deptno, ename, hiredate,
       cnt-rn+1 distance_to_miller,
       lead(hiredate,cnt-rn+1)over(order by hiredate) next_hd
  from (
select deptno,ename,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
 where deptno=10
       )

DEPTNO ENAME  HIREDATE    DISTANCE_TO_MILLER NEXT_HD
------ ------ ----------- ------------------ -----------
    10 CLARK  09-JUN-1981                  1 17-NOV-1981
    10 ant    17-NOV-1981                  5 23-JAN-1982
    10 joe    17-NOV-1981                  4 23-JAN-1982
    10 jim    17-NOV-1981                  3 23-JAN-1982
    10 choi   17-NOV-1981                  2 23-JAN-1982
    10 KING   17-NOV-1981                  1 23-JAN-1982
    10 MILLER 23-JAN-1982                  1 (null)
```

Как видите, при получении соответствующего аргумента (количества строк до строки для сравнения) функция LEAD обеспечивает верное сравнение дат.

9

Работа с датами

В данной главе представлены рецепты для поиска и обработки дат. Запросы с участием дат очень распространены. При работе с датами необходимо соответствующим образом мыслить и хорошо понимать функции, предоставляемые СУБД для обращения с такими данными. Рецепты этой главы формируют необходимый фундамент для перехода к более сложным запросам, в которых задействованы не только даты, но и время.

Прежде чем перейти к рецептам, я хочу еще раз повторить основную мысль (о которой говорилось в предисловии): эти рецепты должны использоваться как руководства к решению конкретных задач. Старайтесь думать «глобально». Например, если рецепт решает задачу для текущего месяца, помните, что его можно использовать (с небольшими изменениями) для любого месяца. Повторюсь, я хочу чтобы представленные рецепты использовались как руководства, а не как абсолютный окончательный вариант. Книга не может охватить ответы на все вопросы, но если понять те решения, которые в ней представлены, то останется просто изменить их соответственно своим задачам. Советую также рассматривать все предлагаемые альтернативные версии решений. Например, если задача решается с использованием определенной предоставляемой СУБД функции, стоит потратить время и силы, чтобы узнать о возможном альтернативном решении, которое может быть более или менее эффективным по сравнению с приведенным здесь. Знание всех возможных вариантов сделает вас лучшим разработчиком на SQL.

В представленных в данной главе рецептах используются простые типы дат. В случае использования более сложных типов дат решения должны быть скорректированы соответствующим образом.

Как определить, является ли год високосным

Задача

Требуется определить, является ли текущий год високосным.

Решение

Те, кто уже имеет некоторый опыт работы с SQL, несомненно, встречали несколько методов решения этой задачи. Хороши практически все известные мне решения, но представленное в данном рецепте, наверное, самое простое: проверяется последний день февраля; если он является 29-м днем, то текущий год – високосный.

DB2

Для возвращения всех дней февраля используйте рекурсивный оператор WITH. С помощью агрегатной функции MAX определите последний день февраля.

```
 1    with x (dy,mth)
 2      as (
 3  select dy, month(dy)
 4    from (
 5  select (current_date -
 6          dayofyear(current_date) days +1 days)
 7           +1 months as dy
 8    from t1
 9         ) tmp1
10   union all
11  select dy+1 days, mth
12    from x
13   where month(dy+1 day) = mth
14  )
15  select max(day(dy))
16    from x
```

Oracle

Чтобы найти последний день февраля, используйте функцию LAST_DAY (последний день):

```
1 select to_char(
2          last_day(add_months(trunc(sysdate,'y'),1)),
3         'DD')
4   from t1
```

PostgreSQL

Чтобы возвратить все дни февраля, используйте функцию GENERATE_SERIES. Затем с помощью агрегатной функции MAX найдите последний день февраля:

```
select max(to_char(tmp2.dy+x.id,'DD')) as dy
  from (
select dy, to_char(dy,'MM') as mth
  from (
select cast(cast(
            date_trunc('year',current_date) as date)
                       + interval '1 month' as date) as dy
  from t1
       ) tmp1
       ) tmp2, generate_series (0,29) x(id)
 where to_char(tmp2.dy+x.id,'MM') = tmp2.mth
```

MySQL

Чтобы найти последний день февраля, используйте функцию LAST_DAY:

```
select day(
       last_day(
       date_add(
       date_add(
       date_add(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day),
                interval 1 month))) dy
  from t1
```

SQL Server

Для возвращения всех дней февраля используйте рекурсивный оператор WITH. С помощью агрегатной функции MAX определите последний день февраля:

```
  with x (dy,mth)
    as (
select dy, month(dy)
  from (
select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
  from t1
       ) tmp1
 union all
select dateadd(dd,1,dy), mth
  from x
 where month(dateadd(dd,1,dy)) = mth
)
select max(day(dy))
  from x
```

Обсуждение[1]

DB2

Вложенный запрос TMP1 в рекурсивном представлении X возвращает первый день февраля следующим образом:

1. Определяется текущая дата.
2. С помощью функции DAYOFYEAR определяется, сколько дней нынешнего года прошло до текущей даты.
3. Вычитанием найденного количества дней из текущей даты получается 31 декабря предыдущего года, и добавляется один день, чтобы достичь 1 января текущего года.
4. Добавляется один месяц, чтобы получить 1 февраля.

Результат всех этих вычислений представлен ниже:

```
select (current_date -
          dayofyear(current_date) days +1 days) +1 months as dy
  from t1

DY
-----------
01-FEB-2005
```

Следующий шаг – с помощью функции MONTH из даты, возвращенной вложенным запросом TMP1, получить месяц:

```
select dy, month(dy) as mth
  from (
select (current_date -
          dayofyear(current_date) days +1 days) +1 months as dy
  from t1
       ) tmp1

DY           MTH
-----------  ---
01-FEB-2005    2
```

Представленные до сих пор результаты обеспечивают отправную точку для рекурсивной операции, в ходе которой генерируется каждый день февраля. Чтобы получить все дни февраля, многократно добавляем по одному дню к DY, до тех пор пока не будет возвращено 1 марта. Результаты выполнения операции WITH приведены ниже:

```
  with x (dy,mth)
    as (
select dy, month(dy)
  from (
select (current_date -
```

[1] Последний день февраля также можно получить путем вычитания одного дня из первого дня марта. – *Примеч. науч. ред.*

```
        dayofyear(current_date) days +1 days) +1 months as dy
  from t1
       ) tmp1
union all
select dy+1 days, mth
  from x
 where month(dy+1 day) = mth
)
select dy,mth
  from x
```

```
DY           MTH
-----------  ---
01-FEB-2005    2
...
10-FEB-2005    2
...
28-FEB-2005    2
```

Заключительный шаг – применить к столбцу DY функцию MAX, чтобы выбрать последний день февраля. Если это окажется 29-е число, то год – високосный.

Oracle

Первый шаг – найти начало года, используя функцию TRUNC:

```
select trunc(sysdate,'y')
  from t1
```

```
DY
-----------
01-JAN-2005
```

Поскольку первый день года – 1 января, следующий шаг – добавляем один месяц, чтобы получить 1 февраля:

```
select add_months(trunc(sysdate,'y'),1) dy
  from t1
```

```
DY
-----------
01-FEB-2005
```

Далее с помощью функции LAST_DAY находим последний день февраля:

```
select last_day(add_months(trunc(sysdate,'y'),1)) dy
  from t1
```

```
DY
-----------
28-FEB-2005
```

Последний шаг (необязательный) – используем функцию TO_CHAR, чтобы возвратить 28 или 29.

PostgreSQL

Первый шаг – проверить результаты, возвращенные вложенным запросом TMP1. С помощью функции DATE_TRUNC находим первый день текущего года и приводим результат к типу DATE:

```
select cast(date_trunc('year',current_date) as date) as dy
  from t1

DY
-----------
01-JAN-2005
```

Следующий шаг – добавляем месяц к первому дню текущего года, чтобы получить первый день февраля, приводя результат к типу DATE:

```
select cast(cast(
            date_trunc('year',current_date) as date)
                       + interval '1 month' as date) as dy
  from t1

DY
-----------
01-FEB-2005
```

Далее возвращаем из вложенного запроса TMP1 значение DY, выбирая из него с помощью функции TO_CHAR порядковый номер месяца:

```
select dy, to_char(dy,'MM') as mth
   from (
 select cast(cast(
             date_trunc('year',current_date) as date)
                        + interval '1 month' as date) as dy
   from t1
        ) tmp1

DY           MTH
-----------  ---
01-FEB-2005    2
```

Представленные до сих пор результаты составляют результирующее множество вложенного запроса TMP2. Следующий шаг – с помощью исключительно полезной функции GENERATE_SERIES получить 29 строк (со значениями от 1 до 29). Каждая возвращенная GENERATE_SERIES строка (множество этих строк образует множество под псевдонимом X) добавляется к DY вложенного запроса TMP2. Результаты частично показаны ниже:

```
select tmp2.dy+x.id as dy, tmp2.mth
  from (
select dy, to_char(dy,'MM') as mth
  from (
select cast(cast(
            date_trunc('year',current_date) as date)
                       + interval '1 month' as date) as dy
```

```
  from t1
       ) tmp1
       ) tmp2, generate_series (0,29) x(id)
 where to_char(tmp2.dy+x.id,'MM') = tmp2.mth
```

```
DY           MTH
-----------  ---
01-FEB-2005   02
...
10-FEB-2005   02
...
28-FEB-2005   02
```

Заключительный шаг – с помощью функции MAX выбрать последний день февраля. К полученному значению применяется функция TO_CHAR, которая возвратит 28 или 29.

MySQL

Первый шаг – найти первый день текущего года, вычитая из текущей даты количество прошедших до нее дней года и затем прибавляя один день. Все это делает функция DATE_ADD:

```
select date_add(
       date_add(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day) dy
  from t1
```

```
DY
-----------
01-JAN-2005
```

После этого добавляем один месяц, опять же используя функцию DATE_ADD:

```
select date_add(
       date_add(
       date_add(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day),
                interval 1 month) dy
  from t1
```

```
DY
-----------
01-FEB-2005
```

Добравшись до февраля, используем функцию LAST_DAY, чтобы найти последний день месяца:

```
select last_day(
       date_add(
       date_add(
```

```
       date_add(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day),
                interval 1 month)) dy
  from t1

DY
-----------
28-FEB-2005
```

Заключительный шаг (необязательный) – использовать функцию DAY, чтобы возвратить 28 или 29.

SQL Server

В данном решении для получения всех дней февраля используется рекурсивный оператор WITH. Первый шаг – найти первый день февраля. Для этого ищем первый день текущего года, вычитая из текущей даты количество прошедших до нее дней года и затем прибавляя один день. Получив первый день текущего года, с помощью функции DATEADD добавляем к нему один месяц, чтобы перейти к первому дню февраля:

```
select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
  from t1

DY
-----------
01-FEB-2005
```

Далее возвращаем первый день февраля и порядковый номер февраля:

```
select dy, month(dy) mth
  from (
select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
  from t1
       ) tmp1

DY          MTH
-----------  ---
01-FEB-2005    2
```

Используем рекурсивные возможности оператора WITH и добавляем по одному дню к DY, возвращаемому вложенным запросом TMP1, до тех пор пока не дойдем до марта (результаты частично показаны ниже):

```
  with x (dy,mth)
    as (
select dy, month(dy)
  from (
select dateadd(mm,1,(getdate()-datepart(dy,getdate()))+1) dy
  from t1
       ) tmp1
 union all
```

```
select dateadd(dd,1,dy), mth
  from x
 where month(dateadd(dd,1,dy)) = mth
)
select dy,mth from x

DY           MTH
-----------  ---
01-FEB-2005   02
...
10-FEB-2005   02
...
28-FEB-2005   02
```

Теперь, возвратив все дни февраля, с помощью функции MAX определяем последний день. Как необязательный заключительный шаг можно использовать функцию DAY, чтобы получить только число 28 или 29, а не всю дату.

Как определить количество дней в году

Задача

Требуется подсчитать количество дней в текущем году.

Решение

Количество дней в текущем году – это разница между первым днем следующего года и первым днем текущего года (в днях). Каждое решение включает в себя следующие этапы:

1. Определение первого дня текущего года.
2. Добавление одного года к этой дате (для получения первого дня следующего года).
3. Вычитание текущего года из результата шага 2.

Решения отличаются только используемыми встроенными функциями.

DB2

С помощью функции DAYOFYEAR найдите первый день текущего года и определите количество дней в текущем году, используя функцию DAYS:

```
1 select days((curr_year + 1 year)) - days(curr_year)
2   from (
3 select (current_date -
4         dayofyear(current_date) day +
5          1 day) curr_year
6   from t1
7        ) x
```

Oracle

С помощью функции TRUNC найдите начало текущего года и, используя функцию ADD_MONTHS, определите начало следующего года:

```
1 select add_months(trunc(sysdate,'y'),12) - trunc(sysdate,'y')
2   from dual
```

PostgreSQL

С помощью функции DATE_TRUNC найдите начало текущего года. Затем, используя арифметику интервалов, найдите начало следующего года:

```
1 select cast((curr_year + interval '1 year') as date) - curr_year
2   from (
3 select cast(date_trunc('year',current_date) as date) as curr_year
4   from t1
5        ) x
```

MySQL

Используйте ADDDATE, чтобы найти начало текущего года. С помощью функции DATEDIFF и арифметики интервалов определите количество дней в году:

```
1 select datediff((curr_year + interval 1 year),curr_year)
2   from (
3 select adddate(current_date,-dayofyear(current_date)+1) curr_year
4   from t1
5        ) x
```

SQL Server

С помощью функции DATEADD найдите первый день текущего года. Чтобы возвратить число дней в текущем году, используйте DATEDIFF:

```
1 select datediff(d,curr_year,dateadd(yy,1,curr_year))
2   from (
3 select dateadd(d,-datepart(dy,getdate())+1,getdate()) curr_year
4   from t1
5        ) x
```

Обсуждение

DB2

Первый шаг – найти первый день текущего года. Для определения количества прошедших дней текущего года используется функция DAYOFYEAR (день года). Вычитая это значение из текущей даты, получаем последний день прошлого года и затем добавляем 1:

```
select (current_date -
        dayofyear(current_date) day +
         1 day) curr_year
```

```
  from t1

CURR_YEAR
-----------
01-JAN-2005
```

Теперь, имея первый день текущего года, просто прибавляем к нему один год, тем самым получаем первый день следующего года. Затем вычитаем начало текущего года из начала следующего года.

Oracle

Первый шаг – найти первый день текущего года, что можно без труда сделать, вызвав встроенную функцию TRUNC и передав в качестве второго аргумента «Y» (таким образом получая, исходя из текущей даты, первый день текущего года):

```
select select trunc(sysdate,'y') curr_year
  from dual

CURR_YEAR
-----------
01-JAN-2005
```

Затем добавляем один год, чтобы получить первый день следующего года. Наконец, находим разность между двумя датами и определяем количество дней в текущем году.

PostgreSQL

Начинаем с получения первого дня текущего года. Для этого вызываем функцию DATE_TRUNC следующим образом:

```
select cast(date_trunc('year',current_date) as date) as curr_year
  from t1

CURR_YEAR
-----------
01-JAN-2005
```

Теперь можно без труда добавить год, чтобы получить первый день следующего года, и тогда останется только вычесть одну дату из другой. Вычитать следует более раннюю дату из более поздней. В результате будет получено количество дней в текущем году.

MySQL

Первый шаг – найти первый день текущего года. С помощью функции DAYOFYEAR находим количество прошедших дней текущего года. Вычитаем это значение из текущей даты и добавляем 1:

```
select adddate(current_date,-dayofyear(current_date)+1) curr_year
  from t1

CURR_YEAR
```

```
-----------
01-JAN-2005
```

Получив первый день текущего года, добавляем к нему один год, в результате чего получаем первый день следующего года. Затем вычитаем начало текущего года из начала следующего года. В итоге находим число дней в текущем году.

SQL Server

Первый шаг – найти первый день текущего года. Вычитаем из текущей даты количество прошедших дней года и добавляем 1, используя функции DATEADD и DATEPART:

```
select dateadd(d,-datepart(dy,getdate())+1,getdate()) curr_year
  from t1

CURR_YEAR
-----------
01-JAN-2005
```

Получив первый день текущего года, добавляем к нему один год, в результате чего получаем первый день следующего года. Затем вычитаем начало текущего года из начала следующего. В итоге находим количество дней в текущем году.

Извлечение единиц времени из даты

Задача

Требуется разложить текущую дату на шесть частей: день, месяц, год, секунды, минуты и часы. Результаты должны быть возвращены в численном виде.

Решение

В данном рецепте текущая дата выбрана для иллюстрации, он свободно может использоваться с другими датами. В главе 1 говорилось о важности изучения и использования преимуществ встроенных функций, предоставляемых СУБД. Это особенно актуально, когда дело касается дат. Существует множество других способов извлечения единиц времени из даты, кроме представленных в данном рецепте. Очень полезно поэкспериментировать с разными техниками и функциями.

DB2

DB2 реализует ряд встроенных функций, которые упрощают задачу по извлечению частей даты. Имена функций HOUR (час), MINUTE (минута), SECOND (секунда), DAY, MONTH и YEAR соответствуют возвращаемым ими единицам измерения времени. Если требуется получить день, используется функция DAY, час – функция HOUR и т. д. Например:

```
1  select   hour( current_timestamp ) hr,
2         minute( current_timestamp ) min,
3         second( current_timestamp ) sec,
4            day( current_timestamp ) dy,
5          month( current_timestamp ) mth,
6           year( current_timestamp ) yr
7    from t1
```

```
  HR   MIN   SEC    DY   MTH    YR
---- ----- ----- ----- ----- -----
  20    28    36    15     6  2005
```

Oracle

Для выбора определенных частей даты, соответствующих тем или иным единицам времени, используйте функции TO_CHAR и TO_NUMBER:

```
1  select to_number(to_char(sysdate,'hh24')) hour,
2         to_number(to_char(sysdate,'mi')) min,
3         to_number(to_char(sysdate,'ss')) sec,
4         to_number(to_char(sysdate,'dd')) day,
5         to_number(to_char(sysdate,'mm')) mth,
6         to_number(to_char(sysdate,'yyyy')) year
7    from dual
```

```
HOUR   MIN   SEC   DAY   MTH  YEAR
---- ----- ----- ----- ----- -----
  20    28    36    15     6  2005
```

PostgreSQL

Для выбора определенных частей даты, соответствующих тем или иным единицам времени, используйте функции TO_CHAR и TO_NUMBER:

```
1 select to_number(to_char(current_timestamp,'hh24'),'99') as hr,
2        to_number(to_char(current_timestamp,'mi'),'99') as min,
3        to_number(to_char(current_timestamp,'ss'),'99') as sec,
4        to_number(to_char(current_timestamp,'dd'),'99') as day,
5        to_number(to_char(current_timestamp,'mm'),'99') as mth,
6        to_number(to_char(current_timestamp,'yyyy'),'9999') as yr
7   from t1
```

```
  HR   MIN   SEC   DAY   MTH    YR
---- ----- ----- ----- ----- -----
  20    28    36    15     6  2005
```

MySQL

Для выбора определенных частей даты, соответствующих тем или иным единицам времени, используйте функцию DATE_FORMAT:

```
1 select date_format(current_timestamp,'%k') hr,
2        date_format(current_timestamp,'%i') min,
3        date_format(current_timestamp,'%s') sec,
4        date_format(current_timestamp,'%d') dy,
```

```
5        date_format(current_timestamp,'%m') mon,
6        date_format(current_timestamp,'%Y') yr
7   from t1
```

```
  HR   MIN   SEC    DY   MTH    YR
---- ----- ----- ----- ----- -----
  20    28    36    15     6  2005
```

SQL Server

Для выбора определенных частей даты, соответствующих тем или иным единицам времени, используйте функцию DATEPART (часть даты):

```
1  select datepart( hour,  getdate()) hr,
2         datepart( minute,getdate()) min,
3         datepart( second,getdate()) sec,
4         datepart( day,   getdate()) dy,
5         datepart( month, getdate()) mon,
6         datepart( year,  getdate()) yr
7    from t1
```

```
  HR   MIN   SEC    DY   MTH    YR
---- ----- ----- ----- ----- -----
  20    28    36    15     6  2005
```

Обсуждение

В этих решениях нет ничего необычного, просто используем то, за что уже заплачено. Потрудитесь изучить функции для работы с датами, которые имеете в своем распоряжении. В решениях этого рецепта мы лишь поверхностно затронули представленные функции. Каждая из них может принимать намного больше аргументов и возвращать больше информации, чем показано здесь.

Определение первого и последнего дней месяца

Задача

Требуется получить первый и последний дни текущего месяца.

Решение

Представленные здесь решения обеспечивают поиск первого и последнего дней текущего месяца. Текущий месяц выбран произвольно. С небольшими корректировками эти решения можно применять к разным месяцам.

DB2

С помощью функции DAY возвращаем количество прошедших в текущем месяце дней на основании представленной текущей даты. Вычитаем это значение из текущей даты и добавляем 1, чтобы получить

первый день месяца. Чтобы определить последний день месяца, добавляем один месяц к текущей дате и вычитаем значение, возвращенное функцией DAY для текущей даты:

```
1 select (current_date - day(current_date) day +1 day) firstday,
2        (current_date +1 month -day(current_date) day) lastday
3   from t1
```

Oracle

Первый день месяца находим с помощью функции TRUNC, последний день – с помощью функции LAST_DAY:

```
1 select trunc(sysdate,'mm') firstday,
2        last_day(sysdate) lastday
3   from dual
```

Описанное здесь применение TRUNC приведет к потере всех компонентов времени даты, тогда как LAST_DAY сохранит время.

PostgreSQL

С помощью функции DATE_TRUNC получаем из текущей даты первый день месяца. Имея первый день месяца, добавляем к нему один месяц и вычитаем один день, чтобы найти последний день текущего месяца:

```
1 select firstday,
2        cast(firstday + interval '1 month'
3                      - interval '1 day' as date) as lastday
4   from (
5 select cast(date_trunc('month',current_date) as date) as firstday
6   from t1
7        ) x
```

MySQL

С помощью функций DATE_ADD и DAY возвращаем количество прошедших в текущем месяце дней согласно представленной текущей дате. Затем вычитаем это значение из текущей даты и добавляем 1, чтобы найти первый день месяца. Последний день месяца получаем, используя функцию LAST_DAY:

```
1 select date_add(current_date,
2                 interval -day(current_date)+1 day) firstday,
3        last_day(current_date) lastday
4   from t1
```

SQL Server

С помощью функций DATEADD и DAY возвращаем количество прошедших в текущем месяце дней согласно представленной текущей дате. Затем вычитаем это значение из текущей даты и добавляем 1, что-

бы найти первый день месяца. Чтобы получить последний день, добавляем один месяц к текущей дате и вычитаем из полученного результата значение, возвращенное функцией DAY для текущей даты, опять используя функции DAY и DATEADD:

```
1 select dateadd(day,-day(getdate())+1,getdate()) firstday,
2        dateadd(day,
3                -day(getdate()),
4                dateadd(month,1,getdate())) lastday
5   from t1
```

Обсуждение

DB2

Для получения первого дня месяца используем функцию DAY: она возвращает день соответственно переданной в нее дате. Если из текущей даты вычесть значение, возвращенное DAY(CURRENT_DATE), то получаем последний день предыдущего месяца; добавляем к нему один день и имеем первый день текущего месяца. Чтобы найти последний день месяца, добавляем один месяц к текущей дате. Тем самым мы попадем в то же число следующего месяца (независимо от того, сколько дней в текущем месяце)[1]. Затем вычитаем значение, возвращенное DAY(CURRENT_DATE), чтобы получить последний день текущего месяца.

Oracle

Чтобы найти первый день текущего месяца, используем функцию TRUNC, передавая в нее в качестве второго аргумента «mm», что обеспечит возвращение первого дня месяца соответственно текущей дате. Чтобы найти последний день, просто используем функцию LAST_DAY.

PostgreSQL

Чтобы найти первый день текущего месяца, используем функцию DATE_TRUNC, передавая в нее в качестве второго аргумента «month», что обеспечит возвращение первого дня месяца соответственно текущей дате. Чтобы найти последний день, добавляем один месяц к первому дню месяца и вычитаем один день.

MySQL

Чтобы найти первый день текущего месяца, используем функцию DAY. Она возвращает день месяца соответственно переданной дате.

[1] Это может вызвать проблему, если в следующем месяце меньше дней, чем текущая дата. Поэтому более корректно сначала вычесть количество прошедших в текущем месяце дней и получить последний день предыдущего месяца, а только потом добавить один месяц для получения последнего дня текущего месяца. – *Примеч. науч. ред.*

Если из текущей даты вычесть значение, возвращенное DAY(CURRENT_DATE), получаем последний день предыдущего месяца; добавляем один день и в итоге имеем первый день текущего месяца. Последний день находим просто с помощью функции LAST_DAY.

SQL Server

Чтобы найти первый день текущего месяца, используем функцию DAY. Она возвращает день месяца соответственно переданной в нее дате. Если из текущей даты вычесть значение, возвращенное DAY(GETDATE()), получаем последний день предыдущего месяца; добавляем один день и в итоге имеем первый день текущего месяца. Последний день находим с помощью функции DATEADD. Добавляем один месяц к текущей дате, затем вычитаем значение, возвращенное DAY(GETDATE()), и получаем последний день текущего месяца.[1]

Выбор всех дат года, выпадающих на определенный день недели

Задача

Требуется найти все даты года, соответствующие заданному дню недели. Например, стоит задача сформировать список пятниц текущего года.

Решение

Независимо от СУБД основой решения являются возвращение всех дней текущего года и выбор из них только тех, которые соответствуют интересующему дню недели. В приведенных примерах извлекаются все пятницы года.

DB2

Рекурсивным оператором WITH возвращаем все дни текущего года. Затем с помощью функции DAYNAME выбираем только пятницы:

```
1    with x (dy,yr)
2      as (
3  select dy, year(dy) yr
4    from (
5  select (current_date -
6           dayofyear(current_date) days +1 days) as dy
7    from t1
8         ) tmp1
9   union all
```

[1] Та же проблема, что и в решении для DB2. Поэтому сначала следует вычесть количество прошедших в текущем месяце дней и получить последний день предыдущего месяца, а только потом добавить один месяц. – *Примеч. науч. ред.*

```
10  select dy+1 days, yr
11    from x
12   where year(dy +1 day) = yr
13  )
14  select dy
15    from x
16   where dayname(dy) = 'Friday'
```

Oracle

Рекурсивным оператором CONNECT BY возвращаем все дни текущего года. Затем с помощью функции TO_CHAR выбираем только пятницы:

```
 1    with x
 2      as (
 3  select trunc(sysdate,'y')+level-1 dy
 4    from t1
 5    connect by level <=
 6       add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
 7  )
 8  select *
 9    from x
10   where to_char( dy, 'dy') = 'fri'
```

PostgreSQL

Используя функцию GENERATE_SERIES, возвращаем все дни текущего года. Затем с помощью функции TO_CHAR выбираем только пятницы:

```
 1 select cast(date_trunc('year',current_date) as date)
 2        + x.id as dy
 3   from generate_series (
 4         0,
 5         ( select cast(
 6                  cast(
 7            date_trunc('year',current_date) as date)
 8                       + interval '1 years' as date)
 9                       - cast(
10                    date_trunc('year',current_date) as date) )-1
11         ) x(id)
12  where to_char(
13           cast(
14     date_trunc('year',current_date)
15                as date)+x.id,'dy') = 'fri'
```

MySQL

Используем сводную таблицу T500, чтобы возвратить все дни текущего года. Затем с помощью функции DAYNAME выбираем только пятницы:

```
1  select dy
2    from (
```

```
 3  select adddate(x.dy,interval t500.id-1 day) dy
 4    from (
 5  select dy, year(dy) yr
 6    from (
 7  select adddate(
 8         adddate(current_date,
 9                 interval -dayofyear(current_date) day),
10                 interval 1 day ) dy
11    from t1
12         ) tmp1
13         ) x,
14         t500
15   where year(adddate(x.dy,interval t500.id-1 day)) = x.yr
16         ) tmp2
17   where dayname(dy) = 'Friday'
```

SQL Server

Рекурсивным оператором WITH возвращаем все дни текущего года. Затем с помощью функции DAYNAME выбираем только пятницы:

```
 1    with x (dy,yr)
 2      as (
 3  select dy, year(dy) yr
 4    from (
 5  select getdate()-datepart(dy,getdate())+1 dy
 6    from t1
 7         ) tmp1
 8   union all
 9  select dateadd(dd,1,dy), yr
10    from x
11   where year(dateadd(dd,1,dy)) = yr
12  )
13  select x.dy
14    from x
15   where datename(dw,x.dy) = 'Friday'
16  option (maxrecursion 400)
```

Обсуждение

DB2

Чтобы выбрать все пятницы текущего года, необходимо суметь возвратить все дни текущего года. Первый шаг – найти первый день года с помощью функции DAYOFYEAR. Вычитая значение, возвращенное DAYOFYEAR(CURRENT_DATE) из текущей даты, получаем 31 декабря предыдущего года. Затем добавляем 1, чтобы получить первый день текущего года:

```
select (current_date -
         dayofyear(current_date) days +1 days) as dy
  from t1
```

```
DY
-----------
01-JAN-2005
```

Имея первый день года, с помощью оператора WITH реализуем многократное добавление одного дня, начиная с первого дня года, до тех пор пока не дойдем до следующего года. Результирующее множество будет включать все дни текущего года (строки, возвращенные рекурсивным представлением X, частично показаны ниже):

```
  with x (dy,yr)
    as (
select dy, year(dy) yr
  from (
select (current_date -
         dayofyear(current_date) days +1 days) as dy
  from t1
       ) tmp1
union all
select dy+1 days, yr
  from x
 where year(dy +1 day) = yr
)
select dy
  from x
```

```
DY
-----------
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

Завершающий шаг – с помощью функции DAYNAME выбрать строки, соответствующие пятницам.

Oracle

Чтобы выбрать все пятницы текущего года, необходимо возвратить все дни текущего года. Сначала используем функцию TRUNC для получения первого дня года:

```
select trunc(sysdate,'y') dy
  from t1
```

```
DY
-----------
01-JAN-2005
```

Далее с помощью оператора CONNECT BY возвращаем все дни текущего года (чтобы понять, как генерировать строки с помощью CONNECT

BY, смотрите раздел «Формирование последовательности числовых значений» главы 10).

В данном рецепте используется оператор WITH, но можно также применять вложенный запрос.

На момент написания данной книги Oracle-оператор WITH не приспособлен для рекурсивных операций (в отличие от DB2 и SQL Server). Рекурсивные операции реализуются с помощью CONNECT BY. Результирующее множество, возвращаемое вложенным представлением X, частично показано ниже:

```
  with x
    as (
select trunc(sysdate,'y')+level-1 dy
  from t1
  connect by level <=
     add_months(trunc(sysdate,'y'),12)-trunc(sysdate,'y')
)
select *
  from x

DY
-----------
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

Завершающий шаг – с использованием функции TO_CHAR выбрать только пятницы.

PostgreSQL

Чтобы выбрать все пятницы текущего года, необходимо сначала получить все дни текущего года (каждый день в отдельной строке). Для этого используем функцию GENERATE_SERIES. Она должна возвратить диапазон значений от 0 до значения, равного количеству дней в текущем году минус 1. Первый передаваемый в GENERATE_SERIES параметр – 0, тогда как второй параметр – это запрос, определяющий количество дней в текущем году (поскольку полученное значение добавляется к первому дню года, необходимо добавить на 1 день меньше, чем есть в текущем году, чтобы не попасть в первый день следующего года). Результат, возвращаемый вторым параметром функции GENERATE_SERIES, показан ниже:

```
select cast(
       cast(
```

```
 date_trunc('year',current_date) as date)
           + interval '1 years' as date)
           - cast(
       date_trunc('year',current_date) as date)-1 as cnt
  from t1

CNT
---
364
```

Учитывая приведенный выше результат, вызов GENERATE_SERIES в операторе FROM будет выглядеть так: GENERATE_SERIES (0, 364). Если текущий год является високосным, как, например, 2004, то второй параметр будет равен 365.

После формирования списка дат года следующий шаг – добавление значений, возвращенных GENERATE_SERIES, к первому дню текущего года. Результаты частично показаны ниже:

```
select cast(date_trunc('year',current_date) as date)
       + x.id as dy
  from generate_series (
       0,
       ( select cast(
                cast(
           date_trunc('year',current_date) as date)
                     + interval '1 years' as date)
                     - cast(
                date_trunc('year',current_date) as date) )-1
       ) x(id)

DY
-----------
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

Завершающий шаг – с использованием функции TO_CHAR выбрать только пятницы.

MySQL

Чтобы выбрать все пятницы текущего года, необходимо возвратить все дни текущего года. Первый шаг – с помощью функции DAYOFYEAR найти первый день года. Вычитаем из текущей даты значение, возвращенное DAYOFYEAR(CURRENT_DATE), и затем добавляем 1, чтобы получить первый день текущего года:

```
select adddate(
       adddate(current_date,
```

```
                interval -dayofyear(current_date) day),
                interval 1 day ) dy
  from t1

DY
-----------
01-JAN-2005
```

Затем используем таблицу T500, чтобы получить количество строк, соответствующее количеству дней текущего года. Сделать это можно, последовательно добавляя значения столбца T500.ID к первому дню года, до тех пор пока не будет достигнут первый день следующего года. Результаты этой операции частично представлены ниже:

```
select adddate(x.dy,interval t500.id-1 day) dy
  from (
select dy, year(dy) yr
  from (
select adddate(
       adddate(current_date,
                interval -dayofyear(current_date) day),
                interval 1 day ) dy
  from t1
       ) tmp1
       ) x,
       t500
 where year(adddate(x.dy,interval t500.id-1 day)) = x.yr

DY
-----------
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

Завершающий шаг – с использованием функции DAYNAME выбрать только пятницы.

SQL Server

Чтобы выбрать все пятницы текущего года, необходимо получить все дни текущего года. Первый шаг – с помощью функции DATEPART найти первый день года. Вычитаем из текущей даты значение, возвращенное DATEPART(DY,GETDATE()), и затем добавляем 1, чтобы получить первый день текущего года

```
select getdate()-datepart(dy,getdate())+1 dy
  from t1

DY
-----------
01-JAN-2005
```

Имея первый день года, с помощью оператора WITH и функции DATEADD реализуем многократное добавление одного дня, начиная с первого дня года, до тех пор пока не будет получен первый день следующего года. Результирующее множество будет включать все дни текущего года (строки, возвращенные рекурсивным представлением X, частично показаны ниже):

```
with x (dy,yr)
  as (
select dy, year(dy) yr
  from (
select getdate()-datepart(dy,getdate())+1 dy
  from t1
       ) tmp1
 union all
select dateadd(dd,1,dy), yr
  from x
 where year(dateadd(dd,1,dy)) = yr
)
select x.dy
  from x
option (maxrecursion 400)
```

```
DY
-----------
01-JAN-2005
...
15-FEB-2005
...
22-NOV-2005
...
31-DEC-2005
```

Наконец, с помощью функции DATENAME выбираем только те строки, которые соответствуют пятницам. Чтобы данное решение было работоспособным, значение параметра MAXRECURSION должно быть задано не менее 366 (ограничение количества дней в году в рекурсивном представлении X гарантирует невозможность получения более 366 строк).

Определение дат первого и последнего появления заданного дня недели

Задача

Требуется найти, например, первый и последний понедельники текущего месяца.

Решение

Понедельник и текущий месяц выбраны в качестве примера. Представленные в данном рецепте решения могут использоваться для любого

дня недели и любого месяца. Поскольку один и тот же день недели повторяется каждые семь дней, получив первую соответствующую ему дату, можно добавить к ней 7 дней и найти вторую дату, а также добавить 14 дней и найти третью. Аналогично, если известна последняя соответствующая заданному дню недели дата месяца, вычитая 7 дней, получаем третью, а, вычитая 14, – вторую дату месяца.

DB2

Используя рекурсивный оператор WITH, получаем все дни текущего месяца. С помощью выражения CASE отмечаем все понедельники. Первым и последним понедельниками будут самая ранняя и самая поздняя из отмеченных дат:

```
 1    with x (dy,mth,is_monday)
 2      as (
 3  select dy,month(dy),
 4         case when dayname(dy)='Monday'
 5              then 1 else 0
 6         end
 7    from (
 8  select (current_date-day(current_date) day +1 day) dy
 9    from t1
10         ) tmp1
11   union all
12  select (dy +1 day), mth,
13         case when dayname(dy +1 day)='Monday'
14              then 1 else 0
15         end
16    from x
17   where month(dy +1 day) = mth
18  )
19  select min(dy) first_monday, max(dy) last_monday
20    from x
21   where is_monday = 1
```

Oracle

Чтобы найти первый и последний понедельники текущего месяца, используйте функции NEXT_DAY и LAST_DAY в сочетании с небольшим объемом операций над датами:

```
select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday,
       next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday
  from dual
```

PostgreSQL

С помощью функции DATE_TRUNC получите первый день месяца. После этого, используя простые вычисления над числовыми значениями дней недели (Вс.–Сб. соответствуют 1–7), находите первый и последний понедельники текущего месяца:

```
select first_monday,
       case to_char(first_monday+28,'mm')
            when mth then first_monday+28
                     else first_monday+21
       end as last_monday
  from (
select case sign(cast(to_char(dy,'d') as integer)-2)
            when  0
            then dy
            when -1
            then dy+abs(cast(to_char(dy,'d') as integer)-2)
            when  1
            then (7-(cast(to_char(dy,'d') as integer)-2))+dy
       end as first_monday,
       mth
  from (
select cast(date_trunc('month',current_date) as date) as dy,
       to_char(current_date,'mm') as mth
  from t1
       ) x
       ) y
```

MySQL

Используйте функцию ADDDATE, чтобы получить первый день месяца. После этого путем простых вычислений над числовыми значениями дней недели (Вс.–Сб. соответствуют 1–7) находите первый и последний понедельники текущего месяца:

```
select first_monday,
       case month(adddate(first_monday,28))
            when mth then adddate(first_monday,28)
                     else adddate(first_monday,21)
       end last_monday
  from (
select case sign(dayofweek(dy)-2)
            when  0 then dy
            when -1 then adddate(dy,abs(dayofweek(dy)-2))
            when  1 then adddate(dy,(7-(dayofweek(dy)-2)))
       end first_monday,
       mth
  from (
select adddate(adddate(current_date,-day(current_date)),1) dy,
       month(current_date) mth
  from t1
       ) x
       ) y
```

SQL Server

Используйте рекурсивный оператор WITH, чтобы получить все дни текущего месяца, и затем с помощью выражения CASE отметьте все

понедельники. Первым и последним понедельниками будут самая ранняя и самая поздняя из отмеченных дат:

```
 1    with x (dy,mth,is_monday)
 2      as (
 3  select dy,mth,
 4         case when datepart(dw,dy) = 2
 5              then 1 else 0
 6         end
 7    from (
 8  select dateadd(day,1,dateadd(day,-day(getdate()),getdate())) dy,
 9         month(getdate()) mth
10    from t1
11         ) tmp1
12   union all
13  select dateadd(day,1,dy),
14         mth,
15         case when datepart(dw,dateadd(day,1,dy)) = 2
16              then 1 else 0
17         end
18    from x
19   where month(dateadd(day,1,dy)) = mth
20  )
21  select min(dy) first_monday,
22         max(dy) last_monday
23    from x
24   where is_monday = 1
```

Обсуждение

DB2 и SQL Server

В DB2 и SQL Server для решения поставленной задачи используются разные функции, но методики аналогичны. Если внимательно рассмотреть решения, то единственное отличие будет найдено в способе суммирования дат. В данном обсуждении анализируются оба решения. Для демонстрации промежуточных шагов используется код решения для DB2.

Если в используемой версии SQL Server или DB2 рекурсивный оператор WITH не предоставляется, можно использовать технику, предлагаемую для PostgreSQL.

Первый шаг при поиске первого и последнего понедельников текущего месяца состоит в получении первого дня месяца. Его поиском занимается вложенный запрос TMP1 в рекурсивном представлении X. Сначала определяется текущая дата, а именно день месяца, соответствующий текущей дате. День месяца, соответствующий текущей дате, представляет, сколько дней месяца прошло (например, 10 апреля – это 10-й день апреля). Если вычесть это значение из текущей даты, получится последний день предыдущего месяца (например, в результате

вычитания 10 из 10-го апреля будем иметь последний день марта). После этого вычитания просто добавляем один день, чтобы получить первый день текущего месяца:

```
select (current_date-day(current_date) day +1 day) dy
  from t1

DY
-----------
01-JUN-2005
```

Далее, используя функцию MONTH, получаем месяц текущей даты и с помощью простого выражения CASE определяем, является ли первый день этого месяца понедельником или нет:

```
select dy, month(dy) mth,
        case when dayname(dy)='Monday'
             then 1 else 0
        end is_monday
   from (
 select (current_date-day(current_date) day +1 day) dy
   from t1
        ) tmp1

DY          MTH  IS_MONDAY
----------- --- ----------
01-JUN-2005   6          0
```

После этого используем рекурсивные возможности оператора WITH и последовательно добавляем по одному дню, начиная с первого дня месяца, до тех пор пока не дойдем до первого дня следующего месяца. В ходе этого с помощью выражения CASE отбираем понедельники (отмечая их флагом «1»). Результат рекурсивного представления X частично показан ниже:

```
with x (dy,mth,is_monday)
     as (
 select dy,month(dy) mth,
        case when dayname(dy)='Monday'
             then 1 else 0
        end is_monday
   from (
 select (current_date-day(current_date) day +1 day) dy
   from t1
        ) tmp1
  union all
 select (dy +1 day), mth,
        case when dayname(dy +1 day)='Monday'
             then 1 else 0
        end
   from x
  where month(dy +1 day) = mth
 )
```

```
select *
  from x

DY          MTH  IS_MONDAY
----------- --- ----------
01-JUN-2005  6           0
02-JUN-2005  6           0
03-JUN-2005  6           0
04-JUN-2005  6           0
05-JUN-2005  6           0
06-JUN-2005  6           1
07-JUN-2005  6           0
08-JUN-2005  6           0
...
```

Значения 1 столбца IS_MONDAY будут соответствовать понедельникам. Таким образом, заключительный шаг – применяя агрегатные функции MIN и MAX к строкам, в которых значение поля IS_MONDAY равно 1, находим первый и последний понедельники месяца.

Oracle

Функция NEXT_DAY значительно упрощает решение этой задачи. Чтобы найти первый понедельник текущего месяца, сначала посредством некоторых вычислений с привлечением функции TRUNC возвращаем последний день предыдущего месяца:

```
select trunc(sysdate,'mm')-1 dy
  from dual

DY
-----------
31-MAY-2005
```

Затем используем функцию NEXT_DAY, чтобы найти первый понедельник после последнего дня предыдущего месяца (т. е. первый понедельник текущего месяца):

```
select next_day(trunc(sysdate,'mm')-1,'MONDAY') first_monday
  from dual

FIRST_MONDAY
------------
06-JUN-2005
```

Чтобы найти последний понедельник текущего месяца, сначала с помощью функции TRUNC возвращаем первый день текущего месяца:

```
select trunc(sysdate,'mm') dy
  from dual

DY
-----------
01-JUN-2005
```

Следующий шаг – находим последнюю неделю (последние семь дней) месяца. С помощью функции LAST_DAY получаем последний день месяца и вычитаем семь дней:

```
select last_day(trunc(sysdate,'mm'))-7 dy
  from dual

DY
-----------
23-JUN-2005
```

Здесь мы возвращаемся на семь дней от последнего дня месяца, чтобы гарантированно получить, по крайней мере, по одному разу каждый день недели. Последний шаг – использовать функцию NEXT_DAY, чтобы найти следующий (и последний) понедельник месяца:

```
select next_day(last_day(trunc(sysdate,'mm'))-7,'MONDAY') last_monday
  from dual

LAST_MONDAY
-----------
27-JUN-2005
```

PostgreSQL и MySQL

В PostgreSQL и MySQL используется аналогичный подход, отличие состоит лишь в вызываемых функциях. Соответствующие запросы предельно просты, несмотря на их размеры; некоторые издержки возникают при поиске первого и последнего понедельников текущего месяца.

Первый шаг – найти первый день текущего месяца. Следующий шаг – определить первый понедельник месяца. Поскольку специальной функции, которая возвращала бы следующую дату, соответствующую заданному дню недели, нет, необходимо прибегнуть к небольшим вычислениям. Выражение CASE, начинающееся в строке 7 (любого решения), вычисляет разницу между числовым значением дня недели первого дня месяца и числовым значением, соответствующим понедельнику. Исходя из того, что функция TO_CHAR (PostgresSQL) с форматной маской 'D' или 'd' и функция DAYOFWEEK (MySQL) для дней недели от воскресенья до субботы возвращают числовые значения от 1 до 7, понедельник всегда будет представлен цифрой 2. Сначала выражение CASE проверяет знак (SIGN) полученной разности между первым днем месяца (каким бы он ни был) и числовым значением понедельника (2). Если результат равен 0, первый день месяца выпадает на понедельник, и это первый понедельник месяца. Если результат равен -1, первый день месяца выпадает на воскресенье, и чтобы найти первый понедельник, надо просто добавить в первому дню месяца разницу в днях между 2 и 1 (числовые значения понедельника и воскресенья соответственно).

Если возникают сложности с пониманием логики этих операций, забудьте о названиях дней недели и оперируйте только числами. Например, пусть первым днем месяца является

вторник, и необходимо найти следующую пятницу. При использовании функции TO_CHAR с форматной маской 'd' или функции DAYOFWEEK получаем 6 для пятницы и 3 для вторника. Чтобы получить 6 из 3, просто находим их разность (6 − 3 = 3) и прибавляем ее к меньшему значению ((6 − 3) + 3 = 6). Итак, независимо от реальных дат, если числовое значение дня, с которого начинается отсчет, меньше, чем числовое значение искомого дня, добавление разности между этими двумя днями к начальной дате даст искомую дату.

Если выражение SIGN дает в результате 1, первый день месяца выпадает на день между вторником и субботой (включительно). Если числовое значение первого дня месяца больше 2 (понедельник), вычтете из 7 разность между числовым значением первого дня месяца и числовым значением понедельника (2) и затем прибавьте полученное значение к первому дню месяца. Таким образом, получаем искомый день недели, в данном случае понедельник.

Опять же, если есть затруднения с пониманием логики операций, забудьте о названиях дней недели и просто оперируйте числами. Например, предположим, требуется найти следующий вторник, начиная с пятницы. Числовое значение вторника (3) меньше, чем значение пятницы (6). Чтобы попасть на третий день с шестого, вычитаем из 7 разность между этими двумя значениями (7 − (|3 − 6|) = 4) и добавляем результат (4) к начальному дню (пятнице). (Применение вертикальных черточек в выражении |3 − 6| обеспечивает получение абсолютного значения разности.) Здесь мы не добавляем 4 к 6 (что в результате дало бы 10), мы добавляем четыре дня к пятнице, что обеспечит возвращение следующего вторника.

Идея, стоящая за применением выражения CASE, заключается в создании своего рода функции «следующий день» для PostgreSQL и MySQL. Если вычисления начинаются не с первого дня месяца, в столбце DY будет располагаться значение, возвращенное функцией CURRENT_DATE. Выражение CASE в этом случае возвратит дату следующего после текущей даты понедельника (если сама текущая дата соответствует понедельнику, будет возвращена она).

Получив первый понедельник месяца, добавляем 21 или 28 дней, чтобы найти последний понедельник. Сколько дней необходимо добавить (21 или 28), определяет выражение CASE (строки 2–5). Оно добавляет к текущей дате 28 дней и проверяет, приводит ли это к переходу в следующий месяц. Выражение CASE осуществляет это следующим образом:

1. К значению, возвращенному функцией FIRST_MONDAY, добавляется 28.
2. С помощью функции TO_CHAR (PostgreSQL) или MONTH из результата выражения FIRST_MONDAY + 28 извлекается название получаемого месяца.

3. Результат шага 2 сравнивается со значением MTH из вложенного запроса. Значение MTH – это название текущего месяца, полученное в результате выполнения функции CURRENT_DATE. Если значения совпадают, следовательно, текущий месяц длинный и необходимо добавлять 28 дней; выражение CASE возвращает FIRST_MONDAY + 28. Если значения месяцев не совпадают, значит, при добавлении 28 дней мы выходим за рамки одного месяца, и выражение CASE возвращает FIRST_MONDAY + 21. Длительность наших месяцев такова, что проверять приходится только два возможных значения, 28 и 21. Это очень удобно.

Можно расширить решение, добавляя 7 и 14 дней, для поиска второго и третьего понедельников месяца соответственно.

Создание календаря

Задача

Требуется создать календарь на текущий месяц. Он должен быть отформатирован, как обычный календарь: семь столбцов в ширину и (как правило) пять строк вниз.

Решение

Решения будут выглядеть немного по-разному, но все они решают эту задачу одинаково: возвращают все дни текущего месяца и затем разделяют их на недели по одному из дней недели, создавая календарь.

Существуют разные форматы календарей. Например, команда *cal* UNIX форматирует неделю от воскресенья до субботы. В примерах данного рецепта используется стандартная нумерация недель ISO, поэтому удобнее всего будет создавать неделю, начиная с понедельника. Полностью разобравшись с решениями, вы поймете, что изменение формата календаря – это просто вопрос корректировки значений, определяемых ISO-номерами недель.

Как только мы начинаем использовать в SQL разные типы форматирования для создания удобных для чтения результатов, запросы становятся длиннее. Не позволяйте этим длинным запросам запутать вас. Представленные в данном рецепте запросы окажутся предельно простыми при разложении их на составляющие и выполнении одного за другим.

DB2

Чтобы возвратить все дни текущего месяца, используйте рекурсивный оператор WITH. Затем разбейте месяц на недели по выбранному дню с помощью выражений CASE и функций MAX:

```
 1   with x(dy,dm,mth,dw,wk)
 2     as (
 3 select (current_date -day(current_date) day +1 day) dy,
 4         day((current_date -day(current_date) day +1 day)) dm,
 5         month(current_date) mth,
 6         dayofweek(current_date -day(current_date) day +1 day) dw,
 7         week_iso(current_date -day(current_date) day +1 day) wk
 8   from t1
 9  union all
10 select dy+1 day, day(dy+1 day), mth,
11        dayofweek(dy+1 day), week_iso(dy+1 day)
12   from x
13  where month(dy+1 day) = mth
14  )
15  select max(case dw when 2 then dm end) as Mo,
16         max(case dw when 3 then dm end) as Tu,
17         max(case dw when 4 then dm end) as We,
18         max(case dw when 5 then dm end) as Th,
19         max(case dw when 6 then dm end) as Fr,
20         max(case dw when 7 then dm end) as Sa,
21         max(case dw when 1 then dm end) as Su
22    from x
23   group by wk
24   order by wk
```

Oracle

Чтобы возвратить все дни текущего месяца, используйте рекурсивный оператор CONNECT BY. Затем разбейте месяц на недели по выбранному дню с помощью выражений CASE и функций MAX:

```
 1  with x
 2    as (
 3  select *
 4    from (
 5  select to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
 6         to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
 7         to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
 8         to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
 9         to_char(sysdate,'mm') mth
10    from dual
11   connect by level <= 31
12         )
13   where curr_mth = mth
14  )
15  select max(case dw when 2 then dm end) Mo,
16         max(case dw when 3 then dm end) Tu,
17         max(case dw when 4 then dm end) We,
18         max(case dw when 5 then dm end) Th,
19         max(case dw when 6 then dm end) Fr,
20         max(case dw when 7 then dm end) Sa,
21         max(case dw when 1 then dm end) Su
```

```
22    from x
23   group by wk
24   order by wk
```

PostgreSQL

Чтобы возвратить все дни текущего месяца, используйте функцию GENERATE_SERIES. Затем разбейте месяц на недели по выбранному дню с помощью выражений CASE и функций MAX:

```
 1  select max(case dw when 2 then dm end) as Mo,
 2         max(case dw when 3 then dm end) as Tu,
 3         max(case dw when 4 then dm end) as We,
 4         max(case dw when 5 then dm end) as Th,
 5         max(case dw when 6 then dm end) as Fr,
 6         max(case dw when 7 then dm end) as Sa,
 7         max(case dw when 1 then dm end) as Su
 8    from (
 9  select *
10    from (
11  select cast(date_trunc('month',current_date) as date)+x.id,
12         to_char(
13            cast(
14      date_trunc('month',current_date)
15                 as date)+x.id,'iw') as wk,
16         to_char(
17            cast(
18      date_trunc('month',current_date)
19                 as date)+x.id,'dd') as dm,
20         cast(
21      to_char(
22         cast(
23   date_trunc('month',current_date)
24                 as date)+x.id,'d') as integer) as dw,
25         to_char(
26            cast(
27      date_trunc('month',current_date)
28                 as date)+x.id,'mm') as curr_mth,
29         to_char(current_date,'mm') as mth
30    from generate_series (0,31) x(id)
31         ) x
32   where mth = curr_mth
33         ) y
34   group by wk
35   order by wk
```

MySQL

Чтобы возвратить все дни текущего месяца, используйте таблицу T500. Затем разбейте месяц на недели по выбранному дню с помощью выражений CASE и функций MAX:

```
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from (
select date_format(dy,'%u') wk,
       date_format(dy,'%d') dm,
       date_format(dy,'%w')+1 dw
  from (
select adddate(x.dy,t500.id-1) dy,
       x.mth
  from (
select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
           adddate(current_date,
                   -dayofmonth(current_date)+1),
                   '%m') mth
  from t1
       ) x,
         t500
 where t500.id <= 31
   and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
       ) y
       ) z
 group by wk
 order by wk
```

SQL Server

Чтобы возвратить все дни текущего месяца, используйте рекурсивный оператор WITH. Затем разбейте месяц на недели по выбранному дню с помощью выражений CASE и функций MAX:

```
  with x(dy,dm,mth,dw,wk)
    as (
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(ww,dy)-1
            else datepart(ww,dy)
       end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
  from t1
       ) x
 union all
 select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
```

```
17          datepart(dw,dateadd(d,1,dy)),
18          case when datepart(dw,dateadd(d,1,dy)) = 1
19               then datepart(wk,dateadd(d,1,dy))-1
20               else datepart(wk,dateadd(d,1,dy))
21          end
22     from x
23    where datepart(m,dateadd(d,1,dy)) = mth
24  )
25  select max(case dw when 2 then dm end) as Mo,
26         max(case dw when 3 then dm end) as Tu,
27         max(case dw when 4 then dm end) as We,
28         max(case dw when 5 then dm end) as Th,
29         max(case dw when 6 then dm end) as Fr,
30         max(case dw when 7 then dm end) as Sa,
31         max(case dw when 1 then dm end) as Su
32    from x
33   group by wk
34   order by wk
```

Обсуждение

DB2

Первый шаг – возвратить все дни месяца, для которого создается календарь. Для этого используется рекурсивный оператор WITH (если WITH недоступен, можно применять сводную таблицу, например, T500, как в решении для MySQL). Кроме всех дней месяца (столбец под псевдонимом DM), понадобится возвратить разные составляющие каждой даты: порядковый номер дня недели (под псевдонимом DW), текущий месяц (под псевдонимом MTH) и ISO-номер недели для каждого дня месяца (под псевдонимом WK). Результаты, возвращаемые рекурсивным представлением X до проведения рекурсии (верхняя часть оператора UNION ALL), показаны ниже:

```
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1

DY          DM MTH         DW WK
----------- -- --- ---------- --
01-JUN-2005 01  06          4 22
```

Далее пошагово увеличиваем значение столбца DM (перебираем дни месяца), до тех пор пока не будут возвращены все дни данного месяца. Для каждого дня месяца при этом также будут получены соответствующий ему день недели и ISO-номер недели, в которую попадает данный день. Результаты частично показаны ниже:

```
with x(dy,dm,mth,dw,wk)
  as (
```

```
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1
 union all
 select dy+1 day, day(dy+1 day), mth,
        dayofweek(dy+1 day), week_iso(dy+1 day)
   from x
  where month(dy+1 day) = mth
)
select *
  from x
```

```
DY          DM MTH         DW WK
----------- -- --- ---------- --
01-JUN-2005 01  06          4 22
02-JUN-2005 02  06          5 22
...
21-JUN-2005 21  06          3 25
22-JUN-2005 22  06          4 25
...
30-JUN-2005 30  06          5 26
```

На данном этапе мы получаем все дни текущего месяца, а также следующие данные для каждого дня: двузначный номер дня месяца, двузначный номер месяца, однозначный номер дня недели (1–7 для Вс.–Сб.) и двузначный ISO-номер недели. Имея в своем распоряжении всю эту информацию, с помощью выражения CASE можно определить, на какой день недели выпадает каждое из значений столбца DM (каждый день месяца). Результаты частично показаны ниже:

```
with x(dy,dm,mth,dw,wk)
  as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1
 union all
 select dy+1 day, day(dy+1 day), mth,
        dayofweek(dy+1 day), week_iso(dy+1 day)
   from x
  where month(dy+1 day) = mth
)
select wk,
       case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
```

```
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from x

WK MO TU WE TH FR SA SU
-- -- -- -- -- -- -- --
22       01
22          02
22             03
22                04
22                   05
23 06
23    07
23       08
23          09
23             10
23                11
23                   12
```

Как видно из частично представленных результатов, каждый день недели возвращается в отдельной строке. Теперь осталось только сгруппировать дни по неделям и собрать все дни одной недели в одну строку. Чтобы возвратить все дни недели в одной строке, используйте агрегатную функцию MAX и группировку по значениям столбца WK (ISO-номеру недели). Чтобы правильно отформатировать календарь и обеспечить верное расположение дней, упорядочиваем результаты по WK. Окончательный результат приведен ниже:

```
with x(dy,dm,mth,dw,wk)
  as (
select (current_date -day(current_date) day +1 day) dy,
       day((current_date -day(current_date) day +1 day)) dm,
       month(current_date) mth,
       dayofweek(current_date -day(current_date) day +1 day) dw,
       week_iso(current_date -day(current_date) day +1 day) wk
  from t1
 union all
 select dy+1 day, day(dy+1 day), mth,
        dayofweek(dy+1 day), week_iso(dy+1 day)
   from x
  where month(dy+1 day) = mth
)
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from x
```

```
 group by wk
 order by wk
```

```
MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Oracle

В Oracle 9*i* Database начинаем с использования рекурсивного оператора CONNECT BY, с помощью которого получаем столько строк, сколько дней в заданном месяце. В более ранних версиях СУБД CONNECT BY не может таким образом использоваться. Вместо этого можно применить сводную таблицу, например T500, как в решении для MySQL.

Для каждого дня месяца понадобится дополнительно получить следующие данные: порядковый номер дня месяца (под псевдонимом DM), день недели (под псевдонимом DW), текущий месяц (под псевдонимом MTH) и ISO-номер недели для каждого дня месяца (под псевдонимом WK). Результаты, возвращаемые представлением X оператора WITH для первого дня текущего месяца, показаны ниже:

```
select trunc(sysdate,'mm') dy,
       to_char(trunc(sysdate,'mm'),'dd') dm,
       to_char(sysdate,'mm') mth,
       to_number(to_char(trunc(sysdate,'mm'),'d')) dw,
       to_char(trunc(sysdate,'mm'),'iw') wk
  from dual
```

```
DY          DM MT         DW WK
----------- -- -- ---------- --
01-JUN-2005 01 06          4 22
```

Далее пошагово увеличиваем значение DM (перебираем дни месяца) до тех пор, пока не будут получены все дни текущего месяца. Для каждого дня месяца также возвращаем соответствующий ему день недели и ISO-номер недели, в которую попадает данный день. Результаты частично показаны ниже (полная дата для каждого дня приведена для ясности):

```
with x
  as (
select *
  from (
select trunc(sysdate,'mm')+level-1 dy,
       to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
```

```
       to_char(sysdate,'mm') mth
  from dual
 connect by level <= 31
       )
 where curr_mth = mth
)
select *
  from x

DY          WK DM         DW CU MT
----------- -- -- ---------- -- --
01-JUN-2005 22 01          4 06 06
02-JUN-2005 22 02          5 06 06
...
21-JUN-2005 25 21          3 06 06
22-JUN-2005 25 22          4 06 06
...
30-JUN-2005 26 30          5 06 06
```

Здесь мы получаем все дни текущего месяца, каждый из которых выведен в отдельной строке. В каждой строке представлены следующие данные: двузначный номер дня месяца, двузначный номер месяца, однозначный номер дня недели (1–7 для Вс.–Сб.) и двузначный ISO-номер недели. Имея в своем распоряжении всю эту информацию, с помощью выражения CASE можно определить, на какой день недели выпадает каждое из значений столбца DM (каждый день месяца). Результаты частично показаны ниже:

```
with x
  as (
select *
  from (
select trunc(sysdate,'mm')+level-1 dy,
       to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
 connect by level <= 31
       )
 where curr_mth = mth
)
select wk,
       case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
```

```
  from x

WK MO TU WE TH FR SA SU
-- -- -- -- -- -- -- --
22       01
22          02
22             03
22                04
22                   05
23 06
23    07
23       08
23          09
23             10
23                11
23                   12
```

Как видно из частично представленного результата, каждый день каждой недели возвращается в отдельной строке, но номер дня располагается в одном из семи столбцов в соответствии с днем недели. Теперь остается только собрать все дни одной недели в одну строку. Чтобы возвратить все дни недели в одной строке, используем агрегатную функцию MAX и группировку по значениям столбца WK (ISO-номеру недели). Чтобы обеспечить верное расположение дней, упорядочиваем результаты по WK. Окончательный результат приведен ниже:

```
with x
  as (
select *
  from (
select to_char(trunc(sysdate,'mm')+level-1,'iw') wk,
       to_char(trunc(sysdate,'mm')+level-1,'dd') dm,
       to_number(to_char(trunc(sysdate,'mm')+level-1,'d')) dw,
       to_char(trunc(sysdate,'mm')+level-1,'mm') curr_mth,
       to_char(sysdate,'mm') mth
  from dual
 connect by level <= 31
       )
 where curr_mth = mth
)
select max(case dw when 2 then dm end) Mo,
       max(case dw when 3 then dm end) Tu,
       max(case dw when 4 then dm end) We,
       max(case dw when 5 then dm end) Th,
       max(case dw when 6 then dm end) Fr,
       max(case dw when 7 then dm end) Sa,
       max(case dw when 1 then dm end) Su
  from x
 group by wk
 order by wk

MO TU WE TH FR SA SU
```

```
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

PostgreSQL

С помощью функции GENERATE_SERIES возвращаем каждый день месяца в отдельной строке. Если используемая версия PostgreSQL не поддерживает GENERATE_SERIES, можно применить сводную таблицу, как показано в решении для MySQL.

Для каждого дня месяца возвращаем следующую информацию: порядковый номер дня месяца (под псевдонимом DM), день недели (под псевдонимом DW), текущий месяц, с которым мы работаем (под псевдонимом MTH), и ISO-номер недели (под псевдонимом WK). Форматирование и явное приведение делают это решение тяжелым для восприятия, но на самом деле оно довольно простое. Результаты, возвращаемые вложенным запросом X, частично показаны ниже:

```
select cast(date_trunc('month',current_date) as date)+x.id as dy,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'iw') as wk,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'dd') as dm,
       cast(
    to_char(
       cast(
 date_trunc('month',current_date)
               as date)+x.id,'d') as integer) as dw,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'mm') as curr_mth,
       to_char(current_date,'mm') as mth
  from generate_series (0,31) x(id)
```

```
DY          WK DM         DW CU MT
----------- -- -- ---------- -- --
01-JUN-2005 22 01          4 06 06
02-JUN-2005 22 02          5 06 06
...
21-JUN-2005 25 21          3 06 06
22-JUN-2005 25 22          4 06 06
...
30-JUN-2005 26 30          5 06 06
```

Обратите внимание, что для каждого дня месяца также возвращаются информация о том, какой это день недели, и соответствующий ISO-номер недели. Чтобы получить дни только интересующего нас месяца, возвращаем строки, где CURR_MTH = MTH (месяц, к которому относится день, должен быть текущим месяцем). На данном этапе для каждого дня текущего месяца возвращаются: двузначный номер дня месяца, двузначный номер месяца, однозначный номер дня недели (1–7 для Вс.–Сб.) и двузначный ISO-номер недели. Следующий шаг – с помощью выражения CASE определить, на какой день недели выпадает каждое из значений столбца DM (каждый день месяца). Результаты частично показны ниже:

```
select case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from (
select *
  from (
select cast(date_trunc('month',current_date) as date)+x.id,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'iw') as wk,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'dd') as dm,
       cast(
    to_char(
       cast(
 date_trunc('month',current_date)
               as date)+x.id,'d') as integer) as dw,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'mm') as curr_mth,
       to_char(current_date,'mm') as mth
  from generate_series (0,31) x(id)
       ) x
 where mth = curr_mth
       ) y
```

```
WK MO TU WE TH FR SA SU
-- -- -- -- -- -- -- --
22       01
22          02
22             03
```

```
22                 04
22                    05
23 06
23    07
23       08
23          09
23             10
23                11
23                   12
```

Как видим из частичного вывода, каждый день недели возвращен в отдельной строке, и его номер располагается в столбце, соответствующем дню недели, на который данный день выпадает. Теперь наша задача – свести все дни одной недели в одну строку. Для этого используем агрегатную функцию MAX и группировку строк по столбцу WK (ISO-номеру недели). В результате все дни каждой недели будут выведены в одной строке, как и в обычных календарях. Чтобы обеспечить правильное расположение дней, упорядочиваем результаты по WK. Окончательный вывод показан ниже:

```
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from (
select *
  from (
select cast(date_trunc('month',current_date) as date)+x.id,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'iw') as wk,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'dd') as dm,
       cast(
    to_char(
       cast(
 date_trunc('month',current_date)
               as date)+x.id,'d') as integer) as dw,
       to_char(
          cast(
    date_trunc('month',current_date)
               as date)+x.id,'mm') as curr_mth,
       to_char(current_date,'mm') as mth
  from generate_series (0,31) x(id)
       ) x
```

```
 where mth = curr_mth
       ) y
 group by wk
 order by wk
```

```
MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

MySQL

Первый шаг – возвратить в отдельной строке каждый день месяца, для которого создается календарь. Для этого выполняем запрос к таблице T500. Добавляя каждое возвращаемое T500 значение к первому дню месяца, получаем все дни месяца.

Для каждой даты необходимо возвратить следующие данные: порядковый номер дня месяца (под псевдонимом DM), день недели (под псевдонимом DW), текущий месяц, с которым мы работаем (под псевдонимом MTH), и ISO-номер недели (под псевдонимом WK). Вложенный запрос X возвращает первый день и двузначный номер текущего месяца. Результаты показаны ниже:

```
select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
           adddate(current_date,
                   -dayofmonth(current_date)+1),
                   '%m') mth
  from t1
```

```
DY          MT
----------- --
01-JUN-2005 06
```

Следующий шаг – получить все дни месяца, начиная с первого. Обратите внимание, что для каждого дня месяца возвращаются также соответствующий ему день недели и ISO-номер недели. Чтобы гарантировать выбор дней только интересующего нас месяца, возвращаем строки, где месяц, к которому относится день, является текущим месяцем. Строки, возвращаемые вложенным запросом Y, частично показаны ниже:

```
select date_format(dy,'%u') wk,
       date_format(dy,'%d') dm,
       date_format(dy,'%w')+1 dw
  from (
select adddate(x.dy,t500.id-1) dy,
       x.mth
  from (
select adddate(current_date,-dayofmonth(current_date)+1) dy,
```

```
       date_format(
           adddate(current_date,
                   -dayofmonth(current_date)+1),
                   '%m') mth
  from t1
       ) x,
         t500
 where t500.id <= 31
   and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
       ) y
```

```
WK DM         DW
-- -- ----------
22 01          4
22 02          5
...
25 21          3
25 22          4
...
26 30          5
```

Теперь для каждого дня текущего месяца мы имеем: двузначный номер дня месяца (DM), однозначный номер дня недели (DW) и двузначный ISO-номер недели (WK). Эту информацию можно использовать в выражении CASE для установления соответствия между каждым значением DM (каждым днем месяца) и днем недели. Результаты частично показаны ниже:

```
select case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from (
select date_format(dy,'%u') wk,
       date_format(dy,'%d') dm,
       date_format(dy,'%w')+1 dw
  from (
select adddate(x.dy,t500.id-1) dy,
       x.mth
  from (
select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
           adddate(current_date,
                   -dayofmonth(current_date)+1),
                   '%m') mth
  from t1
       ) x,
         t500
 where t500.id <= 31
```

```
   and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
       ) y
       ) z

WK MO TU WE TH FR SA SU
-- -- -- -- -- -- -- --
22       01
22          02
22             03
22                04
22                   05
23 06
23    07
23       08
23          09
23             10
23                11
23                   12
```

Как видно из частичного вывода, каждый день недели возвращен в отдельной строке. В каждой строке номер дня располагается в столбце, соответствующем дню недели, на который данный день выпадает. Теперь наша задача – свести все дни одной недели в одну строку. Для этого используем агрегатную функцию MAX и группировку строк по столбцу WK (ISO-номеру недели). Чтобы обеспечить правильное расположение дней, упорядочиваем результаты по WK. Окончательный вывод показан ниже:

```
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from (
select date_format(dy,'%u') wk,
       date_format(dy,'%d') dm,
       date_format(dy,'%w')+1 dw
  from (
select adddate(x.dy,t500.id-1) dy,
       x.mth
  from (
select adddate(current_date,-dayofmonth(current_date)+1) dy,
       date_format(
           adddate(current_date,
                   -dayofmonth(current_date)+1),
                   '%m') mth
  from t1
       ) x,
         t500
 where t500.id <= 31
```

```
   and date_format(adddate(x.dy,t500.id-1),'%m') = x.mth
       ) y
       ) z
 group by wk
 order by wk
```

```
MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

SQL Server

Начинаем с возвращения каждого дня месяца в отдельной строке. Сделать это можно с помощью рекурсивного оператора WITH или, если используемая версия SQL Server не поддерживает рекурсивный WITH, применяя сводную таблицу, как в решении для MySQL. В каждой возвращаемой строке содержатся следующие элементы: порядковый номер дня месяца (под псевдонимом DM), день недели (под псевдонимом DW), текущий месяц (под псевдонимом MTH) и ISO-номер недели (под псевдонимом WK). Рекурсивное представление X до выполнения рекурсии (верхняя часть оператора UNION ALL) показано ниже:

```
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(ww,dy)-1
            else datepart(ww,dy)
       end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
  from t1
       ) x
```

```
DY          DM MTH          DW WK
----------- -- --- ---------- --
01-JUN-2005  1   6           4 23
```

Следующий этап – пошагово увеличиваем значение DM до тех пор, пока не будут получены все дни месяца. Для каждого дня возвращаем также соответствующие ему день недели и ISO-номер недели. Результаты частично показаны ниже:

```
  with x(dy,dm,mth,dw,wk)
    as (
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
```

```
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(ww,dy)-1
            else datepart(ww,dy)
       end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
  from t1
       ) x
 union all
 select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
        datepart(dw,dateadd(d,1,dy)),
        case when datepart(dw,dateadd(d,1,dy)) = 1
             then datepart(wk,dateadd(d,1,dy))-1
             else datepart(wk,dateadd(d,1,dy))
        end
   from x
  where datepart(m,dateadd(d,1,dy)) = mth
)
select *
  from x
```

```
DY          DM MTH         DW WK
----------- -- --- ---------- --
01-JUN-2005 01  06          4 23
02-JUN-2005 02  06          5 23
...
21-JUN-2005 21  06          3 26
22-JUN-2005 22  06          4 26
...
30-JUN-2005 30  06          5 27
```

Теперь для каждого дня текущего месяца мы имеем: двузначный номер дня месяца, двузначный номер месяца, однозначный номер дня недели (1–7 для Вс.–Сб.) и двузначный ISO-номер недели.

С помощью выражения CASE устанавливаем соответствие между каждым значением DM (каждым днем месяца) и днем недели, на который он выпадает. Результаты частично показаны ниже:

```
  with x(dy,dm,mth,dw,wk)
    as (
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(ww,dy)-1
            else datepart(ww,dy)
       end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
```

```
  from t1
       ) x
 union all
 select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
        datepart(dw,dateadd(d,1,dy)),
        case when datepart(dw,dateadd(d,1,dy)) = 1
             then datepart(wk,dateadd(d,1,dy))-1
             else datepart(wk,dateadd(d,1,dy))
        end
   from x
  where datepart(m,dateadd(d,1,dy)) = mth
)
select case dw when 2 then dm end as Mo,
       case dw when 3 then dm end as Tu,
       case dw when 4 then dm end as We,
       case dw when 5 then dm end as Th,
       case dw when 6 then dm end as Fr,
       case dw when 7 then dm end as Sa,
       case dw when 1 then dm end as Su
  from x
```

```
WK MO TU WE TH FR SA SU
-- -- -- -- -- -- -- --
22       01
22          02
22             03
22                04
22                   05
23 06
23    07
23       08
23          09
23             10
23                11
23                   12
```

Каждый день возвращается в отдельной строке. В каждой строке номер дня месяца располагается в столбце, соответствующем дню недели, на который этот день выпадает. Теперь необходимо собрать все дни недели в одну строку. Для этого группируем строки по столбцу WK (ISO-номеру недели) и применяем функцию MAX к разным столбцам. В результате получаем формат календаря, такой как показан ниже:

```
  with x(dy,dm,mth,dw,wk)
    as (
select dy,
       day(dy) dm,
       datepart(m,dy) mth,
       datepart(dw,dy) dw,
       case when datepart(dw,dy) = 1
            then datepart(ww,dy)-1
            else datepart(ww,dy)
```

```
       end wk
  from (
select dateadd(day,-day(getdate())+1,getdate()) dy
  from t1
       ) x
 union all
 select dateadd(d,1,dy), day(dateadd(d,1,dy)), mth,
        datepart(dw,dateadd(d,1,dy)),
        case when datepart(dw,dateadd(d,1,dy)) = 1
             then datepart(wk,dateadd(d,1,dy))-1
             else datepart(wk,dateadd(d,1,dy))
        end
   from x
  where datepart(m,dateadd(d,1,dy)) = mth
)
select max(case dw when 2 then dm end) as Mo,
       max(case dw when 3 then dm end) as Tu,
       max(case dw when 4 then dm end) as We,
       max(case dw when 5 then dm end) as Th,
       max(case dw when 6 then dm end) as Fr,
       max(case dw when 7 then dm end) as Sa,
       max(case dw when 1 then dm end) as Su
  from x
 group by wk
 order by wk
```

```
MO TU WE TH FR SA SU
-- -- -- -- -- -- --
      01 02 03 04 05
06 07 08 09 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30
```

Получение дат начала и конца кварталов года

Задача

Требуется возвратить даты начала и конца каждого из четырех кварталов данного года.

Решение

В году четыре квартала, таким образом, требуется создать четыре строки. После создания необходимого количества строк возвращаем даты начала и окончания кварталов с помощью функций для работы с датами, предоставляемых СУБД. Цель – сформировать следующее результирующее множество (текущий год выбирается произвольно):

```
QTR Q_START     Q_END
--- ----------- -----------
```

```
1 01-JAN-2005 31-MAR-2005
2 01-APR-2005 30-JUN-2005
3 01-JUL-2005 30-SEP-2005
4 01-OCT-2005 31-DEC-2005
```

DB2

Для формирования четырех строк используйте таблицу EMP и ранжирующую функцию ROW_NUMBER OVER. В качестве альтернативы строки можно создать с помощью оператора WITH (как во многих рецептах) или запроса к таблице, содержащей не менее четырех строк. В следующем решении используется функция ROW_NUMBER OVER:

```
select quarter(dy-1 day) QTR,
       dy-3 month Q_start,
       dy-1 day Q_end
  from (
select (current_date -
          (dayofyear(current_date)-1) day
            + (rn*3) month) dy
  from (
select row_number()over() rn
  from emp
 fetch first 4 rows only
       ) x
       ) y
```

Oracle

С помощью функции ADD_MONTHS найдите начальную и конечную даты каждого квартала. Для представления квартала, которому соответствуют та или иная начальная и конечная даты, используйте псевдостолбец ROWNUM (номер строки). В следующем решении для формирования четырех строк используется таблица EMP:

```
select rownum qtr,
       add_months(trunc(sysdate,'y'),(rownum-1)*3) q_start,
       add_months(trunc(sysdate,'y'),rownum*3)-1 q_end
  from emp
 where rownum <= 4
```

PostgreSQL

Используйте функцию GENERATE_SERIES для получения требуемых четырех кварталов. С помощью функции DATE_TRUNC от дат, возвращенных для каждого квартала, отсеките день, оставляя лишь год и месяц. Используя функцию TO_CHAR, установите соответствие между кварталом и начальной и конечной датами:

```
select to_char(dy,'Q') as QTR,
       date(
         date_trunc('month',dy)-(2*interval '1 month')
       ) as Q_start,
```

```
5         dy as Q_end
6    from (
7  select date(dy+((rn*3) * interval '1 month'))-1 as dy
8    from (
9  select rn, date(date_trunc('year',current_date)) as dy
10   from generate_series(1,4) gs(rn)
11        ) x
12        ) y
```

MySQL

Используйте таблицу T500, чтобы получить четрые строки (по одной для каждого квартала). Даты начала и конца каждого квартала вычисляются с помощью функций DATE_ADD и ADDDATE. Используя функцию QUARTER, установите соответствие между кварталом и начальной и конечной датами:

```
1  select quarter(adddate(dy,-1)) QTR,
2         date_add(dy,interval -3 month) Q_start,
3         adddate(dy,-1) Q_end
4    from (
5  select date_add(dy,interval (3*id) month) dy
6    from (
7  select id,
8         adddate(current_date,-dayofyear(current_date)+1) dy
9    from t500
10   where id <= 4
11        ) x
12        ) y
```

SQL Server

Четыре строки сформируйте с помощью рекурсивного оператора WITH. Посредством функции DATEADD найдите начальную и конечную даты. Используя функцию DATEPART, установите соответствие между кварталом и начальной и конечной датами:

```
1   with x (dy,cnt)
2     as (
3  select dateadd(d,-(datepart(dy,getdate())-1),getdate()),
4         1
5    from t1
6   union all
7  select dateadd(m,3,dy), cnt+1
8    from x
9   where cnt+1 <= 4
10 )
11 select datepart(q,dateadd(d,-1,dy)) QTR,
12        dateadd(m,-3,dy) Q_start,
13        dateadd(d,-1,dy) Q_end
14   from x
15  order by 1
```

Обсуждение

DB2

Первый шаг – получить четыре строки (со значениями от 1 до 4), по одной для каждого квартала года. Вложенный запрос X с помощью ранжирующей функции ROW_NUMBER OVER и оператора FETCH FIRST возвращает из таблицы EMP всего четыре строки. Результаты показаны ниже:

```
select row_number()over() rn
  from emp
 fetch first 4 rows only
```

```
RN
--
 1
 2
 3
 4
```

Следующий шаг – найти первый день года и добавить к нему *n* месяцев, где *n* – значение столбца RN, увеличенное в три раза (к первому дню года добавляется 3, 6, 9 и 12 месяцев). Результаты представлены ниже:

```
select (current_date -
         (dayofyear(current_date)-1) day
           + (rn*3) month) dy
  from (
select row_number()over() rn
  from emp
 fetch first 4 rows only
       ) x
```

```
DY
-----------
01-APR-2005
01-JUL-2005
01-OCT-2005
01-JAN-2005
```

На данный момент в столбце DY выведены даты, соответствующие следующему дню после окончания каждого квартала. Следующий шаг – получить даты начала и конца каждого квартала. Чтобы получить дату окончания квартала, вычитаем один день из DY. Чтобы получить дату начала квартала, вычитаем три месяца из DY. Используем функцию QUARTER с параметром DY минус 1 (дата окончания каждого квартала), чтобы определить, какому кварталу соответствуют полученные начальная и конечная даты.

Oracle

Сочетание функций ROWNUM, TRUNC и ADD_MONTHS значительно упрощает решение. Чтобы найти дату начала каждого квартала, просто добавляем *n* месяцев к первому дню года, где *n* – это (ROWNUM – 1)×3 (что в результате дает 0, 3, 6, 9). Чтобы найти дату окончания каждого квартала, добавляем *n* месяцев к первому дню года, где *n* – это ROWNUM, умноженное на 3, и вычитаем один день. Отметим, что при работе с кварталами может быть полезным использование функций TO_CHAR и/или TRUNC с форматной маской 'q'.

PostgreSQL

Первый шаг – с помощью функции DATE_TRUNC, исходя из текущей даты, получить первый день текущего года. Затем добавить *n* месяцев, где *n* – значение столбца RN (RN содержит значения, возвращенные функцией GENERATE_SERIES), умноженное на три, и вычесть один день. Результаты показаны ниже:

```
select date(dy+((rn*3) * interval '1 month'))-1 as dy
  from (
select rn, date(date_trunc('year',current_date)) as dy
  from generate_series(1,4) gs(rn)
       ) x
```

```
DY
-----------
31-MAR-2005
30-JUN-2005
30-SEP-2005
31-DEC-2005
```

Теперь после получения конечных дат всех кварталов остается последний шаг – найти начальные даты. Для этого из каждого значения DY вычитаем два месяца и с помощью функции DATE_TRUNC переходим к первому дню полученного месяца. Применяя функцию TO_CHAR к конечной дате каждого квартала (DY), определяем, какому кварталу соответствуют начальная и конечная даты.

MySQL

Первый шаг – найти первый день года, используя функции ADDDATE и DAYOFYEAR. Затем с помощью функции DATE_ADD добавить *n* месяцев к первому дню года, где *n* – значение T500.ID, умноженное на три. Результаты представлены ниже:

```
select date_add(dy,interval (3*id) month) dy
  from (
select id,
       adddate(current_date,-dayofyear(current_date)+1) dy
  from t500
 where id <= 4
```

```
      ) x

DY
-----------
01-APR-2005
01-JUL-2005
01-OCT-2005
01-JAN-2005
```

На данный момент возвращены даты, соответствующие следующему дню после окончания каждого квартала. Чтобы найти последний день каждого квартала, просто вычтем по одному дню из всех значений DY. Следующий шаг – найти даты начала кварталов, для этого вычитаем по три месяца из каждого значения DY. Применяем функцию QUARTER к конечной дате каждого квартала, чтобы определить, какому кварталу соответствуют начальная и конечная даты.

SQL Server

Первый шаг – найти первый день года. Затем, используя функцию DATEADD, рекурсивно добавить по *n* месяцев, где *n* – номер текущей итерации, умноженный на три (всего выполняется четыре итерации, следовательно, добавляется 3 * 1 месяцев, 3 * 2 месяцев и т. д.). Результаты показаны ниже:

```
with x (dy,cnt)
   as (
select dateadd(d,-(datepart(dy,getdate())-1),getdate()),
       1
  from t1
 union all
select dateadd(m,3,dy), cnt+1
  from x
 where cnt+1 <= 4
)
select dy
  from x

DY
-----------
01-APR-2005
01-JUL-2005
01-OCT-2005
01-JAN-2005
```

Значения DY соответствуют следующим дням после окончания каждого квартала. Чтобы получить даты окончания кварталов, просто вычитаем один день из значений DY, используя функцию DATEADD. Чтобы найти дату начала каждого квартала, с помощью функции DATEADD вычитаем три месяца из каждого значения DY. Применяя функцию DATEPART к конечным датам кварталов, определяем, к каким кварталам относятся полученные начальные и конечные даты.

Определение дат начала и окончания заданного квартала

Задача

Год и квартал заданы в формате YYYYQ (четыре разряда – год, один разряд – квартал), требуется получить даты начала и окончания квартала.

Решение

Ключ к решению – применить к значению YYYYQ функцию вычисления остатка от деления. (Поскольку год представлен четырьмя разрядами, вместо нахождения остатка от деления можно просто извлечь последний разряд как подстроку.) Получив номер квартала, просто умножаем это значение на 3, чтобы найти месяц окончания квартала. В представленных решениях вложенный запрос X возвращает все четыре сочетания года и кварталов. Результирующее множество вложенного запроса X следующее:

```
select 20051 as yrq from t1 union all
select 20052 as yrq from t1 union all
select 20053 as yrq from t1 union all
select 20054 as yrq from t1
```

```
   YRQ
------
 20051
 20052
 20053
 20054
```

DB2

Используйте функцию SUBSTR, чтобы возвратить год из вложенного запроса X. С помощью функции MOD определите искомый квартал:[1]

```
 1  select (q_end-2 month) q_start,
 2         (q_end+1 month)-1 day q_end
 3    from (
 4  select date(substr(cast(yrq as char(4)),1,4) ||'-'||
 5         rtrim(cast(mod(yrq,10)*3 as char(2))) ||'-1') q_end
 6    from (
 7  select 20051 yrq from t1 union all
 8  select 20052 yrq from t1 union all
 9  select 20053 yrq from t1 union all
```

[1] В решениях для этой СУБД и всех следующих можно получить год, разделив значение YYYYQ на 10; этот вариант ближе к способу получения квартала путем вычисления остатка от деления, чем использование функции SUBSTR. – *Примеч. науч. ред.*

```
10  select 20054 yrq from t1
11         ) x
12         ) y
```

Oracle

Используйте функцию SUBSTR, чтобы возвратить год из вложенного запроса X. С помощью функции MOD определите искомый квартал:

```
 1  select add_months(q_end,-2) q_start,
 2         last_day(q_end) q_end
 3    from (
 4  select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
 5    from (
 6  select 20051 yrq from dual union all
 7  select 20052 yrq from dual union all
 8  select 20053 yrq from dual union all
 9  select 20054 yrq from dual
10         ) x
11         ) y
```

PostgreSQL

Используйте функцию SUBSTR, чтобы возвратить год из вложенного запроса X. С помощью функции MOD определите искомый квартал:

```
 1  select date(q_end-(2*interval '1 month')) as q_start,
 2         date(q_end+interval '1 month'-interval '1 day') as q_end
 3    from (
 4  select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') as q_end
 5    from (
 6  select 20051 as yrq from t1 union all
 7  select 20052 as yrq from t1 union all
 8  select 20053 as yrq from t1 union all
 9  select 20054 as yrq from t1
10         ) x
11         ) y
```

MySQL

Используйте функцию SUBSTR, чтобы возвратить год из вложенного запроса X. С помощью функции MOD определите искомый квартал:

```
 1  select date_add(
 2          adddate(q_end,-day(q_end)+1),
 3                  interval -2 month) q_start,
 4         q_end
 5    from (
 6  select last_day(
 7      str_to_date(
 8           concat(
 9           substr(yrq,1,4),mod(yrq,10)*3),'%Y%m')) q_end
10    from (
11  select 20051 as yrq from t1 union all
```

```
12  select 20052 as yrq from t1 union all
13  select 20053 as yrq from t1 union all
14  select 20054 as yrq from t1
15         ) x
16         ) y
```

SQL Server

Используйте функцию SUBSTRING, чтобы возвратить год из вложенного запроса X. С помощью функции вычисления остатка от деления (%) определите искомый квартал:

```
1  select dateadd(m,-2,q_end) q_start,
 2         dateadd(d,-1,dateadd(m,1,q_end)) q_end
 3    from (
 4  select cast(substring(cast(yrq as varchar),1,4)+'-'+
 5         cast(yrq%10*3 as varchar)+'-1' as datetime) q_end
 6    from (
 7  select 20051 yrq from t1 union all
 8  select 20052 yrq from t1 union all
 9  select 20053 yrq from t1 union all
10  select 20054 yrq from t1
11         ) x
12         ) y
```

Обсуждение

DB2

Первый шаг – найти год и квартал, с которыми будем работать. Извлекаем подстроку года из вложенного запроса X (X.YRQ), используя функцию SUBSTR. Чтобы получить квартал, находим остаток от деления YRQ на 10. Вычислив номер квартала, умножаем его на 3 и получаем последний месяц квартала. Результаты представлены ниже:

```
select substr(cast(yrq as char(4)),1,4) yr,
       mod(yrq,10)*3 mth
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
       ) x
```

```
YR      MTH
---- ------
2005      3
2005      6
2005      9
2005     12
```

На данный момент мы имеем год и последний месяц каждого квартала. Используя эти значения, получим необходимые даты, в частности,

первый день последнего месяца каждого квартала. С помощью оператора конкатенации «||» объединим год и месяц и затем с помощью функции DATE преобразуем полученное значение в дату:

```
select date(substr(cast(yrq as char(4)),1,4) ||'-'||
       rtrim(cast(mod(yrq,10)*3 as char(2))) ||'-1') q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
       ) x
```

```
Q_END
-----------
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Значения столбца Q_END – первый день последнего месяца каждого квартала. Чтобы получить последний день месяца, добавим один месяц к значению Q_END и вычтем один день. Чтобы найти дату начала каждого квартала, вычтем из Q_END два месяца.

Oracle

Первый шаг – найти год и квартал, с которыми будем работать. Извлекаем подстроку года из вложенного запроса X (X.YRQ), используя функцию SUBSTR. Чтобы получить квартал, находим остаток от деления YRQ на 10. Вычислив номер квартала, умножаем его на 3 и получаем последний месяц квартала. Результаты представлены ниже:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
       ) x
```

```
YR      MTH
---- ------
2005      3
2005      6
2005      9
2005     12
```

На данный момент мы имеем год и последний месяц каждого квартала. Используя эти значения, получим необходимые даты, в частности, первый день последнего месяца каждого квартала. С помощью оператора конкатенации «||» объединим год и месяц и затем с помощью функции TO_DATE преобразуем полученное значение в дату:

```
select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
       ) x
```

```
Q_END
-----------
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Значения столбца Q_END – первый день последнего месяца каждого квартала. Чтобы получить последний день месяца, применим к значениям Q_END функцию LAST_DAY. Чтобы найти дату начала каждого квартала, вычтем из Q_END два месяца, используя функцию ADD_MONTHS.

PostgreSQL

Первый шаг – найти год и квартал, с которыми будем работать. Извлекаем подстроку года из вложенного запроса X (X.YRQ), используя функцию SUBSTR. Чтобы получить квартал, находим остаток от деления YRQ на 10. Вычислив номер квартала, умножаем его на 3 и получаем последний месяц квартала. Результаты представлены ниже:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
       ) x
```

```
YR      MTH
---- ------
2005      3
2005      6
2005      9
2005     12
```

На данный момент мы имеем год и последний месяц каждого квартала. Используя эти значения, получим необходимые даты, в частности, первый день последнего месяца каждого квартала. С помощью оператора конкатенации «||» объединим год и месяц и затем с помощью функции TO_DATE преобразуем полученное значение в дату:

```
select to_date(substr(yrq,1,4)||mod(yrq,10)*3,'yyyymm') q_end
  from (
select 20051 yrq from dual union all
```

```
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
       ) x
```

```
Q_END
-----------
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Значения столбца Q_END – первый день последнего месяца каждого квартала. Чтобы получить последний день месяца, добавим к значениям Q_END один месяц и вычтем один день. Чтобы найти дату начала каждого квартала, вычтем два месяца из Q_END. Представим конечные результаты как даты.

MySQL

Первый шаг – найти год и квартал, с которыми будем работать. Извлекаем подстроку года из вложенного запроса X (X.YRQ), используя функцию SUBSTR. Чтобы получить квартал, находим остаток от деления YRQ на 10. Вычислив номер квартала, умножаем его на 3 и получаем последний месяц квартала. Результаты представлены ниже:

```
select substr(yrq,1,4) yr, mod(yrq,10)*3 mth
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
       ) x
```

```
YR      MTH
---- ------
2005      3
2005      6
2005      9
2005     12
```

На данный момент мы имеем год и последний месяц каждого квартала. Используя эти значения, получим необходимые даты, в частности, последний день каждого квартала. С помощью функции CONCAT объединим год и месяц и затем с помощью функции STR_TO_DATE преобразуем полученное значение в дату. Последний день каждого квартала найдем, применяя функцию LAST_DAY:

```
select last_day(
    str_to_date(
         concat(
         substr(yrq,1,4),mod(yrq,10)*3),'%Y%m')) q_end
  from (
```

```
select 20051 as yrq from t1 union all
select 20052 as yrq from t1 union all
select 20053 as yrq from t1 union all
select 20054 as yrq from t1
       ) x
```

```
Q_END
-----------
31-MAR-2005
30-JUN-2005
30-SEP-2005
31-DEC-2005
```

Поскольку мы уже получили даты окончания кварталов, осталось только найти даты их начала. Используем функцию DAY, чтобы получить день месяца, на который выпадает окончание каждого из кварталов, и с помощью функции ADDDATE вычтем полученное значение из Q_END, чтобы получить последний день предыдущего месяца. Добавим один день и найдем первый день последнего месяца каждого квартала. Последний шаг – посредством функции DATE_ADD вычитаем два месяца из первого дня последнего месяца каждого квартала и получаем дату начала каждого квартала.

SQL Server

Первый шаг – найти год и квартал, с которыми будем работать. Используя функцию SUBSTR, извлекаем подстроку года из вложенного запроса X (X.YRQ). Чтобы получить квартал, находим остаток от деления YRQ на 10. Вычислив номер квартала, умножаем его на 3 и получаем последний месяц квартала. Результаты представлены ниже:

```
select substring(yrq,1,4) yr, yrq%10*3 mth
  from (
select 20051 yrq from dual union all
select 20052 yrq from dual union all
select 20053 yrq from dual union all
select 20054 yrq from dual
       ) x
```

```
YR      MTH
---- ------
2005      3
2005      6
2005      9
2005     12
```

На данный момент мы имеем год и последний месяц каждого квартала. Используя эти значения, получим необходимые даты, в частности, первый день последнего месяца каждого квартала. С помощью оператора конкатенации «+» объединим год и месяц и затем с помощью функции CAST представим полученное значение как дату:

```
select cast(substring(cast(yrq as varchar),1,4)+'-'+
       cast(yrq%10*3 as varchar)+'-1' as datetime) q_end
  from (
select 20051 yrq from t1 union all
select 20052 yrq from t1 union all
select 20053 yrq from t1 union all
select 20054 yrq from t1
       ) x
```

```
Q_END
-----------
01-MAR-2005
01-JUN-2005
01-SEP-2005
01-DEC-2005
```

Значения столбца Q_END – первый день последнего месяца каждого квартала. Чтобы получить последний день месяца, прибавляем один месяц к значению Q_END и вычитаем один день, используя функцию DATEADD. Чтобы найти дату начала каждого квартала, вычитаем два месяца из Q_END, используя функцию DATEADD.

Дополнение отсутствующих дат

Задача

Требуется вывести все даты (месяц, неделю, год) внутри заданного интервала времени, каждую в отдельной строке. Такие наборы строк часто используются для формирования итоговых отчетов. Например, необходимо подсчитать, сколько служащих было принято на работу в каждом месяце каждого года, когда производился прием на работу. Проверив даты приема на работу всех служащих, выясняем, что людей нанимали с 1980 по 1983 год.

```
select distinct
       extract(year from hiredate) as year
  from emp
```

```
YEAR
-----
 1980
 1981
 1982
 1983
```

Требуется определить, сколько сотрудников принималось на работу каждый месяц в период с 1980 по 1983 г. Ниже частично представлено результирующее множество, которое должно быть получено:

```
MTH          NUM_HIRED
-----------  ----------
01-JAN-1981           0
```

```
01-FEB-1981          2
01-MAR-1981          0
01-APR-1981          1
01-MAY-1981          1
01-JUN-1981          1
01-JUL-1981          0
01-AUG-1981          0
01-SEP-1981          2
01-OCT-1981          0
01-NOV-1981          1
01-DEC-1981          2
```

Решение

Тонкость в том, что требуется возвратить строки для всех месяцев, даже для тех, в которые не было принято на работу ни одного служащего (т. е. количество принятых на работу равно нулю). Поэтому строки для тех месяцев, в которые служащие не принимались, необходимо сгенерировать самостоятельно и затем провести их внешнее объединение с таблицей EMP по столбцу HIREDATE (выбирая из фактических значений HIREDATE только месяц, чтобы их можно было сопоставлять со сгенерированными).

DB2

Чтобы получить все месяцы (первый день каждого месяца с 1 января 1980 по 1 декабря 1983 г.), используйте рекурсивный оператор WITH. Имея все месяцы заданного диапазона дат, проведите внешнее объединение с таблицей EMP и используйте агрегатную функцию COUNT, чтобы подсчитать, сколько сотрудников было принято на работу в каждом месяце:

```
 1    with x (start_date,end_date)
 2      as (
 3  select (min(hiredate) -
 4            dayofyear(min(hiredate)) day +1 day) start_date,
 5         (max(hiredate) -
 6            dayofyear(max(hiredate)) day +1 day) +1 year end_date
 7    from emp
 8   union all
 9  select start_date +1 month, end_date
10    from x
11   where (start_date +1 month) < end_date
12  )
13  select x.start_date mth, count(e.hiredate) num_hired
14    from x left join emp e
15      on (x.start_date = (e.hiredate-(day(hiredate)-1) day))
16   group by x.start_date
17   order by 1
```

Oracle

Чтобы получить все месяцы с 1980 по 1983 год, используйте оператор CONNECT BY. Затем проведите внешнее объединение с таблицей EMP и используйте агрегатную функцию COUNT, чтобы подсчитать, сколько сотрудников было принято на работу в каждом месяце. Для Oracle 8*i* Database и более ранних версий ANSI-синтаксис внешнего объединения недоступен, также как и применение оператора CONNECT BY для формирования строк. Простой выход в таком случае – традиционная сводная таблица (как в решении для MySQL). Ниже приведено решение для Oracle с использованием синтаксиса Oracle для внешнего объединения:

```
  with x
    as (
select add_months(start_date,level-1) start_date
  from (
select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
  from emp
       )
 connect by level <= months_between(end_date,start_date)
)
select x.start_date MTH, count(e.hiredate) num_hired
  from x, emp e
 where x.start_date = trunc(e.hiredate(+),'mm')
 group by x.start_date
 order by 1
```

а далее – второе решение для Oracle, на этот раз с использованием синтаксиса, соответствующего стандарту ANSI:

```
  with x
    as (
select add_months(start_date,level-1) start_date
  from (
select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
  from emp
       )
 connect by level <= months_between(end_date,start_date)
)
select x.start_date MTH, count(e.hiredate) num_hired
  from x left join emp e
    on (x.start_date = trunc(e.hiredate,'mm'))
 group by x.start_date
 order by 1
```

PostgreSQL

Чтобы сделать код более понятным, в данном решении для вычисления количества месяцев между первым днем первого месяца года

приема на работу первого служащего и первым днем последнего месяца года приема на работу последнего служащего, используется представление V. Значение, возвращенное представлением V, используйте как второй параметр функции GENERATE_SERIES, чтобы обеспечить получение верного количества месяцев (строк). Получив все месяцы заданного диапазона дат, проведите внешнее объединение с таблицей EMP и с помощью агрегатной функции COUNT подсчитайте, сколько сотрудников было принято на работу в каждом месяце:

```
create view v
as
select cast(
        extract(year from age(last_month,first_month))*12-1
          as integer) as mths
  from (
select cast(date_trunc('year',min(hiredate)) as date) as first_month,
       cast(cast(date_trunc('year',max(hiredate))
             as date) + interval '1 year'
             as date) as last_month
  from emp
       ) x

 1  select y.mth, count(e.hiredate) as num_hired
 2    from (
 3  select cast(e.start_date + (x.id * interval '1 month')
 4           as date) as mth
 5    from generate_series (0,(select mths from v)) x(id),
 6         ( select cast(
 7                   date_trunc('year',min(hiredate))
 8                     as date) as start_date
 9            from emp ) e
10         ) y left join emp e
11      on (y.mth = date_trunc('month',e.hiredate))
12   group by y.mth
13   order by 1
```

MySQL

Чтобы получить все месяцы между 1980 и 1983 годами, используйте сводную таблицу. Затем проведите внешнее объединение с таблицей EMP и с помощью агрегатной функции COUNT подсчитайте, сколько служащих было принято на работу в каждом месяце:

```
 1  select z.mth, count(e.hiredate) num_hired
 2    from (
 3  select date_add(min_hd,interval t500.id-1 month) mth
 4    from (
 5  select min_hd, date_add(max_hd,interval 11 month) max_hd
 6    from (
 7  select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
 8         adddate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
 9    from emp
```

```
10         ) x
11         ) y,
12         t500
13   where date_add(min_hd,interval t500.id-1 month) <= max_hd
14         ) z left join emp e
15      on (z.mth = adddate(
16                  date_add(
17                  last_day(e.hiredate),interval -1 month),1))
18   group by z.mth
19   order by 1
```

SQL Server

Чтобы получить все месяцы (первый день каждого месяца с 1 января 1980 по 1 декабря 1983), используйте рекурсивный оператор WITH. Имея все месяцы заданного диапазона дат, проведите внешнее объединение с таблицей EMP и используйте агрегатную функцию COUNT, чтобы подсчитать, сколько сотрудников было принято на работу в каждом месяце:

```
 1    with x (start_date,end_date)
 2      as (
 3  select (min(hiredate) -
 4            datepart(dy,min(hiredate))+1) start_date,
 5         dateadd(yy,1,
 6          (max(hiredate) -
 7           datepart(dy,max(hiredate))+1)) end_date
 8    from emp
 9   union all
10  select dateadd(mm,1,start_date), end_date
11    from x
12   where dateadd(mm,1,start_date) < end_date
13  )
14  select x.start_date mth, count(e.hiredate) num_hired
15    from x left join emp e
16      on (x.start_date =
17             dateadd(dd,-day(e.hiredate)+1,e.hiredate))
18   group by x.start_date
19   order by 1
```

Обсуждение

DB2

Первый шаг – получить все месяцы (на самом деле первый день каждого месяца) с 1980 по 1983 год. Чтобы найти граничные месяцы, применяем к значениям столбца HIREDATE функцию DAYOFYEAR совместно с функциями MIN и MAX:

```
select (min(hiredate) -
          dayofyear(min(hiredate)) day +1 day) start_date,
       (max(hiredate) -
```

```
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
  from emp

START_DATE  END_DATE
----------- -----------
01-JAN-1980 01-JAN-1984
```

Следующий шаг – начиная со значения START_DATE, последовательно многократно добавляем по одному месяцу. Таким образом получаем все месяцы, необходимые для формирования окончательного результирующего множества. Значение END_DATE (1 января 1984 года) на один день больше, чем должно быть. Ничего страшного. При рекурсивном добавлении месяцев к START_DATE можно будет остановиться на значении END_DATE, не включая его в результирующее множество. Полученные месяцы частично показаны ниже:

```
  with x (start_date,end_date)
    as (
select (min(hiredate) -
        dayofyear(min(hiredate)) day +1 day) start_date,
       (max(hiredate) -
        dayofyear(max(hiredate)) day +1 day) +1 year end_date
  from emp
 union all
select start_date +1 month, end_date
  from x
 where (start_date +1 month) < end_date
)
select *
  from x

START_DATE  END_DATE
----------- -----------
01-JAN-1980 01-JAN-1984
01-FEB-1980 01-JAN-1984
01-MAR-1980 01-JAN-1984
...
01-OCT-1983 01-JAN-1984
01-NOV-1983 01-JAN-1984
01-DEC-1983 01-JAN-1984
```

Теперь мы имеем все необходимые месяцы и можем провести внешнее объединение с EMP.HIREDATE. Поскольку в START_DATE хранятся первые дни месяцев, значения EMP.HIREDATE необходимо преобразовать в первые дни соответствующих месяцев с помощью функции TRUNC. Наконец, применяем к EMP.HIREDATE агрегатную функцию COUNT.

Oracle

Первый шаг – получить первый день каждого месяца с 1980 по 1983 год. Начинаем с определения граничных месяцев, применяя к значениям

HIREDATE функции TRUNC и ADD_MONTHS вместе с функциями MIN и MAX:

```
select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
  from emp
```

```
START_DATE  END_DATE
----------- -----------
01-JAN-1980 01-JAN-1984
```

Затем, начиная со START_DATE, многократно добавляем по одному месяцу, чтобы получить все месяцы, необходимые для формирования окончательного результирующего множества. Значение END_DATE соответствует следующему дню после окончания рассматриваемого периода. Ничего страшного, мы прекратим рекурсивное добавление месяцев по достижении END_DATE (не включая это значение в результат). Полученные месяцы частично показаны ниже:

```
with x as (
select add_months(start_date,level-1) start_date
  from (
select min(trunc(hiredate,'y')) start_date,
       add_months(max(trunc(hiredate,'y')),12) end_date
  from emp
       )
 connect by level <= months_between(end_date,start_date)
)
select *
  from x
```

```
START_DATE
-----------
01-JAN-1980
01-FEB-1980
01-MAR-1980
...
01-OCT-1983
01-NOV-1983
01-DEC-1983
```

На данный момент мы имеем все необходимые месяцы. Выполняем внешнее объединение с EMP.HIREDATE. Поскольку в START_DATE хранятся первые дни месяцев, значения EMP.HIREDATE необходимо преобразовать в первые дни соответствующих месяцев с помощью функции TRUNC. Наконец, применяем к EMP.HIREDATE агрегатную функцию COUNT.

PostgreSQL

В данном решении для получения всех необходимых месяцев используется функция GENERATE_SERIES. Если GENERATE_SERIES недоступна, можно обратиться к сводной таблице, как в решении для

MySQL. Первый шаг – понять представление V. Это представление находит, сколько месяцев требуется возвратить, определяя граничные даты диапазона. Для получения граничных дат во вложенном запросе X представления V используются наибольшее и наименьшее значения HIREDATE (выявленные функциями MIN и MAX), как показано ниже:

```
select cast(date_trunc('year',min(hiredate)) as date) as first_month,
       cast(cast(date_trunc('year',max(hiredate))
            as date) + interval '1 year'
            as date) as last_month
  from emp

FIRST_MONTH LAST_MONTH
----------- -----------
01-JAN-1980 01-JAN-1984
```

На самом деле значение LAST_MONTH соответствует следующему дню после окончания рассматриваемого периода. Ничего страшного, поскольку при подсчете месяцев между этими двумя датами из значения LAST_MONTH можно просто вычесть 1. Следующий шаг – с помощью функции AGE найти разницу между этими двумя датами в годах и умножить результат на 12 (и не забудьте вычесть 1!):

```
select cast(
        extract(year from age(last_month,first_month))*12-1
          as integer) as mths
  from (
select cast(date_trunc('year',min(hiredate)) as date) as first_month,
       cast(cast(date_trunc('year',max(hiredate))
            as date) + interval '1 year'
            as date) as last_month
  from emp
       ) x

MTHS
----
  47
```

Для получения необходимого количества месяцев используем значение, возвращенное представлением V, как второй параметр GENERATE_SERIES. Следующий шаг – найти начальную дату, с которой начинается рекурсивное добавление месяцев для формирования результирующего множества. Вложенный запрос Y находит начальную дату, применяя функцию DATE_TRUNC к MIN(HIREDATE), и использует значения, возвращенные GENERATE_SERIES, для добавления месяцев. Результаты частично показаны ниже:

```
select cast(e.start_date + (x.id * interval '1 month')
         as date) as mth
  from generate_series (0,(select mths from v)) x(id),
       ( select cast(
                  date_trunc('year',min(hiredate))
```

```
                    as date) as start_date
            from emp
          ) e

MTH
-----------
01-JAN-1980
01-FEB-1980
01-MAR-1980
...
01-OCT-1983
01-NOV-1983
01-DEC-1983
```

Теперь, имея все необходимые для формирования результирующего множества месяцы, проводим внешнее объединение с EMP.HIREDATE и используем агрегатную функцию COUNT, чтобы подсчитать, сколько сотрудников было принято на работу в каждом месяце.

MySQL

Сначала находим граничные даты, используя агрегатные функции MIN и MAX в сочетании с функциями DAYOFYEAR и ADDDATE. Вложенный запрос X возвращает следующее результирующее множество:

```
select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
       adddate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
  from emp

MIN_HD      MAX_HD
----------- -----------
01-JAN-1980 01-JAN-1983
```

Далее, используя значение MAX_HD, получаем последний месяц года:

```
select min_hd, date_add(max_hd,interval 11 month) max_hd
  from (
select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
       adddate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
  from emp
       ) x

MIN_HD      MAX_HD
----------- -----------
01-JAN-1980 01-DEC-1983
```

Теперь, имея граничные даты, добавляем месяцы, начиная с MIN_HD до MAX_HD включительно. Для получения необходимого количества строк используем сводную таблицу T500. Результаты частично показаны ниже:

```
select date_add(min_hd,interval t500.id-1 month) mth
  from (
select min_hd, date_add(max_hd,interval 11 month) max_hd
  from (
```

```
select adddate(min(hiredate),-dayofyear(min(hiredate))+1) min_hd,
       adddate(max(hiredate),-dayofyear(max(hiredate))+1) max_hd
  from emp
       ) x
       ) y,
       t500
 where date_add(min_hd,interval t500.id-1 month) <= max_hd

MTH
-----------
01-JAN-1980
01-FEB-1980
01-MAR-1980
...
01-OCT-1983
01-NOV-1983
01-DEC-1983
```

Имея все необходимые для формирования результирующего множества месяцы, проводим внешнее объединение с EMP.HIREDATE (не забудьте применить функцию TRUNC, чтобы значения EMP.HIREDATE представляли собой первые дни соответствующих месяцев) и применяем к EMP.HIREDATE агрегатную функцию COUNT, чтобы подсчитать, сколько сотрудников было принято на работу в каждом месяце.

SQL Server

Начинаем с получения всех месяцев (на самом деле первого дня каждого месяца) в период с 1980 по 1983 год. Затем находим граничные месяцы, применяя функцию DAYOFYEAR к наименьшему и наибольшему значениям HIREDATE (выявленным посредством функций MIN и MAX):

```
select (min(hiredate) -
         datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
        (max(hiredate) -
         datepart(dy,max(hiredate))+1)) end_date
  from emp

START_DATE  END_DATE
----------- -----------
01-JAN-1980 01-JAN-1984
```

Следующий шаг – многократное добавление месяцев, начиная с START_DATE, для получения всех необходимых для формирования результирующего множества месяцев. Значение END_DATE соответствует следующему дню после окончания рассматриваемого периода. Ничего страшного, мы прекратим рекурсивное добавление месяцев по достижении END_DATE (не включая это значение в результат). Полученные месяцы частично показаны ниже:

```
  with x (start_date,end_date)
    as (
select (min(hiredate) -
         datepart(dy,min(hiredate))+1) start_date,
       dateadd(yy,1,
        (max(hiredate) -
         datepart(dy,max(hiredate))+1)) end_date
  from emp
 union all
select dateadd(mm,1,start_date), end_date
  from x
 where dateadd(mm,1,start_date) < end_date
)
select *
  from x
```

```
START_DATE  END_DATE
----------- -----------
01-JAN-1980 01-JAN-1984
01-FEB-1980 01-JAN-1984
01-MAR-1980 01-JAN-1984
...
01-OCT-1983 01-JAN-1984
01-NOV-1983 01-JAN-1984
01-DEC-1983 01-JAN-1984
```

На данный момент мы имеем все необходимые месяцы. Выполняем внешнее объединение с EMP.HIREDATE. Поскольку в START_DATE хранятся первые дни месяцев, значения EMP.HIREDATE с помощью функции TRUNC также необходимо преобразовать в первые дни соответствующих месяцев. Последний шаг – применяем агрегатную функцию COUNT к EMP.HIREDATE.

Поиск по заданным единицам времени

Задача

Требуется выбрать даты, соответствующие заданному месяцу, или дню недели, или некоторой другой единице времени. Например, необходимо найти всех служащих, которые были приняты на работу в феврале или декабре, а также служащих, нанятых во вторник.

Решение

Для извлечения из даты месяца или названия дня недели используйте функции, предоставляемые СУБД. Этот рецепт может пригодиться во многих случаях, например, для выбора значений HIREDATE по определенному месяцу (или любой другой части даты). В примерах решения данной задачи осуществляется поиск по названию месяца и дня недели. Изучив функции форматирования дат, предоставляемые СУБД, можно без труда внести небольшие коррективы и использовать эти

решения для поиска дат по году, кварталу, году и кварталу, месяцу и году, и т. д.

DB2 и MySQL

Чтобы получить название месяца и дня недели приема служащего на работу, используйте функции MONTHNAME (название месяца) и DAYNAME (название дня) соответственно:

```
1 select ename
2   from emp
3  where monthname(hiredate) in ('February','December')
4     or dayname(hiredate) = 'Tuesday'
```

Oracle и PostgreSQL

Чтобы получить название месяца и дня недели приема служащего на работу, используйте функцию TO_CHAR. С помощью функции RTRIM удалите замыкающие пробелы:

```
1 select ename
2   from emp
3  where rtrim(to_char(hiredate,'month')) in ('february','december')
4     or rtrim(to_char(hiredate,'day')) = 'tuesday'
```

SQL Server

Чтобы получить название месяца и дня недели приема служащего на работу, используйте функцию DATENAME (имя даты):

```
1  select ename
2    from emp
3   where datename(m,hiredate) in ('February','December')
4      or datename(dw,hiredate) = 'Tuesday'
```

Обсуждение

Для решения поставленной задачи надо просто знать, какие функции использовать и как с ними работать. Для проверки значений, возвращаемых этими функциями, они помещаются в оператор SELECT. Ниже представлено результирующее множество для служащих 10-го отдела (полученное с помощью синтаксиса SQL Server):

```
select ename,datename(m,hiredate) mth,datename(dw,hiredate) dw
  from emp
 where deptno = 10

ENAME  MTH       DW
------ --------- ---------
CLARK  June      Tuesday
KING   November  Tuesday
MILLER January   Saturday
```

Когда известно, что возвращает(-ют) функция(-и), использовать ее (их) для поиска строк не составляет труда.

Сравнение строк по определенной части даты

Задача

Требуется найти служащих, которые были приняты на работу в один месяц и день недели. Например, если один служащий был нанят в понедельник 10 марта 1988 года, а другой служащий – в понедельник 2 марта 2001, они должны быть включены в результирующее множество, поскольку день недели и месяц их приема на работу совпадают. В таблице EMP только трое служащих отвечают этому требованию. Должно быть получено следующее результирующее множество:

```
MSG
-----------------------------------------------------
JAMES was hired on the same month and weekday as FORD
SCOTT was hired on the same month and weekday as JAMES
SCOTT was hired on the same month and weekday as FORD
```

Решение

Поскольку требуется сравнивать значение HIREDATE одного служащего со значениями HIREDATE других служащих, понадобится провести рефлексивное объединение таблицы EMP. Тем самым мы получим все возможные сочетания HIREDATE. После этого для каждой строки мы просто будем извлекать день недели и месяц из значений HIREDATE и сравнивать их у разных строк.

DB2

После рефлексивного объединения таблицы EMP с помощью функции DAYOFWEEK возвратите численное значение дня недели. Используйте функцию MONTHNAME, чтобы получить название месяца:

```
1 select a.ename ||
2        ' was hired on the same month and weekday as '||
3        b.ename msg
4   from emp a, emp b
5  where (dayofweek(a.hiredate),monthname(a.hiredate)) =
6        (dayofweek(b.hiredate),monthname(b.hiredate))
7    and a.empno < b.empno
8  order by a.ename
```

Oracle и PostgreSQL

После рефлексивного объединения таблицы EMP с помощью функции TO_CHAR извлеките из HIREDATE день недели и месяц для проведения сравнения:

```
1  select a.ename ||
2         ' was hired on the same month and weekday as '||
3         b.ename as msg
4    from emp a, emp b
5   where to_char(a.hiredate,'DMON') =
```

```
6         to_char(b.hiredate,'DMON')
7     and a.empno < b.empno
8   order by a.ename
```

MySQL

После рефлексивного объединения таблицы EMP с помощью функции DATE_FORMAT извлеките из HIREDATE день недели и месяц для проведения сравнения:

```
1  select concat(a.ename,
2         ' was hired on the same month and weekday as ',
3         b.ename) msg
4    from emp a, emp b
5   where date_format(a.hiredate,'%w%M') =
6         date_format(b.hiredate,'%w%M')
7     and a.empno < b.empno
8   order by a.ename
```

SQL Server

После рефлексивного объединения таблицы EMP с помощью функции DATENAME извлеките из HIREDATE день недели и месяц для проведения сравнения:

```
1 select a.ename +
2        ' was hired on the same month and weekday as '+
3        b.ename msg
4   from emp a, emp b
5  where datename(dw,a.hiredate) = datename(dw,b.hiredate)
6    and datename(m,a.hiredate)  = datename(m,b.hiredate)
7    and a.empno < b.empno
8  order by a.ename
```

Обсуждение

Решения отличаются лишь функциями работы с датами, используемыми для форматирования HIREDATE. В данном обсуждении я буду опираться на решение для Oracle/PostgreSQL (потому что оно самое лаконичное), но приводимые объяснения правомочны и для всех остальных решений.

Первый шаг – рефлексивное объединение EMP. Таким образом мы получаем все возможные комбинации значения HIREDATE заданного служащего и значений HIREDATE других служащих, что упрощает процесс сравнения. Рассмотрим приведенные ниже результаты запроса (фильтрация осуществляется по служащему SCOTT):

```
select a.ename as scott, a.hiredate as scott_hd,
       b.ename as other_emps, b.hiredate as other_hds
  from emp a, emp b
 where a.ename  = 'SCOTT'
   and a.empno != b.empno
```

```
SCOTT      SCOTT_HD    OTHER_EMPS OTHER_HDS
---------- ----------- ---------- -----------
SCOTT      09-DEC-1982 SMITH      17-DEC-1980
SCOTT      09-DEC-1982 ALLEN      20-FEB-1981
SCOTT      09-DEC-1982 WARD       22-FEB-1981
SCOTT      09-DEC-1982 JONES      02-APR-1981
SCOTT      09-DEC-1982 MARTIN     28-SEP-1981
SCOTT      09-DEC-1982 BLAKE      01-MAY-1981
SCOTT      09-DEC-1982 CLARK      09-JUN-1981
SCOTT      09-DEC-1982 KING       17-NOV-1981
SCOTT      09-DEC-1982 TURNER     08-SEP-1981
SCOTT      09-DEC-1982 ADAMS      12-JAN-1983
SCOTT      09-DEC-1982 JAMES      03-DEC-1981
SCOTT      09-DEC-1982 FORD       03-DEC-1981
SCOTT      09-DEC-1982 MILLER     23-JAN-1982
```

Благодаря рефлексивному объединению таблицы EMP мы можем сравнивать значение HIREDATE служащего SCOTT со значениями HIREDATE всех остальных служащих. Для EMPNO задан фильтр, который обеспечивает, что значение HIREDATE служащего SCOTT не будет включено в столбец OTHER_HDS. Следующий шаг – использовать предоставляемые СУБД функции форматирования дат для сравнения значений HIREDATE по дню недели и месяцу и выбора только тех значений, которые соответствуют заданному:

```
select a.ename as emp1, a.hiredate as emp1_hd,
       b.ename as emp2, b.hiredate as emp2_hd
  from emp a, emp b
 where to_char(a.hiredate,'DMON') =
       to_char(b.hiredate,'DMON')
   and a.empno != b.empno
 order by 1
```

```
EMP1       EMP1_HD     EMP2       EMP2_HD
---------- ----------- ---------- -----------
FORD       03-DEC-1981 SCOTT      09-DEC-1982
FORD       03-DEC-1981 JAMES      03-DEC-1981
JAMES      03-DEC-1981 SCOTT      09-DEC-1982
JAMES      03-DEC-1981 FORD       03-DEC-1981
SCOTT      09-DEC-1982 JAMES      03-DEC-1981
SCOTT      09-DEC-1982 FORD       03-DEC-1981
```

На данном этапе соответствующие значения HIREDATE выбраны правильно, но возвращено шесть строк вместо трех, приведенных в разделе «Задача» данного рецепта. Причина вывода лишних строк кроется в фильтре по EMPNO. Использование оператора «не равно» не обеспечивает отсеивание обратных равенств. Например, в первой строке сопоставляются FORD и SCOTT, а в последней строке – SCOTT и FORD. Шесть строк результирующего множества, с технической точки зрения, возвращены правильно, но они дублируют друг друга. Избавиться от дублирования позволит оператор «меньше чем» (значения HIRE-

DATE удаляются, чтобы максимально приблизить результаты промежуточных запросов к окончательному результирующему множеству):

```
select a.ename as emp1, b.ename as emp2
  from emp a, emp b
 where to_char(a.hiredate,'DMON') =
       to_char(b.hiredate,'DMON')
   and a.empno < b.empno
 order by 1
```

```
EMP1       EMP2
---------- ----------
JAMES      FORD
SCOTT      JAMES
SCOTT      FORD
```

Заключительный шаг – простая конкатенация результирующего множества для формирования сообщения.

Выявление наложений диапазонов дат

Задача

Требуется найти всех служащих, начинающих новый проект до завершения текущего. Рассмотрим таблицу EMP_PROJECT:

```
select *
  from emp_project
```

```
EMPNO ENAME      PROJ_ID PROJ_START  PROJ_END
----- ---------- ------- ----------- -----------
 7782 CLARK            1 16-JUN-2005 18-JUN-2005
 7782 CLARK            4 19-JUN-2005 24-JUN-2005
 7782 CLARK            7 22-JUN-2005 25-JUN-2005
 7782 CLARK           10 25-JUN-2005 28-JUN-2005
 7782 CLARK           13 28-JUN-2005 02-JUL-2005
 7839 KING             2 17-JUN-2005 21-JUN-2005
 7839 KING             8 23-JUN-2005 25-JUN-2005
 7839 KING            14 29-JUN-2005 30-JUN-2005
 7839 KING            11 26-JUN-2005 27-JUN-2005
 7839 KING             5 20-JUN-2005 24-JUN-2005
 7934 MILLER           3 18-JUN-2005 22-JUN-2005
 7934 MILLER          12 27-JUN-2005 28-JUN-2005
 7934 MILLER          15 30-JUN-2005 03-JUL-2005
 7934 MILLER           9 24-JUN-2005 27-JUN-2005
 7934 MILLER           6 21-JUN-2005 23-JUN-2005
```

Взглянув на результаты для служащего KING, мы видим, что он начал проект PROJ_ID 8 до завершения PROJ_ID 5, а также начал проект PROJ_ID 5 до завершения проекта PROJ_ID 2. Должно быть получено следующее результирующее множество:

```
EMPNO ENAME      MSG
----- ---------- --------------------------------
```

```
7782 CLARK      project 7 overlaps project 4
7782 CLARK      project 10 overlaps project 7
7782 CLARK      project 13 overlaps project 10
7839 KING       project 8 overlaps project 5
7839 KING       project 5 overlaps project 2
7934 MILLER     project 12 overlaps project 9
7934 MILLER     project 6 overlaps project 3
```

Решение

Ключ к решению – найти строки, в которых PROJ_START (дата начала нового проекта) совпадает или выпадает между значениями PROJ_START и PROJ_END (дата окончания проекта) другого проекта. Для начала мы должны получить возможность сравнивать даты начала и окончания проектов (одного служащего). Осуществляя рефлексивное объединение EMP_PROJECT по служащим, мы получаем все возможные сочетания двух проектов для каждого служащего. Чтобы выявить наложения, просто находим строки, где PROJ_START для одного PROJ_ID выпадает между PROJ_START и PROJ_END другого PROJ_ID одного и того же служащего.

DB2, PostgreSQL и Oracle

Выполните рефлексивное объединение EMP_PROJECT. Затем с помощью оператора конкатенации «||» скомпонуйте сообщение, описывающее, какие проекты перекрываются:

```
1  select a.empno,a.ename,
2         'project '||b.proj_id||
3          ' overlaps project '||a.proj_id as msg
4    from emp_project a,
5         emp_project b
6   where a.empno = b.empno
7     and b.proj_start >= a.proj_start
8     and b.proj_start <= a.proj_end
9     and a.proj_id != b.proj_id
```

MySQL

Выполните рефлексивное объединение EMP_PROJECT. Затем с помощью функции конкатенации CONCAT скомпонуйте сообщение, описывающее, какие проекты перекрываются:

```
1  select a.empno,a.ename,
2         concat('project ',b.proj_id,
3          ' overlaps project ',a.proj_id) as msg
4    from emp_project a,
5         emp_project b
6   where a.empno = b.empno
7     and b.proj_start >= a.proj_start
8     and b.proj_start <= a.proj_end
9     and a.proj_id != b.proj_id
```

SQL Server

Выполните рефлексивное объединение EMP_PROJECT. Затем с помощью оператора конкатенации «+» скомпонуйте сообщение, описывающее, какие проекты перекрываются:

```
1  select a.empno,a.ename,
2         'project '+b.proj_id+
3          ' overlaps project '+a.proj_id as msg
4    from emp_project a,
5         emp_project b
6   where a.empno = b.empno
7     and b.proj_start >= a.proj_start
8     and b.proj_start <= a.proj_end
9     and a.proj_id != b.proj_id
```

Обсуждение

Все решения отличаются лишь в части, касающейся конкатенации строк, поэтому обсудим всех их на примере синтаксиса DB2. Первый шаг – рефлексивное объединение EMP_PROJECT, которое обеспечивает возможность сравнения дат PROJ_START разных проектов. Результат рефлексивного объединения для служащего KING показан ниже. Из него становится ясно, как каждый проект может «видеть» другие проекты:

```
select a.ename,
       a.proj_id as a_id,
       a.proj_start as a_start,
       a.proj_end as a_end,
       b.proj_id as b_id,
       b.proj_start as b_start
  from emp_project a,
       emp_project b
 where a.ename    = 'KING'
   and a.empno    = b.empno
   and a.proj_id != b.proj_id
  order by 2

ENAME   A_ID A_START     A_END        B_ID B_START
------ ----- ----------- ----------- ----- -----------
KING       2 17-JUN-2005 21-JUN-2005     8 23-JUN-2005
KING       2 17-JUN-2005 21-JUN-2005    14 29-JUN-2005
KING       2 17-JUN-2005 21-JUN-2005    11 26-JUN-2005
KING       2 17-JUN-2005 21-JUN-2005     5 20-JUN-2005
KING       5 20-JUN-2005 24-JUN-2005     2 17-JUN-2005
KING       5 20-JUN-2005 24-JUN-2005     8 23-JUN-2005
KING       5 20-JUN-2005 24-JUN-2005    11 26-JUN-2005
KING       5 20-JUN-2005 24-JUN-2005    14 29-JUN-2005
KING       8 23-JUN-2005 25-JUN-2005     2 17-JUN-2005
KING       8 23-JUN-2005 25-JUN-2005    14 29-JUN-2005
KING       8 23-JUN-2005 25-JUN-2005     5 20-JUN-2005
```

```
KING       8 23-JUN-2005 25-JUN-2005    11 26-JUN-2005
KING      11 26-JUN-2005 27-JUN-2005     2 17-JUN-2005
KING      11 26-JUN-2005 27-JUN-2005     8 23-JUN-2005
KING      11 26-JUN-2005 27-JUN-2005    14 29-JUN-2005
KING      11 26-JUN-2005 27-JUN-2005     5 20-JUN-2005
KING      14 29-JUN-2005 30-JUN-2005     2 17-JUN-2005
KING      14 29-JUN-2005 30-JUN-2005     8 23-JUN-2005
KING      14 29-JUN-2005 30-JUN-2005     5 20-JUN-2005
KING      14 29-JUN-2005 30-JUN-2005    11 26-JUN-2005
```

Как видно из приведенного выше результирующего множества, рефлексивное объединение упрощает задачу по выявлению перекрытия дат; теперь остается просто выбрать все строки, в которых B_START выпадает между A_START и A_ END. Предикат WHERE в строках 7 и 8 решения:

```
and b.proj_start >= a.proj_start
and b.proj_start <= a.proj_end
```

выполняет именно это. Когда получены необходимые строки, создание сообщений – это всего лишь дело конкатенации возвращенных значений.

Пользователям Oracle доступна оконная функция LEAD OVER, которая позволяет избежать рефлексивного объединения, если один служащий ведет фиксированное число проектов. Использование этой функции может пригодиться, если рефлексивное объединение слишком «дорого» для конкретного рассматриваемого случая (если рефлексивное объединение требует больше ресурсов, чем сортировка, необходимая для LEAD OVER). Например, рассмотрим альтернативное решение для служащего KING с использованием функции LEAD OVER:

```
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
       when lead(proj_start,1)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       when lead(proj_start,2)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       when lead(proj_start,3)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       when lead(proj_start,4)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       end is_overlap
  from emp_project
```

```
 where ename = 'KING'

EMPNO ENAME  PROJ_ID PROJ_START  PROJ_END    IS_OVERLAP
----- ------ ------- ----------- ----------- ----------
 7839 KING         2 17-JUN-2005 21-JUN-2005          5
 7839 KING         5 20-JUN-2005 24-JUN-2005          8
 7839 KING         8 23-JUN-2005 25-JUN-2005
 7839 KING        11 26-JUN-2005 27-JUN-2005
 7839 KING        14 29-JUN-2005 30-JUN-2005
```

Поскольку для служащего KING количество проектов фиксировано и равно пяти, для сравнения дат всех проектов без рефлексивного объединения можно использовать LEAD OVER. С этого момента формирование результирующего множества не составляет труда. Просто выбираем строки, в которых значение столбца IS_OVERLAP (перекрывается) не NULL:

```
select empno,ename,
       'project '||is_overlap||
        ' overlaps project '||proj_id msg
  from (
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
       when lead(proj_start,1)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       when lead(proj_start,2)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       when lead(proj_start,3)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       when lead(proj_start,4)over(order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(order by proj_start)
       end is_overlap
  from emp_project
 where ename = 'KING'
       )
 where is_overlap is not null

EMPNO ENAME  MSG
----- ------ --------------------------------
 7839 KING   project 5 overlaps project 2
 7839 KING   project 8 overlaps project 5
```

Чтобы в решении участвовали все служащие (не только KING), в функции LEAD OVER выполняем сегментирование по ENAME:

```
select empno,ename,
       'project '||is_overlap||
        ' overlaps project '||proj_id msg
  from (
select empno,
       ename,
       proj_id,
       proj_start,
       proj_end,
       case
       when lead(proj_start,1)over(partition by ename
                                   order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(partition by ename
                              order by proj_start)
       when lead(proj_start,2)over(partition by ename
                                   order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(partition by ename
                              order by proj_start)
       when lead(proj_start,3)over(partition by ename
                                   order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(partition by ename
                              order by proj_start)
       when lead(proj_start,4)over(partition by ename
                                   order by proj_start)
            between proj_start and proj_end
       then lead(proj_id)over(partition by ename
                              order by proj_start)
       end is_overlap
  from emp_project
       )
 where is_overlap is not null

EMPNO ENAME  MSG
----- ------ --------------------------------
 7782 CLARK  project 7 overlaps project 4
 7782 CLARK  project 10 overlaps project 7
 7782 CLARK  project 13 overlaps project 10
 7839 KING   project 5 overlaps project 2
 7839 KING   project 8 overlaps project 5
 7934 MILLER project 6 overlaps project 3
 7934 MILLER project 12 overlaps project 9
```

10

Работа с диапазонами данных

Данная глава рассказывает об «обыденных» запросах с участием диапазонов. В повседневной жизни диапазоны встречаются всюду. Например, работа над проектом занимает определенный диапазон времени. В SQL часто возникает необходимость поиска диапазонов, или формирования диапазонов, или любой другой обработки диапазонов данных. Представленные здесь запросы немного сложнее обсуждавшихся в предыдущих главах, но они так же часто используются. Овладев ими в полной мере, вы почувствуете настоящую мощь SQL.

Поиск диапазона последовательных значений

Задача

Требуется найти строки, представляющие ряд последовательных проектов. Рассмотрим следующее представление V, содержащее данные о проекте и даты его начала и окончания:

```
select *
  from V

PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005
      3 03-JAN-2005 04-JAN-2005
      4 04-JAN-2005 05-JAN-2005
      5 06-JAN-2005 07-JAN-2005
      6 16-JAN-2005 17-JAN-2005
      7 17-JAN-2005 18-JAN-2005
      8 18-JAN-2005 19-JAN-2005
      9 19-JAN-2005 20-JAN-2005
     10 21-JAN-2005 22-JAN-2005
```

```
11 26-JAN-2005 27-JAN-2005
12 27-JAN-2005 28-JAN-2005
13 28-JAN-2005 29-JAN-2005
14 29-JAN-2005 30-JAN-2005
```

Для всех строк, кроме первой, значение PROJ_START (начало проекта) должно быть равным значению PROJ_END (конец проекта) предыдущей строки («предыдущая» строка для текущей строки определяется как PROJ_ID – 1). Рассмотрев первые пять строк представления V, можно заметить, что проекты с PROJ_ID от 1 до 3 являются частью одной «группы», так как для каждого из них PROJ_END равняется PROJ_START следующей строки. Поскольку требуется найти диапазон дат последовательно выполняющихся проектов, должны быть выбраны все строки, в которых значение PROJ_END текущей строки равно значению PROJ_START следующей строки. Если бы представление V состояло только из первых пяти строк, были бы выбраны лишь первые три из них. Необходимо получить следующее результирующее множество (используя все 14 строк представления V):

```
PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005
      3 03-JAN-2005 04-JAN-2005
      6 16-JAN-2005 17-JAN-2005
      7 17-JAN-2005 18-JAN-2005
      8 18-JAN-2005 19-JAN-2005
     11 26-JAN-2005 27-JAN-2005
     12 27-JAN-2005 28-JAN-2005
     13 28-JAN-2005 29-JAN-2005
```

Строки с PROJ_ID 4, 5, 9, 10 и 14 не вошли в это результирующее множество, потому что их PROJ_END не совпадает с PROJ_START последующих строк.

Решение

DB2, MySQL, PostgreSQL и SQL Server

Для выявления строк с последовательными значениями используйте рефлексивное объединение:

```
1 select v1.proj_id,
2        v1.proj_start,
3        v1.proj_end
4   from V v1, V v2
5  where v1.proj_end = v2.proj_start
```

Oracle

Предыдущее решение подходит и для Oracle. Привожу альтернативное решение, в котором для проверки значения BEGIN_DATE «сле-

дующей» строки используется оконная функция LEAD OVER, что устраняет необходимость в рефлексивном объединении:

```
1 select proj_id,proj_start,proj_end
2   from (
3 select proj_id,proj_start,proj_end,
4        lead(proj_start)over(order by proj_id) next_proj_start
5   from V
6        )
7  where next_proj_start = proj_end
```

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

Благодаря объединению представления с самим собой (рефлексивному объединению) каждую строку можно сравнить со всеми остальными строками. Рассмотрим часть результирующего множества для строк со значениями ID 1 и 4:

```
select v1.proj_id as v1_id,
       v1.proj_end as v1_end,
       v2.proj_start as v2_begin,
       v2.proj_id as v2_id
  from v v1, v v2
 where v1.proj_id in ( 1,4 )
```

```
V1_ID V1_END      V2_BEGIN         V2_ID
----- ----------- ----------- ----------
    1 02-JAN-2005 01-JAN-2005          1
    1 02-JAN-2005 02-JAN-2005          2
    1 02-JAN-2005 03-JAN-2005          3
    1 02-JAN-2005 04-JAN-2005          4
    1 02-JAN-2005 06-JAN-2005          5
    1 02-JAN-2005 16-JAN-2005          6
    1 02-JAN-2005 17-JAN-2005          7
    1 02-JAN-2005 18-JAN-2005          8
    1 02-JAN-2005 19-JAN-2005          9
    1 02-JAN-2005 21-JAN-2005         10
    1 02-JAN-2005 26-JAN-2005         11
    1 02-JAN-2005 27-JAN-2005         12
    1 02-JAN-2005 28-JAN-2005         13
    1 02-JAN-2005 29-JAN-2005         14
    4 05-JAN-2005 01-JAN-2005          1
    4 05-JAN-2005 02-JAN-2005          2
    4 05-JAN-2005 03-JAN-2005          3
    4 05-JAN-2005 04-JAN-2005          4
    4 05-JAN-2005 06-JAN-2005          5
    4 05-JAN-2005 16-JAN-2005          6
    4 05-JAN-2005 17-JAN-2005          7
    4 05-JAN-2005 18-JAN-2005          8
    4 05-JAN-2005 19-JAN-2005          9
    4 05-JAN-2005 21-JAN-2005         10
```

```
         4 05-JAN-2005 26-JAN-2005          11
         4 05-JAN-2005 27-JAN-2005          12
         4 05-JAN-2005 28-JAN-2005          13
         4 05-JAN-2005 29-JAN-2005          14
```

Из этого результирующего множества видно, почему строка с PROJ_ID 1 вошла в окончательное результирующее множество, а строка с PROJ_ID 4 нет: для V1_ID 4 нет такой строки, в которой значение V2_BEGIN соответствовало бы значению V1_END.

Но в зависимости от того, как рассматриваются данные, PROJ_ID 4 может быть включен в результирующее множество. Рассмотрим следующее результирующее множество:

```
select *
  from V
 where proj_id <= 5

PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005
      3 03-JAN-2005 04-JAN-2005
      4 04-JAN-2005 05-JAN-2005
      5 06-JAN-2005 07-JAN-2005
```

Если «последовательность» образуют проекты, которые начинаются в день окончания другого проекта, PROJ_ID 4 должен быть включен в результирующее множество. Изначально PROJ_ID 4 был исключен из-за сравнения со следующей строкой (сравнивалось значение PROJ_END текущей строки со значением PROJ_START из следующей). Но при сравнении с предыдущей строкой (сравнивается значение PROJ_START текущей строки со значением PROJ_END из предыдущей) PROJ_ID 4 будет включен в результат.

Изменить решение так, чтобы PROJ_ID 4 был включен в результирующее множество, просто: требуется ввести дополнительный предикат, обеспечивающий проверку не только значения PROJ_END, но и PROJ_START. Внесенные изменения, которые демонстрирует следующий запрос, обеспечивают создание результирующего множества, включающего PROJ_ID 4 (ключевое слово DISTINCT необходимо, поскольку некоторые строки удовлетворяют обоим условиям):

```
select distinct
       v1.proj_id,
       v1.proj_start,
       v1.proj_end
  from V v1, V v2
 where v1.proj_end   = v2.proj_start
    or v1.proj_start = v2.proj_end

PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
```

```
         1 01-JAN-2005 02-JAN-2005
         2 02-JAN-2005 03-JAN-2005
         3 03-JAN-2005 04-JAN-2005
         4 04-JAN-2005 05-JAN-2005
```

Oracle

Несомненно, рефлексивное объединение выполняет поставленную задачу, но идеальным решением проблем такого типа является использование оконной функции LEAD OVER.[1] Она позволяет сравнивать строки без рефлексивного объединения (хотя для этого ей требуется упорядоченное результирующее множество). Рассмотрим результаты вложенного запроса (строки 3–5) для ID 1 и 4:

```
select *
  from (
select proj_id,proj_start,proj_end,
       lead(proj_start)over(order by proj_id) next_proj_start
  from v
       )
 where proj_id in ( 1,4 )
```

```
PROJ_ID PROJ_START  PROJ_END    NEXT_PROJ_START
------- ----------- ----------- ---------------
      1 01-JAN-2005 02-JAN-2005 02-JAN-2005
      4 04-JAN-2005 05-JAN-2005 06-JAN-2005
```

После анализа приведенного выше фрагмента кода и результирующего множества становится понятным, почему PROJ_ID 4 не включен в окончательное результирующее множество решения: потому что его значение PROJ_END (05-JAN-2005) не совпадает с датой начала «следующего» проекта (06-JAN-2005).

Функция LEAD OVER исключительно полезна при решении подобных задач, особенно при рассмотрении частичных результатов. При работе с оконными функциями необходимо помнить, что они выполняются после операторов FROM и WHERE. Таким образом, функция LEAD OVER в приведенном выше запросе должна располагаться во вложенном запросе. В противном случае LEAD OVER применяется к результирующему множеству уже после того, как предикат отсеет все строки, кроме строк с PROJ_ID 1 и 4.

Теперь в зависимости от того, как рассматриваются данные, может потребоваться включить проект с PROJ_ID 4 в окончательное результирующее множество. Рассмотрим первые пять строк представления V:

```
select *
  from V
```

[1] Не совсем корректно говорить о функции LEAD OVER, т. к. функцией является только LEAD, а OVER – это связанное с ней аналитическое выражение. – *Примеч. науч. ред.*

```
 where proj_id <= 5

PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005
      3 03-JAN-2005 04-JAN-2005
      4 04-JAN-2005 05-JAN-2005
      5 06-JAN-2005 07-JAN-2005
```

Если предъявляемые требования таковы, что проект с PROJ_ID 4 считается последовательным (потому что его PROJ_START соответствует PROJ_END проекта PROJ_ID 3), и условиям не удовлетворяет только проект PROJ_ID 5, предлагаемое для данного рецепта решение неправильное (!) или, по крайней мере, неполное:

```
select proj_id,proj_start,proj_end
  from (
select proj_id,proj_start,proj_end,
       lead(proj_start)over(order by proj_id) next_start
  from V
 where proj_id <= 5
       )
 where proj_end = next_start

PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005
      3 03-JAN-2005 04-JAN-2005
```

Если проект с PROJ_ID 4 должен быть включен, просто добавьте в запрос LAG OVER и используйте дополнительный фильтр в предикате WHERE:

```
select proj_id,proj_start,proj_end
  from (
select proj_id,proj_start,proj_end,
       lead(proj_start)over(order by proj_id) next_start,
       lag(proj_end)over(order by proj_id) last_end
  from V
 where proj_id <= 5
       )
 where proj_end   = next_start
    or proj_start = last_end

PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005
      3 03-JAN-2005 04-JAN-2005
      4 04-JAN-2005 05-JAN-2005
```

Теперь проект с PROJ_ID 4 включен в окончательное результирующее множество, и только проект с PROJ_ID 5 не вошел в него. Применяя эти рецепты в своем коде, внимательно изучайте предъявляемые требования.

Вычисление разности между значениями строк одной группы или сегмента

Задача

Требуется выбрать для каждого служащего DEPTNO, ENAME и SAL, а также найти разность между значениями SAL для служащих одного отдела (т. е. для служащих с одинаковыми значениями DEPTNO). Разность должна вычисляться между каждым служащим и служащим, который был принят на работу сразу после него (необходимо выяснить, есть ли зависимость между стажем работы и заработной платой для каждого отдела). Для служащих, которые были приняты на работу позже всех в своих отделах, возвращается значение «N/A». Должно быть получено следующее результирующее множество:

```
DEPTNO ENAME             SAL HIREDATE    DIFF
------ ---------- ---------- ----------- ----------
    10 CLARK            2450 09-JUN-1981      -2550
    10 KING             5000 17-NOV-1981       3700
    10 MILLER           1300 23-JAN-1982        N/A
    20 SMITH             800 17-DEC-1980      -2175
    20 JONES            2975 02-APR-1981        -25
    20 FORD             3000 03-DEC-1981          0
    20 SCOTT            3000 09-DEC-1982       1900
    20 ADAMS            1100 12-JAN-1983        N/A
    30 ALLEN            1600 20-FEB-1981        350
    30 WARD             1250 22-FEB-1981      -1600
    30 BLAKE            2850 01-MAY-1981       1350
    30 TURNER           1500 08-SEP-1981        250
    30 MARTIN           1250 28-SEP-1981        300
    30 JAMES             950 03-DEC-1981        N/A
```

Решение

Данная задача – еще один пример, когда пригодятся оконные функции[1] Oracle LEAD OVER и LAG OVER. Они обеспечивают возможность доступа к последующей и предыдущей строкам без дополнительных объединений. Для остальных СУБД можно использовать скалярные подзапросы, хотя они усложняют решение. Эта конкретная задача теряет всю свою прелесть, когда приходится решать ее с помощью скалярных подзапросов и рефлексивных объединений.

[1] В документации по Oracle эти функции также называются аналитическими. – *Примеч. науч. ред.*

DB2, MySQL, PostgreSQL и SQL Server

Для получения значения HIREDATE для служащего, принятого на работу следующим после рассматриваемого служащего, используйте скалярный подзапрос. Затем с помощью другого скалярного подзапроса найдите его заработную плату:

```
 1  select deptno,ename,hiredate,sal,
 2         coalesce(cast(sal-next_sal as char(10)),'N/A') as diff
 3    from (
 4  select e.deptno,
 5         e.ename,
 6         e.hiredate,
 7         e.sal,
 8         (select min(sal) from emp d
 9           where d.deptno=e.deptno
10             and d.hiredate =
11                  (select min(hiredate) from emp d
12                    where e.deptno=d.deptno
13                      and d.hiredate > e.hiredate)) as next_sal
14    from emp e
15         ) x
```

Oracle

Чтобы получить заработную плату служащего, «следующего» по отношению к текущей строке, используйте оконную функцию LEAD OVER:

```
1 select deptno,ename,sal,hiredate,
2        lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
3   from (
4 select deptno,ename,sal,hiredate,
5        lead(sal)over(partition by deptno
6                          order by hiredate) next_sal
7   from emp
8        )
```

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

Первый шаг – с помощью скалярного подзапроса найти значение HIREDATE служащего, принятого на работу в тот же отдел сразу после рассматриваемого служащего. В скалярном подзапросе используется функция MIN(HIREDATE), которая обеспечивает возвращение только одного значения, даже если в один день было нанято несколько человек:

```
select e.deptno,
       e.ename,
       e.hiredate,
       e.sal,
       (select min(hiredate) from emp d
          where e.deptno=d.deptno
```

```
           and d.hiredate > e.hiredate) as next_hire
  from emp e
 order by 1
```

```
DEPTNO ENAME      HIREDATE           SAL NEXT_HIRE
------ ---------- ----------- ---------- -----------
    10 CLARK      09-JUN-1981       2450 17-NOV-1981
    10 KING       17-NOV-1981       5000 23-JAN-1982
    10 MILLER     23-JAN-1982       1300
    20 SMITH      17-DEC-1980        800 02-APR-1981
    20 ADAMS      12-JAN-1983       1100
    20 FORD       03-DEC-1981       3000 09-DEC-1982
    20 SCOTT      09-DEC-1982       3000 12-JAN-1983
    20 JONES      02-APR-1981       2975 03-DEC-1981
    30 ALLEN      20-FEB-1981       1600 22-FEB-1981
    30 BLAKE      01-MAY-1981       2850 08-SEP-1981
    30 MARTIN     28-SEP-1981       1250 03-DEC-1981
    30 JAMES      03-DEC-1981        950
    30 TURNER     08-SEP-1981       1500 28-SEP-1981
    30 WARD       22-FEB-1981       1250 01-MAY-1981
```

Следующий шаг – с помощью другого скалярного подзапроса найти заработную плату служащего, принятого на работу в день, соответствующий дате NEXT_HIRE. Опять же в решении используется функция MIN, что гарантирует возвращение одного значения:

```
select e.deptno,
       e.ename,
       e.hiredate,
       e.sal,
       (select min(sal) from emp d
         where d.deptno=e.deptno
           and d.hiredate =
                (select min(hiredate) from emp d
                  where e.deptno=d.deptno
                    and d.hiredate > e.hiredate)) as next_sal
  from emp e
 order by 1
```

```
DEPTNO ENAME      HIREDATE           SAL   NEXT_SAL
------ ---------- ----------- ---------- ----------
    10 CLARK      09-JUN-1981       2450       5000
    10 KING       17-NOV-1981       5000       1300
    10 MILLER     23-JAN-1982       1300
    20 SMITH      17-DEC-1980        800       2975
    20 ADAMS      12-JAN-1983       1100
    20 FORD       03-DEC-1981       3000       3000
    20 SCOTT      09-DEC-1982       3000       1100
    20 JONES      02-APR-1981       2975       3000
    30 ALLEN      20-FEB-1981       1600       1250
    30 BLAKE      01-MAY-1981       2850       1500
    30 MARTIN     28-SEP-1981       1250        950
```

```
    30 JAMES      03-DEC-1981        950
    30 TURNER     08-SEP-1981       1500       1250
    30 WARD       22-FEB-1981       1250       2850
```

Заключительный шаг – найти разность между значениями SAL и NEXT_SAL, а в случае необходимости возвратить «N/A». Для этого используем функцию COALESCE. Поскольку результатом вычитания является число или NULL, в функции COALESCE необходимо выполнить приведение результата вычитания к строковому типу:

```
select deptno,ename,hiredate,sal,
       coalesce(cast(sal-next_sal as char(10)),'N/A') as diff
  from (
select e.deptno,
       e.ename,
       e.hiredate,
       e.sal,
       (select min(sal) from emp d
         where d.deptno=e.deptno
           and d.hiredate =
                (select min(hiredate) from emp d
                  where e.deptno=d.deptno
                    and d.hiredate > e.hiredate)) as next_sal
  from emp e
       ) x
 order by 1
```

```
DEPTNO ENAME      HIREDATE           SAL DIFF
------ ---------- ----------- ---------- ---------
    10 CLARK      09-JUN-1981       2450 -2550
    10 KING       17-NOV-1981       5000 3700
    10 MILLER     23-JAN-1982       1300 N/A
    20 SMITH      17-DEC-1980        800 -2175
    20 ADAMS      12-JAN-1983       1100 N/A
    20 FORD       03-DEC-1981       3000 0
    20 SCOTT      09-DEC-1982       3000 1900
    20 JONES      02-APR-1981       2975 -25
    30 ALLEN      20-FEB-1981       1600 350
    30 BLAKE      01-MAY-1981       2850 1350
    30 MARTIN     28-SEP-1981       1250 300
    30 JAMES      03-DEC-1981        950 N/A
    30 TURNER     08-SEP-1981       1500 250
    30 WARD       22-FEB-1981       1250 -1600
```

Применение в данном решении MIN(SAL) является примером того, как можно непреднамеренно ввести в запрос бизнес-логику при решении чисто технической задачи. Если данной дате соответствуют несколько заработных плат, какую следует выбрать? Наименьшую? Наибольшую? Среднюю? В своем примере я решил брать наименьшую. В реальности я бы предоставил возможность бизнес-клиенту, запросившему отчет, принимать это решение.

Oracle

Первый шаг – с помощью оконной функции LEAD OVER найти для каждого служащего заработную плату «следующего» служащего в его отделе. В столбце NEXT_SAL для служащего любого отдела, принятого на работу последним, располагается значение NULL:

```
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp

DEPTNO ENAME             SAL HIREDATE      NEXT_SAL
------ ---------- ---------- ----------- ----------
    10 CLARK            2450 09-JUN-1981       5000
    10 KING             5000 17-NOV-1981       1300
    10 MILLER           1300 23-JAN-1982
    20 SMITH             800 17-DEC-1980       2975
    20 JONES            2975 02-APR-1981       3000
    20 FORD             3000 03-DEC-1981       3000
    20 SCOTT            3000 09-DEC-1982       1100
    20 ADAMS            1100 12-JAN-1983
    30 ALLEN            1600 20-FEB-1981       1250
    30 WARD             1250 22-FEB-1981       2850
    30 BLAKE            2850 01-MAY-1981       1500
    30 TURNER           1500 08-SEP-1981       1250
    30 MARTIN           1250 28-SEP-1981        950
    30 JAMES             950 03-DEC-1981
```

Следующий шаг – для каждого служащего найти разность между его заработной платой и заработной платой служащего, принятого на работу в этот отдел сразу после него:

```
select deptno,ename,sal,hiredate, sal-next_sal diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp
       )

DEPTNO ENAME             SAL HIREDATE          DIFF
------ ---------- ---------- ----------- ----------
    10 CLARK            2450 09-JUN-1981      -2550
    10 KING             5000 17-NOV-1981       3700
    10 MILLER           1300 23-JAN-1982
    20 SMITH             800 17-DEC-1980      -2175
    20 JONES            2975 02-APR-1981        -25
    20 FORD             3000 03-DEC-1981          0
    20 SCOTT            3000 09-DEC-1982       1900
    20 ADAMS            1100 12-JAN-1983
    30 ALLEN            1600 20-FEB-1981        350
    30 WARD             1250 22-FEB-1981      -1600
    30 BLAKE            2850 01-MAY-1981       1350
    30 TURNER           1500 08-SEP-1981        250
```

```
    30 MARTIN          1250 28-SEP-1981        300
    30 JAMES            950 03-DEC-1981
```

Следующий шаг – использовать функцию NVL, чтобы возвратить значение «N/A», если DIFF является NULL. Чтобы возвратить «N/A», необходимо привести значение DIFF к строковому типу, в противном случае функция NVL даст сбой:

```
select deptno,ename,sal,hiredate,
       nvl(to_char(sal-next_sal),'N/A') diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp
       )

DEPTNO ENAME             SAL HIREDATE    DIFF
------ ---------- ---------- ----------- ---------------
    10 CLARK            2450 09-JUN-1981 -2550
    10 KING             5000 17-NOV-1981 3700
    10 MILLER           1300 23-JAN-1982 N/A
    20 SMITH             800 17-DEC-1980 -2175
    20 JONES            2975 02-APR-1981 -25
    20 FORD             3000 03-DEC-1981 0
    20 SCOTT            3000 09-DEC-1982 1900
    20 ADAMS            1100 12-JAN-1983 N/A
    30 ALLEN            1600 20-FEB-1981 350
    30 WARD             1250 22-FEB-1981 -1600
    30 BLAKE            2850 01-MAY-1981 1350
    30 TURNER           1500 08-SEP-1981 250
    30 MARTIN           1250 28-SEP-1981 300
    30 JAMES             950 03-DEC-1981 N/A
```

Последний шаг – с помощью функции LPAD форматировать значения для столбца DIFF. Это необходимо, потому что по умолчанию числа выравниваются по правому краю, а строки – по левому. Используя LPAD, мы выравниваем все значения столбца по правому краю:

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno order by hiredate) next_sal
  from emp
       )

DEPTNO ENAME             SAL HIREDATE    DIFF
------ ---------- ---------- ----------- ----------
    10 CLARK            2450 09-JUN-1981      -2550
    10 KING             5000 17-NOV-1981       3700
    10 MILLER           1300 23-JAN-1982        N/A
    20 SMITH             800 17-DEC-1980      -2175
    20 JONES            2975 02-APR-1981        -25
```

```
    20 FORD            3000 03-DEC-1981          0
    20 SCOTT           3000 09-DEC-1982       1900
    20 ADAMS           1100 12-JAN-1983        N/A
    30 ALLEN           1600 20-FEB-1981        350
    30 WARD            1250 22-FEB-1981      -1600
    30 BLAKE           2850 01-MAY-1981       1350
    30 TURNER          1500 08-SEP-1981        250
    30 MARTIN          1250 28-SEP-1981        300
    30 JAMES            950 03-DEC-1981        N/A
```

Хотя в подавляющем большинстве рецептов, предлагаемых в данной книге, сценарии «что если» не рассматриваются (из соображений удобства для чтения и с целью сохранения психического здоровья автора), сценарий с участием дублирующихся значений при таком использовании функции Oracle LEAD OVER требует особого внимания. В простых данных таблицы EMP, используемых нами для примера, ни для одного служащего значения столбца HIREDATE не дублируются, тем не менее такая возможность весьма вероятна. В обычной ситуации я бы не обсуждал обработку дубликатов (поскольку таких значений нет в таблице EMP), но не всем (особенно тем, кто не работает с Oracle) может быть понятна методика с использованием функции LEAD. Рассмотрим следующий запрос, возвращающий разность заработных плат для служащих 10-го отдела (разности вычисляются в порядке приема служащих на работу):

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                         order by hiredate) next_sal
  from emp
 where deptno=10 and empno > 10
       )

DEPTNO ENAME    SAL HIREDATE    DIFF
------ ------ ----- ----------- ----------
    10 CLARK   2450 09-JUN-1981      -2550
    10 KING    5000 17-NOV-1981       3700
    10 MILLER  1300 23-JAN-1982        N/A
```

Здесь все правильно, если рассматривать данные таблицы EMP. Но если бы в таблице присутствовали строки с дублирующимися значениями, решение дало бы сбой. Посмотрим на пример ниже, в котором в таблицу вносятся данные о еще четырех служащих, которые были приняты на работу в один день со служащим KING:

```
insert into emp (empno,ename,deptno,sal,hiredate)
values (1,'ant',10,1000,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (2,'joe',10,1500,to_date('17-NOV-1981'))
```

```
insert into emp (empno,ename,deptno,sal,hiredate)
values (3,'jim',10,1600,to_date('17-NOV-1981'))

insert into emp (empno,ename,deptno,sal,hiredate)
values (4,'jon',10,1700,to_date('17-NOV-1981'))

select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
  from (
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                         order by hiredate) next_sal
  from emp
 where deptno=10
       )

DEPTNO ENAME    SAL HIREDATE    DIFF
------ ------ ----- ----------- ----------
    10 CLARK   2450 09-JUN-1981       1450
    10 ant     1000 17-NOV-1981       -500
    10 joe     1500 17-NOV-1981      -3500
    10 KING    5000 17-NOV-1981       3400
    10 jim     1600 17-NOV-1981       -100
    10 jon     1700 17-NOV-1981        400
    10 MILLER  1300 23-JAN-1982        N/A
```

Как мы видим, за исключением служащего JON, для всех служащих, принятых в один день (17 ноября), разность заработных плат вычисляется относительно другого служащего, который был нанят в тот же день! Это неправильно. Заработная плата всех служащих, которые устроились 17 ноября, должна сравниваться с заработной платой служащего MILLER. Рассмотрим, например, служащего ANT. Значение столбца DIFF для ANT равно -500, потому что его значение SAL сравнивается с SAL служащего JOE и оно на 500 меньше, чем заработная плата JOE, отсюда получаем -500. Значение DIFF служащего ANT должно составлять -300, поскольку ANT получает на 300 денежных единиц меньше, чем MILLER, служащий, принятый на работу следующим, согласно HIREDATE. Такие ошибки возникают из-за стандартного поведения Oracle-функции LEAD OVER. По умолчанию LEAD OVER проверяет только следующую строку. И для служащего ANT, исходя из HIREDATE, следующим значением SAL является SAL служащего JOE, потому что LEAD OVER просто берет следующую строку и не пропускает дубликаты. К счастью, Oracle предусматривает такую ситуацию: в LEAD OVER может быть передан дополнительный параметр, определяющий, как далеко вперед должна «заглянуть» функция. В приведенном выше примере вопрос стоит лишь в определении того, на сколько строк отстоит запись каждого служащего, принятого на работу 17 ноября, от записи, соответствующей 23 января (дате приема на работу служащего MILLER). Решение ниже показывает, как это реализовывается:

```
select deptno,ename,sal,hiredate,
       lpad(nvl(to_char(sal-next_sal),'N/A'),10) diff
```

```
  from (
select deptno,ename,sal,hiredate,
       lead(sal,cnt-rn+1)over(partition by deptno
                         order by hiredate) next_sal
  from (
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate order by sal) rn
  from emp
 where deptno=10
       )
       )

DEPTNO ENAME    SAL HIREDATE    DIFF
------ ------ ----- ----------- ----------
    10 CLARK   2450 09-JUN-1981       1450
    10 ant     1000 17-NOV-1981       -300
    10 joe     1500 17-NOV-1981        200
    10 jim     1600 17-NOV-1981        300
    10 jon     1700 17-NOV-1981        400
    10 KING    5000 17-NOV-1981       3700
    10 MILLER  1300 23-JAN-1982        N/A
```

Теперь все правильно. Как видите, заработные платы всех служащих, принятых 17 ноября, сравниваются с заработной платой MILLER. В результате для служащего ANT значение DIFF равно -300, что и требовалось получить. Может быть, не все поняли, но переданное в LEAD OVER выражение CNT – RN + 1 – это просто количество строк между записями каждого служащего, поступившего на работу 17 ноября, и служащего MILLER. Рассмотрим вложенный запрос, приведенный ниже, который демонстрирует значения CNT и RN:

```
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate order by sal) rn
  from emp
 where deptno=10

DEPTNO ENAME    SAL HIREDATE           CNT         RN
------ ------ ----- ----------- ---------- ----------
    10 CLARK   2450 09-JUN-1981          1          1
    10 ant     1000 17-NOV-1981          5          1
    10 joe     1500 17-NOV-1981          5          2
    10 jim     1600 17-NOV-1981          5          3
    10 jon     1700 17-NOV-1981          5          4
    10 KING    5000 17-NOV-1981          5          5
    10 MILLER  1300 23-JAN-1982          1          1
```

Значение CNT для каждого служащего показывает, сколько в таблице служащих с таким значением HIREDATE. Значение RN представляет ранг служащего. Ранги подразделяются по DEPTNO и HIREDATE, поэтому только служащие с дублирующимся HIREDATE могут иметь ранг

больше единицы. Ранжирование выполняется согласно размеру заработной платы (это произвольный выбор; использовать значения столбца SAL удобно, но точно так же можно было бы выбрать и столбец EMPNO). Теперь, когда мы знаем общее количество дубликатов и упорядочили их, расстояние до записи MILLER – это просто общее число дубликатов минус ранг текущей записи плюс один (CNT – RN + 1). Результаты вычисления расстояния и выполнения LEAD OVER показаны ниже:

```
select deptno,ename,sal,hiredate,
       lead(sal)over(partition by deptno
                         order by hiredate) incorrect,
       cnt-rn+1 distance,
       lead(sal,cnt-rn+1)over(partition by deptno
                         order by hiredate) correct
  from (
select deptno,ename,sal,hiredate,
       count(*)over(partition by deptno,hiredate) cnt,
       row_number()over(partition by deptno,hiredate
                           order by sal) rn
  from emp
 where deptno=10
       )

DEPTNO ENAME    SAL HIREDATE     INCORRECT   DISTANCE    CORRECT
------ ------ ----- ----------- ---------- ---------- ----------
    10 CLARK   2450 09-JUN-1981       1000          1       1000
    10 ant     1000 17-NOV-1981       1500          5       1300
    10 joe     1500 17-NOV-1981       1600          4       1300
    10 jim     1600 17-NOV-1981       1700          3       1300
    10 jon     1700 17-NOV-1981       5000          2       1300
    10 KING    5000 17-NOV-1981       1300          1       1300
    10 MILLER  1300 23-JAN-1982                     1
```

Сейчас можно ясно видеть эффект от передачи в LEAD OVER второго параметра. Столбец INCORRECT (неправильно) представляет значения, возвращаемые LEAD OVER при реализации поведения по умолчанию (сравнения со значением следующей строки). Столбец CORRECT (правильно) представляет значения, возвращаемые LEAD OVER при использовании расстояния для каждого служащего с дублирующимся HIREDATE до строки служащего MILLER. На данном этапе осталось лишь найти разность между значениями CORRECT и SAL для каждой строки, что уже было показано.

Определение начала и конца диапазона последовательных значений

Задача

Данный рецепт является расширением предыдущего и использует то же представление V. Теперь, когда диапазоны последовательных зна-

чений определены, требуется найти всего лишь начало и конец этих диапазонов. В отличие от рецепта выше, если строки не входят в множество последовательных значений, они все равно должны быть возвращены. Почему? Потому что именно такие строки и представляют начало и конец диапазонов. Используя данные представления V:

```
select *
  from V

PROJ_ID PROJ_START  PROJ_END
------- ----------- -----------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005
      3 03-JAN-2005 04-JAN-2005
      4 04-JAN-2005 05-JAN-2005
      5 06-JAN-2005 07-JAN-2005
      6 16-JAN-2005 17-JAN-2005
      7 17-JAN-2005 18-JAN-2005
      8 18-JAN-2005 19-JAN-2005
      9 19-JAN-2005 20-JAN-2005
     10 21-JAN-2005 22-JAN-2005
     11 26-JAN-2005 27-JAN-2005
     12 27-JAN-2005 28-JAN-2005
     13 28-JAN-2005 29-JAN-2005
     14 29-JAN-2005 30-JAN-2005
```

требуется получить такое результирующее множество:

```
PROJ_GRP PROJ_START  PROJ_END
-------- ----------- -----------
       1 01-JAN-2005 05-JAN-2005
       2 06-JAN-2005 07-JAN-2005
       3 16-JAN-2005 20-JAN-2005
       4 21-JAN-2005 22-JAN-2005
       5 26-JAN-2005 30-JAN-2005
```

Решение

Эта задача несколько сложнее предыдущей. Во-первых, необходимо найти диапазоны. Диапазон строк определяется значениями PROJ_START и PROJ_END. Чтобы строка считалась «последовательной» или частью группы, ее значение PROJ_START должно быть равным значению PROJ_END предыдущей строки. В случае, когда значение PROJ_START строки не равно значению PROJ_END предыдущей строки и ее значение PROJ_END не равно значению PROJ_START следующей строки, строка является экземпляром группы, состоящей из одной строки. После определения диапазонов необходимо сгруппировать строки в этих диапазонах (в группы) и возвратить только их начальные и конечные точки.

Рассмотрим первую строку требуемого результирующего множества. Значение PROJ_START – это PROJ_START строки PROJ_ID 1 пред-

ставления V, и PROJ_END – это PROJ_END строки PROJ_ID 4 представления V. Хотя за значением строки PROJ_ID 4 не следует никакого последовательного значения, это последнее значение диапазона последовательных значений, таким образом, оно включено в первую группу.

DB2, MySQL, PostgreSQL и SQL Server

В решении для этих платформ используется представление V2, чтобы сделать код более понятным. Представление V2 определено следущим образом:

```
create view v2
as
select a.*,
       case
         when (
            select b.proj_id
              from V b
             where a.proj_start = b.proj_end
              )
              is not null then 0 else 1
       end as flag
  from V a
```

Результирующее множество представления V2:

```
select *
  from V2

PROJ_ID PROJ_START  PROJ_END          FLAG
------- ----------- ----------- ----------
      1 01-JAN-2005 02-JAN-2005          1
      2 02-JAN-2005 03-JAN-2005          0
      3 03-JAN-2005 04-JAN-2005          0
      4 04-JAN-2005 05-JAN-2005          0
      5 06-JAN-2005 07-JAN-2005          1
      6 16-JAN-2005 17-JAN-2005          1
      7 17-JAN-2005 18-JAN-2005          0
      8 18-JAN-2005 19-JAN-2005          0
      9 19-JAN-2005 20-JAN-2005          0
     10 21-JAN-2005 22-JAN-2005          1
     11 26-JAN-2005 27-JAN-2005          1
     12 27-JAN-2005 28-JAN-2005          0
     13 28-JAN-2005 29-JAN-2005          0
     14 29-JAN-2005 30-JAN-2005          0
```

При использовании V2 порядок решения такой: сначала находим строки, составляющие множество последовательных значений, затем группируем эти строки, после чего с помощью функций MIN и MAX находим начальную и конечную точки:

```
1  select proj_grp,
2         min(proj_start) as proj_start,
```

```
 3         max(proj_end) as proj_end
 4    from (
 5  select a.proj_id,a.proj_start,a.proj_end,
 6         (select sum(b.flag)
 7            from V2 b
 8           where b.proj_id <= a.proj_id) as proj_grp
 9    from V2 a
10         ) x
11   group by proj_grp
```

Oracle

Хотя решения для других СУБД подойдут и для Oracle, использование оконной функции Oracle LAG OVER устраняет необходимость вводить дополнительные представления. С помощью LAG OVER сравниваем значения PROJ_END предыдущей строки со значением PROJ_START текущей строки и распределяем строки по группам. Сформировав группы, находим их граничные значения с помощью функций MIN и MAX:

```
 1  select proj_grp, min(proj_start), max(proj_end)
 2    from (
 3  select proj_id,proj_start,proj_end,
 4         sum(flag)over(order by proj_id) proj_grp
 5    from (
 6  select proj_id,proj_start,proj_end,
 7         case when
 8              lag(proj_end)over(order by proj_id) = proj_start
 9              then 0 else 1
10         end flag
11    from V
12         )
13         )
14   group by proj_grp
```

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

Применение представления V2 упрощает решение данной задачи. Представление V2 с помощью скалярного подзапроса в выражении CASE определяет принадлежность той или иной строки к множеству последовательных значений. Выражение CASE под псевдонимом FLAG возвращает 0, если текущая строка является частью последовательного множества, и 1 в противном случае (членство в последовательном множестве определяется существованием записи, значение PROJ_END которой соответствует значению PROJ_START текущей строки). Следующий шаг – проверить вложенный запрос X (строки 5–9). Вложенный запрос X возвращает все строки представления V2 и текущую сумму по FLAG; эта текущая сумма и является в данном случае признаком группы, что можно увидеть ниже:

```
select a.proj_id,a.proj_start,a.proj_end,
       (select sum(b.flag)
          from v2 b
         where b.proj_id <= a.proj_id) as proj_grp
  from v2 a

PROJ_ID PROJ_START  PROJ_END      PROJ_GRP
------- ----------- ----------- ----------
      1 01-JAN-2005 02-JAN-2005          1
      2 02-JAN-2005 03-JAN-2005          1
      3 03-JAN-2005 04-JAN-2005          1
      4 04-JAN-2005 05-JAN-2005          1
      5 06-JAN-2005 07-JAN-2005          2
      6 16-JAN-2005 17-JAN-2005          3
      7 17-JAN-2005 18-JAN-2005          3
      8 18-JAN-2005 19-JAN-2005          3
      9 19-JAN-2005 20-JAN-2005          3
     10 21-JAN-2005 22-JAN-2005          4
     11 26-JAN-2005 27-JAN-2005          5
     12 27-JAN-2005 28-JAN-2005          5
     13 28-JAN-2005 29-JAN-2005          5
     14 29-JAN-2005 30-JAN-2005          5
```

Сгруппировав строки по диапазонам, находим начальную и конечную точку каждой группы, просто применяя агрегатные функции MIN и MAX к значениям столбцов PROJ_START и PROJ_END соответственно, и группируем результаты по значениям текущей суммы.

Oracle

Оконная функция LAG OVER исключительно полезна в данной ситуации. Она позволяет проверять значение PROJ_END предыдущей строки без рефлексивного объединения, без скалярного подзапроса и без представления. Результаты выполнения функции LAG OVER без выражения CASE следующие:

```
select proj_id,proj_start,proj_end,
       lag(proj_end)over(order by proj_id) prior_proj_end
  from V

PROJ_ID PROJ_START  PROJ_END    PRIOR_PROJ_END
------- ----------- ----------- --------------
      1 01-JAN-2005 02-JAN-2005
      2 02-JAN-2005 03-JAN-2005 02-JAN-2005
      3 03-JAN-2005 04-JAN-2005 03-JAN-2005
      4 04-JAN-2005 05-JAN-2005 04-JAN-2005
      5 06-JAN-2005 07-JAN-2005 05-JAN-2005
      6 16-JAN-2005 17-JAN-2005 07-JAN-2005
      7 17-JAN-2005 18-JAN-2005 17-JAN-2005
      8 18-JAN-2005 19-JAN-2005 18-JAN-2005
      9 19-JAN-2005 20-JAN-2005 19-JAN-2005
     10 21-JAN-2005 22-JAN-2005 20-JAN-2005
     11 26-JAN-2005 27-JAN-2005 22-JAN-2005
```

```
        12 27-JAN-2005 28-JAN-2005 27-JAN-2005
        13 28-JAN-2005 29-JAN-2005 28-JAN-2005
        14 29-JAN-2005 30-JAN-2005 29-JAN-2005
```

Выражение CASE в решении просто сравнивает значение, возвращаемое LAG OVER, со значением PROJ_START текущей строки. Если они совпадают, возвращается 0, в противном случае возвращается 1. Следующий шаг – вычислить текущую сумму 0 и 1, возвращенных выражением CASE, чтобы распределить все строки по группам. Результаты вычисления текущей суммы приведены ниже:

```
select proj_id,proj_start,proj_end,
       sum(flag)over(order by proj_id) proj_grp
  from (
select proj_id,proj_start,proj_end,
       case when
            lag(proj_end)over(order by proj_id) = proj_start
            then 0 else 1
       end flag
  from V
       )

PROJ_ID PROJ_START  PROJ_END      PROJ_GRP
------- ----------- ----------- ----------
      1 01-JAN-2005 02-JAN-2005          1
      2 02-JAN-2005 03-JAN-2005          1
      3 03-JAN-2005 04-JAN-2005          1
      4 04-JAN-2005 05-JAN-2005          1
      5 06-JAN-2005 07-JAN-2005          2
      6 16-JAN-2005 17-JAN-2005          3
      7 17-JAN-2005 18-JAN-2005          3
      8 18-JAN-2005 19-JAN-2005          3
      9 19-JAN-2005 20-JAN-2005          3
     10 21-JAN-2005 22-JAN-2005          4
     11 26-JAN-2005 27-JAN-2005          5
     12 27-JAN-2005 28-JAN-2005          5
     13 28-JAN-2005 29-JAN-2005          5
     14 29-JAN-2005 30-JAN-2005          5
```

Распределив все строки по группам, просто применяем агрегатные функции MIN и MAX к значениям PROJ_START и PROJ_END соответственно и группируем полученные значения по столбцу текущей суммы PROJ_GRP.

Вставка пропущенных значений диапазона

Задача

Требуется найти, по сколько служащих устраивалось на работу каждый год в 1980-х, причем в этой декаде есть несколько лет, когда прием служащих на работу не проводился. Должно быть получено следующее результирующее множество:

```
YR          CNT
---- ----------
1980          1
1981         10
1982          2
1983          1
1984          0
1985          0
1986          0
1987          0
1988          0
1989          0
```

Решение

В этом решении тонкость в том, что требуется возвратить нули для тех лет, когда найм служащих не производился. Если в данном году ни один служащий не был принят на работу, значит, в таблице EMP нет строки, соответствующей этому году. Если года нет в таблице, как возвратить для него итоговое количество служащих, пусть даже это будет нуль? Здесь понадобится внешнее объединение. Необходимо предоставить результирующее множество, содержащее все запрашиваемые годы, и затем подсчитать по таблице EMP, сколько служащих было принято на работу в каждом из них.

DB2

Используя таблицу EMP как сводную таблицу (поскольку в ней 14 строк) и встроенную функцию YEAR, сгенерируйте по строке для каждого года десятилетия 1980-х. Выполните внешнее объединение с таблицей EMP и подсчитайте, по сколько служащих устраивалось на работу каждый год:

```
 1 select x.yr, coalesce(y.cnt,0) cnt
 2   from (
 3 select year(min(hiredate)over()) -
 4        mod(year(min(hiredate)over()),10) +
 5        row_number()over()-1 yr
 6   from emp fetch first 10 rows only
 7        ) x
 8   left join
 9        (
10 select year(hiredate) yr1, count(*) cnt
11   from emp
12  group by year(hiredate)
13        ) y
14     on ( x.yr = y.yr1 )
```

Oracle

Используя таблицу EMP как сводную таблицу (поскольку в ней 14 строк) и встроенные функции TO_NUMBER и TO_CHAR, сгенери-

руйте по строке для каждого года десятилетия 1980-х. Выполните внешнее объединение с таблицей EMP и подсчитайте, по сколько служащих устраивалось на работу каждый год:

```
 1  select x.yr, coalesce(cnt,0) cnt
 2    from (
 3  select extract(year from min(hiredate)over()) -
 4         mod(extract(year from min(hiredate)over()),10) +
 5         rownum-1 yr
 6    from emp
 7   where rownum <= 10
 8         ) x,
 9         (
10  select to_number(to_char(hiredate,'YYYY')) yr, count(*) cnt
11    from emp
12   group by to_number(to_char(hiredate,'YYYY'))
13         ) y
14   where x.yr = y.yr(+)
```

При работе с Oracle 9*i* Database или более поздними версиями можно реализовать решение, используя новый оператор JOIN:

```
 1  select x.yr, coalesce(cnt,0) cnt
 2    from (
 3  select extract(year from min(hiredate)over()) -
 4         mod(extract(year from min(hiredate)over()),10) +
 5         rownum-1 yr
 6    from emp
 7   where rownum <= 10
 8         ) x
 9    left join
10         (
11  select to_number(to_char(hiredate,'YYYY')) yr, count(*) cnt
12    from emp
13   group by to_number(to_char(hiredate,'YYYY'))
14         ) y
15      on ( x.yr = y.yr )
```

PostgreSQL и MySQL

Используя таблицу T10 как сводную таблицу (поскольку в ней 10 строк) и встроенную функцию EXTRACT, сгенерируйте по строке для каждого года десятилетия 1980-х. Выполните внешнее объединение с таблицей EMP и подсчитайте, по сколько служащих устраивалось на работу каждый год:

```
1  select y.yr, coalesce(x.cnt,0) as cnt
2    from (
3  select min_year-mod(cast(min_year as int),10)+rn as yr
4    from (
5  select (select min(extract(year from hiredate))
6            from emp) as min_year,
7         id-1 as rn
```

```
 8    from t10
 9         ) a
10         ) y
11    left join
12         (
13  select extract(year from hiredate) as yr, count(*) as cnt
14    from emp
15   group by extract(year from hiredate)
16         ) x
17      on ( y.yr = x.yr )
```

SQL Server

Используя таблицу EMP как сводную таблицу (поскольку в ней 14 строк) и встроенную функцию YEAR, сгенерируйте по строке для каждого года десятилетия 1980-х. Выполните внешнее объединение с таблицей EMP и подсчитайте, по сколько служащих устраивалось на работу каждый год:

```
 1  select x.yr, coalesce(y.cnt,0) cnt
 2    from (
 3  select top (10)
 4         (year(min(hiredate)over()) -
 5          year(min(hiredate)over())%10)+
 6          row_number()over(order by hiredate)-1 yr
 7    from emp
 8         ) x
 9    left join
10         (
11  select year(hiredate) yr, count(*) cnt
12    from emp
13   group by year(hiredate)
14         ) y
15      on ( x.yr = y.yr )
```

Обсуждение

Несмотря на различия в синтаксисе, во всех решениях подход один и тот же. Вложенный запрос X находит, в каком году десятилетия 80-х был принят первый служащий (минимальное значение HIREDATE). Следующий шаг – к разности между годом первого приема на работу и остатком от деления этого года на 10 добавляется RN – 1. Чтобы разобраться, просто выполните вложенный запрос X и получите каждое из участвующих в этой операции значений по отдельности. Ниже представлено результирующее множество вложенного запроса X, использующего оконную функцию MIN OVER (DB2, Oracle, SQL Server) и скалярный подзапрос (MySQL, PostgreSQL):

```
select year(min(hiredate)over()) -
       mod(year(min(hiredate)over()),10) +
       row_number()over()-1 yr,
```

```
       year(min(hiredate)over()) min_year,
       mod(year(min(hiredate)over()),10) mod_yr,
       row_number()over()-1 rn
  from emp fetch first 10 rows only
```

```
  YR   MIN_YEAR     MOD_YR         RN
---- ---------- ---------- ----------
1980       1980          0          0
1981       1980          0          1
1982       1980          0          2
1983       1980          0          3
1984       1980          0          4
1985       1980          0          5
1986       1980          0          6
1987       1980          0          7
1988       1980          0          8
1989       1980          0          9
```

```
select min_year-mod(min_year,10)+rn as yr,
       min_year,
       mod(min_year,10) as mod_yr
       rn
  from (
select (select min(extract(year from hiredate))
          from emp) as min_year,
       id-1 as rn
  from t10
       ) x
```

```
  YR   MIN_YEAR     MOD_YR         RN
---- ---------- ---------- ----------
1980       1980          0          0
1981       1980          0          1
1982       1980          0          2
1983       1980          0          3
1984       1980          0          4
1985       1980          0          5
1986       1980          0          6
1987       1980          0          7
1988       1980          0          8
1989       1980          0          9
```

Вложенный запрос Y возвращает годы, соответствующие значениям HIREDATE, и количество служащих, принятых на работу в течение каждого из них:

```
select year(hiredate) yr, count(*) cnt
  from emp
 group by year(hiredate)
```

```
   YR        CNT
----- ----------
 1980          1
 1981         10
```

```
1982            2
1983            1
```

Для получения окончательного решения выполняем внешнее объединение вложенного запроса Y с вложенным запросом X, чтобы в результирующем множестве получить все годы десятилетия, даже те, в течение которых ни один служащий не был принят на работу.

Формирование последовательности числовых значений

Задача

Хотелось бы иметь в своем распоряжении «генератор строк». Генераторы строк пригодятся для запросов, в которых требуется выполнить разворачивание. Например, стоит задача получить результирующее множество, подобное приведенному ниже, с любым заданным количеством строк:

```
ID
---
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
...
```

Если СУБД предоставляет встроенные функции для динамического формирования строк, в предварительном создании сводной таблицы с фиксированным числом строк нет необходимости. В этом и состоит преимущество динамического генератора строк. В противном случае, если требуется получить определенное количество строк, приходится использовать традиционную сводную таблицу (и этого не всегда достаточно).

Решение

В данном решении показано, как получить 10 строк, в которых располагаются числа по возрастанию, начиная с 1. Это решение можно без труда адаптировать для получения любого количества строк.

Возможность получать ряд последовательных возрастающих значений, начиная с 1, открывает путь ко многим другим решениям. Например, можно генерировать числа и, добавляя их к датам, получать последовательности дней. Кроме того, можно использовать такие ряды чисел для синтаксического разбора строк.

DB2 и SQL Server

Для формирования последовательности строк с возрастающими значениями используйте рекурсивный оператор WITH. В качестве отправной точки для формирования строк используйте таблицу T1, содержащую одну строку; все остальное сделает оператор WITH:

```
 1  with x (id)
 2  as (
 3  select 1
 4    from t1
 5   union all
 6  select id+1
 7    from x
 8   where id+1 <= 10
 9  )
10  select * from x
```

Далее приведено второе, альтернативное решение, подходящее только для DB2. Его преимущество состоит в том, что в нем не требуется таблица T1:[1]

```
1  with x (id)
2  as (
3  values (1)
4   union all
5  select id+1
6    from x
7   where id+1 <= 10
8  )
9  select * from x
```

Oracle

Используйте рекурсивный оператор CONNECT BY (Oracle 9*i* Database или более поздние версии). В Oracle 9*i* Database необходимо поместить решение с применением CONNECT BY в конструкцию WITH:

```
1  with x
2  as (
3  select level id
4    from dual
5    connect by level <= 10
6  )
7  select * from x
```

В Oracle Database 10*g* или более поздних версиях можно формировать строки с помощью оператора MODEL:

```
1  select array id
2    from dual
```

[1] Для SQL Server также можно обойтись без таблицы T1, для этого достаточно опустить строку 4 в общем решении. – *Примеч. науч. ред.*

```
3   model
4     dimension by (0 idx)
5     measures(1 array)
6     rules iterate (10) (
7       array[iteration_number] = iteration_number+1
8     )
```

PostgreSQL

Используйте очень удобную функцию GENERATE_SERIES, которая разработана именно для формирования строк:

```
1 select id
2   from generate_series (1,10) x(id)
```

Обсуждение

DB2 и SQL Server

Рекурсивный оператор WITH увеличивает ID (начиная с 1) до тех пор, пока не будет выполнен предикат WHERE. Для начала необходимо получить одну строку со значением 1. Сделать это можно с помощью таблицы, состоящей из одной строки, или в случае с DB2, используя оператор VALUES, который создаст результирующее множество, содержащее одну строку.[1]

Oracle

В этом решении CONNECT BY помещен в конструкцию WITH. Строки будут формироваться до тех пор, пока не выполнится предикат WHERE. Oracle автоматически увеличивает значение псевдостолбца LEVEL.

В решении с использованием оператора MODEL применяется явная команда ITERATE, которая обеспечивает формирование множества строк. Без конструкции ITERATE возвращена будет лишь одна строка, поскольку в DUAL всего одна строка. Например:

```
select array id
  from dual
 model
   dimension by (0 idx)
   measures(1 array)
   rules ()

ID
--
 1
```

Оператор MODEL не только обеспечивает возможность доступа к строкам, как к элементам массива, он позволяет «создавать» или возвра-

[1] В SQL Server, чтобы сформировать одну строку, можно указать SELECT без FROM части. – *Примеч. науч. ред.*

щать строки, даже если их нет в таблице, из которой производится выбор строк. В данном решении IDX – это индекс массива (местоположение конкретного значения в массиве), и ARRAY (под псевдонимом ID) – это «массив» строк. Первая строка по умолчанию получает значение 1, к ней можно обратиться через ARRAY[0]. Oracle предоставляет функцию ITERATION_NUMBER, с помощью которой можно отслеживать количество выполненных итераций. В решении выполняется 10 итераций, таким образом, ITERATION_NUMBER возвращает значения от 0 до 9. Добавляя 1 к каждому из этих значений, получаем результаты от 1 до 10.

Следующий запрос поможет наглядно представить, что происходит с конструкцией MODEL:

```
select 'array['||idx||'] = '||array as output
  from dual
 model
   dimension by (0 idx)
   measures(1 array)
   rules iterate (10) (
     array[iteration_number] = iteration_number+1
   )

OUTPUT
------------------
array[0] = 1
array[1] = 2
array[2] = 3
array[3] = 4
array[4] = 5
array[5] = 6
array[6] = 7
array[7] = 8
array[8] = 9
array[9] = 10
```

PostgreSQL

Всю работу выполняет функция GENERATE_SERIES. Она принимает три параметра, все они являются числовыми значениями. Первый параметр – начальное значение, второй – конечное значение, и третий необязательный параметр – «шаг» (какое приращение получает каждое значение). Если третий параметр опущен, по умолчанию приращение равно 1.

Функция GENERATE_SERIES обладает достаточной гибкостью, что позволяет не прописывать в коде передаваемые в нее параметры. Например, стоит задача возвратить пять строк, начиная со значения 10, заканчивая значением 30, с шагом 5. Чтобы получить следующее результирующее множество:

```
 ID
---
 10
 15
 20
 25
 30
```

можно применить творческий подход и сделать что-то такое:

```
select id
  from generate_series(
         (select min(deptno) from emp),
         (select max(deptno) from emp),
         5
       ) x(id)
```

Обратите внимание, что здесь фактические значения, передаваемые в GENERATE_SERIES, на момент написания запроса не известны. Они формируются подзапросами при выполнении основного запроса.

11

Расширенный поиск

По большому счету, вся эта книга посвящена поиску. Мы уже ознакомились со всеми необходимыми видами запросов, использующими объединения, предикаты WHERE и методики группировки для выявления и возвращения результатов. Однако некоторые типы операций поиска стоят особняком, поскольку представляют другой взгляд на поиск. Например, требуется выводить результирующее множество постранично. Одна половина этой задачи состоит в определении (поиске) всего множества записей, которые должны быть представлены. Другая – в осуществлении повторного поиска следующей страницы результатов для вывода на экран пользователя, когда он просматривает записи. На первый взгляд кажется, что разбиение результатов на страницы не является задачей поиска, но эту задачу *можно* рассматривать с этой точки зрения, и она может быть решена таким образом. О таких решениях и пойдет речь в данной главе.

Разбиение результирующего множества на страницы

Задача

Требуется разбить на страницы или обеспечить возможность «прокрутки» результирующего множества. Например, необходимо возвратить первые пять заработных плат из таблицы EMP, затем следующие пять и т. д. Наша цель – предоставить пользователю возможность просматривать по пять строк, переходя к следующей порции результатов по нажатию кнопки «Далее».

Решение

Поскольку в SQL нет понятия первой, последней или следующей строки, необходимо каким-то образом упорядочить рассматриваемые стро-

ки. Только упорядочив их, мы сможем возвращать строки строго соответственно диапазонам значений.

DB2, Oracle и SQL Server

С помощью ранжирующей функции ROW_NUMBER OVER упорядочьте строки. В предикате WHERE определите окно записей, которое требуется возвращать. Например, чтобы возвратить строки от 1 до 5:

```
select sal
  from (
select row_number() over (order by sal) as rn,
       sal
  from emp
       ) x
 where rn between 1 and 5
```

```
SAL
----
 800
 950
1100
1250
1250
```

Затем, чтобы возвратить строки 6–10:

```
select sal
  from (
select row_number() over (order by sal) as rn,
       sal
  from emp
       ) x
 where rn between 6 and 10
```

```
  SAL
-----
 1300
 1500
 1600
 2450
 2850
```

Можно выбрать любой диапазон строк, просто меняя предикат WHERE внешнего запроса.

MySQL и PostgreSQL

Данные продукты поддерживают операторы LIMIT и OFFSET, что сильно упрощает задачу по просмотру результирующего множества. С помощью LIMIT задайте, какое количество строк должно быть возвращено, и с помощью OFFSET определите, сколько строк необходимо пропустить. Например, чтобы возвратить первые пять строк в порядке возрастания заработных плат:

```
select sal
  from emp
 order by sal limit 5 offset 0
```

```
   SAL
------
   800
   950
  1100
  1250
  1250
```

Чтобы возвратить следующие пять строк:

```
select sal
  from emp
 order by sal limit 5 offset 5
```

```
  SAL
-----
 1300
 1500
 1600
 2450
 2850
```

LIMIT и OFFSET не только упрощают написание решения для MySQL и PostgreSQL, но также делают их предельно понятными.

Обсуждение

DB2, Oracle и SQL Server

Ранжирующая функция ROW_NUMBER OVER во вложенном запросе X присвоит каждой заработной плате уникальный номер (в порядке возрастания, начиная с 1). Ниже представлено результирующее множество вложенного запроса X:

```
select row_number() over (order by sal) as rn,
       sal
  from emp
```

```
RN        SAL
-- ----------
 1        800
 2        950
 3       1100
 4       1250
 5       1250
 6       1300
 7       1500
 8       1600
 9       2450
10       2850
11       2975
```

```
12          3000
13          3000
14          5000
```

Когда каждой заработной плате поставлен в соответствие порядковый номер, остается просто выбирать необходимый диапазон строк, задавая значения RN.

Для пользователей Oracle существует альтернативный вариант: для формирования последовательности номеров строк вместо функции ROW_NUMBER OVER можно использовать функцию ROWNUM:

```
select sal
  from (
select sal, rownum rn
  from (
select sal
  from emp
 order by sal
       )
       )
 where rn between 6 and 10
```

```
  SAL
-----
 1300
 1500
 1600
 2450
 2850
```

Применение ROWNUM вынуждает создавать дополнительный подзапрос. Самый внутренний подзапрос сортирует строки соответственно размеру заработной платы. Следующий подзапрос присваивает этим строкам порядковые номера. И, наконец, самый внешний SELECT возвращает запрашиваемые данные.

MySQL и PostgreSQL

Введение в конструкцию SELECT оператора OFFSET делает перемещение по результатам простой и понятной задачей. Если задать OFFSET равным 0, результаты будут выводиться, начиная с первой строки; если OFFSET равен 5 – с шестой строки; и если OFFSET равен 10 – с одиннадцатой строки. Оператор LIMIT ограничивает число возвращаемых строк. Сочетая эти два оператора, можно задавать, с какой строки и сколько строк результирующего множества должно быть возвращено.

Как пропустить n строк таблицы

Задача

Требуется в результате запроса возвратить служащих из таблицы EMP через одного; т. е. необходимо получить записи для первого, третьего,

пятого служащего и т. д. Например, из следующего результирующего множества:

```
ENAME
--------
ADAMS
ALLEN
BLAKE
CLARK
FORD
JAMES
JONES
KING
MARTIN
MILLER
SCOTT
SMITH
TURNER
WARD
```

мы должны получить:

```
ENAME
----------
ADAMS
BLAKE
FORD
JONES
MARTIN
SCOTT
TURNER
```

Решение

Чтобы пропустить вторую, или четвертую, или *n*-ную строку результирующего множества, необходимо упорядочить его, в противном случае для множества не существует понятия первой, следующей, второй или четвертой строки.

DB2, Oracle и SQL Server

Применяя функцию ROW_NUMBER OVER, пронумеруйте строки, тогда с помощью функции вычисления остатка от деления можно будет пропускать некоторые из них. В DB2 и Oracle функцией вычисления остатка от деления является функция MOD. В SQL Server используется оператор %. В следующем примере с помощью оператора MOD будут пропущены строки под четными номерами:

```
1  select ename
2    from (
3  select row_number() over (order by ename) rn,
4         ename
```

```
5    from emp
6         ) x
7   where mod(rn,2) = 1
```

MySQL и PostgreSQL

Поскольку в этих СУБД нет встроенных функций, реализующих ранжирование или нумерование строк, расположить строки в определенном порядке (в данном примере по именам) поможет скалярный подзапрос. Затем пропустите строки, используя остаток от деления:

```
 1  select x.ename
 2    from (
 3  select a.ename,
 4         (select count(*)
 5            from emp b
 6           where b.ename <= a.ename) as rn
 7    from emp a
 8         ) x
 9   where mod(x.rn,2) = 1
```

Обсуждение

DB2, Oracle и SQL Server

Вызов ранжирующей функции ROW_NUMBER OVER во вложенном запросе X обеспечивает присвоение ранга каждой строке (никаких связей даже между дублирующимися именами). Результаты показаны ниже:

```
select row_number() over (order by ename) rn, ename
  from emp
```

```
RN ENAME
-- --------
 1 ADAMS
 2 ALLEN
 3 BLAKE
 4 CLARK
 5 FORD
 6 JAMES
 7 JONES
 8 KING
 9 MARTIN
10 MILLER
11 SCOTT
12 SMITH
13 TURNER
14 WARD
```

Последний шаг – просто, используя функцию вычисления остатка от деления, пропустить каждую вторую строку.

MySQL и PostgreSQL

С функцией, выполняющей ранжирование или нумерацию строк, можно использовать скалярный подзапрос, чтобы сначала ранжировать имена служащих. Вложенный запрос X определяет ранг каждого имени, как показано ниже:

```
select a.ename,
       (select count(*)
          from emp b
         where b.ename <= a.ename) as rn
  from emp a
```

```
ENAME              RN
---------- ----------
ADAMS               1
ALLEN               2
BLAKE               3
CLARK               4
FORD                5
JAMES               6
JONES               7
KING                8
MARTIN              9
MILLER             10
SCOTT              11
SMITH              12
TURNER             13
WARD               14
```

Заключительный шаг – применить к полученному рангу функцию вычисления остатка от деления, чтобы пропустить определенные строки.

Использование логики OR во внешних объединениях

Задача

Требуется получить для каждого служащего 10 и 20-го отделов информацию об его имени и отделе, а также информацию об отделе (без личной информации) для служащих 30 и 40-го отделов. Сделаем первую попытку решить эту задачу:

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d, emp e
 where d.deptno = e.deptno
   and (e.deptno = 10 or e.deptno = 20)
 order by 2
```

```
ENAME       DEPTNO DNAME          LOC
------- ---------- -------------- -------------
CLARK           10 ACCOUNTING     NEW YORK
KING            10 ACCOUNTING     NEW YORK
MILLER          10 ACCOUNTING     NEW YORK
```

```
SMITH           20 RESEARCH       DALLAS
ADAMS           20 RESEARCH       DALLAS
FORD            20 RESEARCH       DALLAS
SCOTT           20 RESEARCH       DALLAS
JONES           20 RESEARCH       DALLAS
```

Поскольку в данном запросе реализовано внутреннее объединение, в результирующее множество не вошла информация о 30 и 40-м отделах (DEPTNO 30 и 40).

В следующем запросе делается попытка внешнего объединения таблицы EMP с таблицей DEPT, но по-прежнему получается неверный результат:

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d left join emp e
    on (d.deptno = e.deptno)
 where e.deptno = 10
    or e.deptno = 20
 order by 2
```

```
ENAME       DEPTNO DNAME          LOC
------- ---------- -------------- -----------
CLARK           10 ACCOUNTING     NEW YORK
KING            10 ACCOUNTING     NEW YORK
MILLER          10 ACCOUNTING     NEW YORK
SMITH           20 RESEARCH       DALLAS
ADAMS           20 RESEARCH       DALLAS
FORD            20 RESEARCH       DALLAS
SCOTT           20 RESEARCH       DALLAS
JONES           20 RESEARCH       DALLAS
```

Требуется получить такое результирующее множество:

```
ENAME       DEPTNO DNAME          LOC
------- ---------- -------------- ---------
CLARK           10 ACCOUNTING     NEW YORK
KING            10 ACCOUNTING     NEW YORK
MILLER          10 ACCOUNTING     NEW YORK
SMITH           20 RESEARCH       DALLAS
JONES           20 RESEARCH       DALLAS
SCOTT           20 RESEARCH       DALLAS
ADAMS           20 RESEARCH       DALLAS
FORD            20 RESEARCH       DALLAS
                30 SALES          CHICAGO
                40 OPERATIONS     BOSTON
```

Решение

DB2, MySQL, PostgreSQL и SQL Server

Перенесите условие OR (ИЛИ) в оператор JOIN:

```
1  select e.ename, d.deptno, d.dname, d.loc
2    from dept d left join emp e
```

```
3      on (d.deptno = e.deptno
4          and (e.deptno=10 or e.deptno=20))
5   order by 2
```

Альтернативный вариант: можно сначала во вложенном запросе провести фильтрацию по EMP.DEPTNO, а затем осуществить внешнее объединение:

```
1  select e.ename, d.deptno, d.dname, d.loc
2    from dept d
3    left join
4         (select ename, deptno
5            from emp
6           where deptno in ( 10, 20 )
7          ) e on ( e.deptno = d.deptno )
8   order by 2
```

Oracle

При работе с Oracle 9*i* Database или более поздними версиями можно использовать любое решение для продуктов других производителей. Для остальных версий решение строится на выражениях CASE или DECODE. Далее представлено решение с применением CASE:

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d, emp e
 where d.deptno = e.deptno (+)
   and d.deptno = case when e.deptno(+) = 10 then e.deptno(+)
                       when e.deptno(+) = 20 then e.deptno(+)
                  end
 order by 2
```

И вот то же решение, но на этот раз с использованием DECODE:

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d, emp e
 where d.deptno = e.deptno (+)
   and d.deptno = decode(e.deptno(+), 10, e.deptno(+),
                                      20, e.deptno(+))
 order by 2
```

Если применить собственный синтаксис Oracle для внешнего объединения (+) в сочетании с предикатами IN или OR к столбцу, участвующему во внешнем объединении, запрос возвратит ошибку. Выход из ситуации – перенести предикат IN или OR во вложенный запрос:

```
select e.ename, d.deptno, d.dname, d.loc
  from dept d,
      ( select ename, deptno
          from emp
         where deptno in ( 10, 20 )
       ) e
 where d.deptno = e.deptno (+)
 order by 2
```

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

Для данных продуктов предлагаются два решения. В первом условие OR располагается в конструкции JOIN, что делает его частью условия объединения. Тем самым можно фильтровать строки, извлекаемые из таблицы EMP, не теряя отделы с DEPTNO 30 и 40 из таблицы DEPT.

Во втором решении фильтрация перенесена во вложенный запрос. Он фильтрует записи по EMP.DEPTNO и возвращает интересующие нас строки EMP. Затем осуществляется их внешнее объединение с DEPT. Поскольку DEPT находится в левой части левостороннего внешнего объединения, возвращены будут все отделы, включая 30 и 40.

Oracle

Функции CASE и DECODE помогут избежать ошибок, которые возникли бы в случае реализации решения с помощью более старого синтаксиса внешнего объединения. В решении, использующем вложенный запрос E, сначала из таблицы EMP выбираются необходимые строки, а затем выполняется их внешнее объединение с таблицей DEPT.

Выявление строк со взаимообратными значениями

Задача

Имеется таблица, содержащая результаты двух тестов, и требуется найти пары взаимообратных значений. Рассмотрим результирующее множество, возвращаемое представлением V:

```
select *
  from V

TEST1      TEST2
----- ----------
   20         20
   50         25
   20         20
   60         30
   70         90
   80        130
   90         70
  100         50
  110         55
  120         60
  130         80
  140         70
```

Проанализировав эти результаты, видим, что результаты теста 70 для TEST1 и 90 для TEST2 являются взаимообратными (существуют результат 90 для TEST1 и результат 70 для TEST2). Аналогично результаты 80

для TEST1 и 130 для TEST2 являются взаимообратными для результатов 130 теста TEST1 и 80 теста TEST2. Кроме того, результаты 20 теста TEST1 и 20 теста TEST2 взаимообратные для результатов 20 теста TEST2 и 20 теста TEST1. Необходимо выбрать только один набор взаимообратных значений. Должно быть получено следующее результирующее множество:

```
TEST1      TEST2
----- ----------
   20          20
   70          90
   80         130
```

но не такое:

```
TEST1      TEST2
----- ----------
   20          20
   20          20
   70          90
   80         130
   90          70
  130          80
```

Решение

Используя рефлексивное объединение, выберите строки, в которых значения TEST1 равны значениям TEST2 и наоборот:

```
select distinct v1.*
  from V v1, V v2
 where v1.test1  = v2.test2
   and v1.test2  = v2.test1
   and v1.test1 <= v1.test2
```

Обсуждение

В результате рефлексивного объединения получаем декартово произведение, в котором каждый результат TEST1 можно сравнить с каждым результатом TEST2 и наоборот. Представленный ниже запрос выявит строки со взаимообратными значениями:

```
select v1.*
  from V v1, V v2
 where v1.test1 = v2.test2
   and v1.test2 = v2.test1
```

```
TEST1      TEST2
----- ----------
   20          20
   20          20
   20          20
   20          20
```

```
 90          70
130          80
 70          90
 80         130
```

Применение ключевого слова DISTINCT гарантирует удаление дублирующихся строк из окончательного результирующего множества. Последний фильтр предиката WHERE (and V1.TEST1 <= V1.TEST2) обеспечит возвращение только одной пары взаимообратных значений (в которой значение TEST1 меньше или равно значению TEST2).

Как выбрать записи с n-ым количеством наивысших значений

Задача

Требуется ограничить результирующее множество определенным количеством записей, выбор которых производится на основании рангов или сортировки. Например, должны быть получены имена и заработные платы сотрудников с пятью самыми высокими зарплатами.

Решение

Решение этой задачи складывается из двух частей: сначала строки ранжируются на основании любой интересующей нас величины, затем результирующее множество ограничивается необходимым числом строк.[1]

DB2, Oracle и SQL Server

Решение данной задачи зависит от использования ранжирующей функции. То, какая ранжирующая функция будет применяться, определяется тем, как должны обрабатываться связи. В следующем решении используется функция DENSE_RANK, таким образом, все дублирующиеся значения заработной платы будут рассматриваться как одно значение:

```
1  select ename,sal
2    from (
3  select ename, sal,
4         dense_rank() over (order by sal desc) dr
5    from emp
6         ) x
7   where dr <= 5
```

[1] В данном решении выбираются сотрудники с пятью самыми высокими заработными платами, а не пять самых высокооплачиваемых сотрудников, поэтому в результирующем наборе может быть более пяти записей. Если поставить задачу выбрать именно пять первых записей, то в SQL Server можно воспользоваться выражением TOP(5), а в MySQL и PostgreSQL – LIMIT 5. – *Примеч. науч. ред.*

Общее число возвращенных строк может быть больше пяти, но разных заработных плат будет только пять. Используйте ROW_NUMBER OVER, если хотите получить пять строк независимо от связей (поскольку эта функция не учитывает связи).

MySQL и PostgreSQL

Ранжирование заработных плат выполните с помощью скалярного подзапроса. Затем ограничьте результаты этого подзапроса соответственно рангу возвращенных строк:

```
 1  select ename,sal
 2    from (
 3  select (select count(distinct b.sal)
 4            from emp b
 5           where a.sal <= b.sal) as rnk,
 6         a.sal,
 7         a.ename
 8    from emp a
 9         )
10   where rnk <= 5
```

Обсуждение

DB2, Oracle и SQL Server

Все делает ранжирующая функция DENSE_RANK OVER во вложенном запросе X. Следующий пример показывает всю таблицу после применения этой функции:

```
select ename, sal,
       dense_rank() over (order by sal desc) dr
  from emp

ENAME      SAL         DR
------- ------ ----------
KING      5000          1
SCOTT     3000          2
FORD      3000          2
JONES     2975          3
BLAKE     2850          4
CLARK     2450          5
ALLEN     1600          6
TURNER    1500          7
MILLER    1300          8
WARD      1250          9
MARTIN    1250          9
ADAMS     1100         10
JAMES      950         11
SMITH      800         12
```

Теперь осталось только выбрать строки, для которых DR меньше или равно пяти.

MySQL и PostgreSQL

Скалярный подзапрос во вложенном запросе X ранжирует заработные платы следующим образом:

```
select (select count(distinct b.sal)
          from emp b
         where a.sal <= b.sal) as rnk,
       a.sal,
       a.ename
  from emp a
```

```
RNK    SAL ENAME
--- ------ -------
  1   5000 KING
  2   3000 SCOTT
  2   3000 FORD
  3   2975 JONES
  4   2850 BLAKE
  5   2450 CLARK
  6   1600 ALLEN
  7   1500 TURNER
  8   1300 MILLER
  9   1250 WARD
  9   1250 MARTIN
 10   1100 ADAMS
 11    950 JAMES
 12    800 SMITH
```

Заключительный шаг – выбрать только те строки, в которых значение RNK меньше или равно пяти.

Как найти записи с наибольшим и наименьшим значениями

Задача

Требуется найти «крайние» значения (экстремумы) таблицы. Например, стоит задача выбрать из таблицы EMP служащих с наибольшей и наименьшей заработными платами.

Решение

DB2, Oracle и SQL Server

Для поиска наименьшей и наибольшей заработных плат используйте оконные функции MIN OVER и MAX OVER соответственно:

```
1  select ename
2    from (
3  select ename, sal,
4         min(sal)over() min_sal,
5         max(sal)over() max_sal
```

```
6    from emp
7         ) x
8   where sal in (min_sal,max_sal)
```

MySQL и PostgreSQL

Напишите два подзапроса: один для возвращения MIN значения SAL, другой – MAX значения SAL:

```
1  select ename
2    from emp
3   where sal in ( (select min(sal) from emp),
4                  (select max(sal) from emp) )
```

Обсуждение

DB2, Oracle и SQL Server

Оконные функции MIN OVER и MAX OVER обеспечивают возможность доступа к наименьшей и наибольшей заработным платам в каждой строке. Результирующее множество вложенного запроса X следующее:

```
select ename, sal,
       min(sal)over() min_sal,
       max(sal)over() max_sal
  from emp
```

```
ENAME      SAL    MIN_SAL    MAX_SAL
------- ------ ---------- ----------
SMITH       800        800       5000
ALLEN      1600        800       5000
WARD       1250        800       5000
JONES      2975        800       5000
MARTIN     1250        800       5000
BLAKE      2850        800       5000
CLARK      2450        800       5000
SCOTT      3000        800       5000
KING       5000        800       5000
TURNER     1500        800       5000
ADAMS      1100        800       5000
JAMES       950        800       5000
FORD       3000        800       5000
MILLER     1300        800       5000
```

Имея это результирующее множество, осталось лишь выбрать строки, в которых значение SAL равно MIN_SAL или MAX_SAL.

MySQL и PostgreSQL

В этом решении для поиска наименьшей и наибольшей заработных плат используются два подзапроса в одном списке оператора IN. Внешний запрос возвращает строки, в которых значения заработных плат соответствуют значениям, возвращенным хотя бы одним из подзапросов.

Сбор информации из последующих строк

Задача

Требуется найти каждого служащего, который зарабатывает меньше, чем служащий, принятый на работу сразу после него. Исходя из следующего результирующего множества:

```
ENAME             SAL HIREDATE
---------- ---------- ---------
SMITH              800 17-DEC-80
ALLEN             1600 20-FEB-81
WARD              1250 22-FEB-81
JONES             2975 02-APR-81
BLAKE             2850 01-MAY-81
CLARK             2450 09-JUN-81
TURNER            1500 08-SEP-81
MARTIN            1250 28-SEP-81
KING              5000 17-NOV-81
JAMES              950 03-DEC-81
FORD              3000 03-DEC-81
MILLER            1300 23-JAN-82
SCOTT             3000 09-DEC-82
ADAMS             1100 12-JAN-83
```

заработная плата SMITH, WARD, MARTIN, JAMES и MILLER меньше, чем человека, принятого сразу после каждого из них, т. е. запрос должен выбрать записи этих служащих.

Решение

Первый шаг – определить, что значит «последующие». Чтобы можно было определять строку как последующую по отношению к данной, необходимо упорядочить результирующее множество.

DB2, MySQL, PostgreSQL и SQL Server

С помощью подзапросов определите для каждого служащего:

- Дату приема на работу следующего служащего, имеющего более высокую заработную плату.
- Дату приема на работу следующего служащего.

Если эти две даты совпадают, мы нашли то, что искали:

```
1  select ename, sal, hiredate
2    from (
3  select a.ename, a.sal, a.hiredate,
4         (select min(hiredate) from emp b
5           where b.hiredate > a.hiredate
6             and b.sal      > a.sal ) as next_sal_grtr,
7         (select min(hiredate) from emp b
8           where b.hiredate > a.hiredate) as next_hire
```

```
 9    from emp a
10         ) x
11   where next_sal_grtr = next_hire
```

Oracle

С помощью оконной функции LEAD OVER можно получить заработную плату служащего, принятого на работу следующим после рассматриваемого служащего. Затем остается просто сравнить их заработные платы:

```
1  select ename, sal, hiredate
2    from (
3  select ename, sal, hiredate,
4         lead(sal)over(order by hiredate) next_sal
5    from emp
6         )
7   where sal < next_sal
```

Обсуждение

DB2, MySQL, PostgreSQL и SQL Server

Для каждого служащего скалярные подзапросы возвращают значение HIREDATE служащего, принятого на работу сразу после него, и HIREDATE первого служащего из тех, кто был принят на работу позже, но зарабатывает больше рассматриваемого служащего. Рассмотрим исходные данные:

```
select a.ename, a.sal, a.hiredate,
       (select min(hiredate) from emp b
         where b.hiredate > a.hiredate
           and b.sal      > a.sal ) as next_sal_grtr,
       (select min(hiredate) from emp b
         where b.hiredate > a.hiredate) as next_hire
  from emp a
```

```
ENAME      SAL HIREDATE  NEXT_SAL_GRTR NEXT_HIRE
------- ------ --------- ------------- ---------
SMITH      800 17-DEC-80 20-FEB-81     20-FEB-81
ALLEN     1600 20-FEB-81 02-APR-81     22-FEB-81
WARD      1250 22-FEB-81 02-APR-81     02-APR-81
JONES     2975 02-APR-81 17-NOV-81     01-MAY-81
MARTIN    1250 28-SEP-81 17-NOV-81     17-NOV-81
BLAKE     2850 01-MAY-81 17-NOV-81     09-JUN-81
CLARK     2450 09-JUN-81 17-NOV-81     08-SEP-81
SCOTT     3000 09-DEC-82               12-JAN-83
KING      5000 17-NOV-81               03-DEC-81
TURNER    1500 08-SEP-81 17-NOV-81     28-SEP-81
ADAMS     1100 12-JAN-83
JAMES      950 03-DEC-81 23-JAN-82     23-JAN-82
FORD      3000 03-DEC-81               23-JAN-82
MILLER    1300 23-JAN-82 09-DEC-82     09-DEC-82
```

Не обязательно, чтобы те, кто был принят на работу позже, был принят сразу после рассматриваемого служащего. Следующий (и последний) шаг – выбрать только те строки, в которых значение NEXT_SAL_GRTR (наименьшее значение HIREDATE среди служащих, зарабатывающих больше рассматриваемого служащего) равно NEXT_HIRE (ближайшее последующее значение HIREDATE относительно HIREDATE текущего служащего).

Oracle

Оконная функция идеально подходит для решения задач такого типа. С ее использованием запрос становится не только более понятным, чем решения для других продуктов, но LEAD OVER также обеспечивает большую гибкость, потому что может принимать аргумент, определяющий, на сколько строк вперед она должна «заглянуть» (по умолчанию аргумент равен 1). Возможность «перепрыгивать» через несколько строк важна в случае присутствия дублирующихся значений в столбце, по которому осуществляется упорядочивание.

Следующий пример показывает, как просто с помощью LEAD OVER получить заработную плату служащего, принятого на работу «следующим»:

```
select ename, sal, hiredate,
       lead(sal)over(order by hiredate) next_sal
  from emp

ENAME      SAL HIREDATE    NEXT_SAL
------- ------ --------- ----------
SMITH      800 17-DEC-80       1600
ALLEN     1600 20-FEB-81       1250
WARD      1250 22-FEB-81       2975
JONES     2975 02-APR-81       2850
BLAKE     2850 01-MAY-81       2450
CLARK     2450 09-JUN-81       1500
TURNER    1500 08-SEP-81       1250
MARTIN    1250 28-SEP-81       5000
KING      5000 17-NOV-81        950
JAMES      950 03-DEC-81       3000
FORD      3000 03-DEC-81       1300
MILLER    1300 23-JAN-82       3000
SCOTT     3000 09-DEC-82       1100
ADAMS     1100 12-JAN-83
```

Заключительный шаг – выбрать только те строки, в которых значение SAL меньше NEXT_SAL. Поскольку по умолчанию областью действия LEAD OVER является одна строка, в случае присутствия дубликатов в таблице EMP, в частности, если несколько служащих были приняты на работу в один день, сравниваться будут их SAL. Это может не соответствовать требуемому поведению. Если стоит задача сравнивать SAL каждого служащего с SAL служащего, который был принят на работу

следующим, но не в один день с рассматриваемым служащим, в качестве альтернативы можно использовать следующее решение:

```
select ename, sal, hiredate
  from (
select ename, sal, hiredate,
       lead(sal,cnt-rn+1)over(order by hiredate) next_sal
  from (
select ename,sal,hiredate,
       count(*)over(partition by hiredate) cnt,
       row_number()over(partition by hiredate order by empno) rn
  from emp
       )
       )
 where sal < next_sal
```

Основная идея данного решения – найти, на сколько строк текущая строка отстоит от строки, с которой должна сравниваться. Например, если имеется пять дубликатов, первый из пяти должен сравниваться со строкой, отстоящей от него на пять строк. Значение CNT для каждого служащего с дублирующимся значением HIREDATE представляет общее количество дубликатов данного HIREDATE. Значение RN представляет ранг служащего 10-го отдела (DEPTNO 10). Ранги подразделяются по HIREDATE, поэтому только для служащих с дублирующимися HIREDATE это значение может быть больше 1. Сортируются ранги по EMPNO (это выбрано произвольно). Теперь, когда известно общее количество дублирующихся значений и для каждого их них определен ранг, чтобы найти расстояние до следующего значения HIREDATE, необходимо просто из общего числа дубликатов вычесть ранг текущего и прибавить единицу (CNT – RN + 1).

См. также

Дополнительные примеры использования LEAD OVER при наличии дублирующихся значений (и более подробное обсуждение приведенной выше техники) представлены в главе 8 в разделе «Определение интервала времени в днях между текущей и следующей записями» и в главе 10 в разделе «Вычисление разности между значениями строк одной группы или сегмента».

Смещение значений строк

Задача

Требуется выбрать имя и заработную плату каждого служащего, а также наибольшую и наименьшую заработные платы служащих, зарабатывающих соотвественно меньше и больше рассматриваемого. Граничные значения заработных плат замыкаются (первым значением SAL указывается последнее значение SAL и наоборот). Должно быть получено следующее результирующее множество:

```
ENAME             SAL    FORWARD     REWIND
---------- ---------- ---------- ----------
SMITH             800        950       5000
JAMES             950       1100        800
ADAMS            1100       1250        950
WARD             1250       1250       1100
MARTIN           1250       1300       1250
MILLER           1300       1500       1250
TURNER           1500       1600       1300
ALLEN            1600       2450       1500
CLARK            2450       2850       1600
BLAKE            2850       2975       2450
JONES            2975       3000       2850
SCOTT            3000       3000       2975
FORD             3000       5000       3000
KING             5000        800       3000
```

Решение

Для пользователей Oracle оконные функции LEAD OVER и LAG OVER сильно упрощают данную задачу и делают окончательные запросы предельно понятными. Для других СУБД можно использовать скалярные подзапросы, хотя связи будут представлять проблему, из-за которой в СУБД, не поддерживающих оконные функции, возможно лишь неполное решение этой задачи.

DB2, SQL Server, MySQL и PostgreSQL

С помощью скалярного подзапроса для каждой заработной платы найдите следующую и предыдущую:

```
 1  select e.ename, e.sal,
 2         coalesce(
 3           (select min(sal) from emp d where d.sal > e.sal),
 4           (select min(sal) from emp)
 5         ) as forward,
 6         coalesce(
 7           (select max(sal) from emp d where d.sal < e.sal),
 8           (select max(sal) from emp)
 9         ) as rewind
10    from emp e
11   order by 2
```

Oracle

С помощью оконных функций LAG OVER и LEAD OVER организуйте доступ к предыдущей и следующей строкам относительно текущей строки:

```
1 select ename,sal,
2        nvl(lead(sal)over(order by sal),min(sal)over()) forward,
3        nvl(lag(sal)over(order by sal),max(sal)over()) rewind
4   from emp
```

Обсуждение

DB2, SQL Server, MySQL и PostgreSQL

Использование скалярного подзапроса не обеспечивает полного решения данной задачи. Если любые две записи будут содержать одинаковые значения SAL, данное приближение даст сбой, но это лучшее, что можно сделать в отсутствие оконных функций.

Oracle

Оконные функции LAG OVER и LEAD OVER будут (по умолчанию и если не определено ничего другого) возвращать значения предыдущей и последующей строк соответственно. Порядок строк определяется частью ORDER BY конструкции OVER. Если посмотреть на решение, на первом шаге возвращаются следующая и предыдущая строки для текущей строки, причем строки упорядочены по SAL:

```
select ename,sal,
       lead(sal)over(order by sal) forward,
       lag(sal)over(order by sal) rewind
  from emp
```

```
ENAME             SAL    FORWARD     REWIND
---------- ---------- ---------- ----------
SMITH             800        950
JAMES             950       1100        800
ADAMS            1100       1250        950
WARD             1250       1250       1100
MARTIN           1250       1300       1250
MILLER           1300       1500       1250
TURNER           1500       1600       1300
ALLEN            1600       2450       1500
CLARK            2450       2850       1600
BLAKE            2850       2975       2450
JONES            2975       3000       2850
SCOTT            3000       3000       2975
FORD             3000       5000       3000
KING             5000                  3000
```

Обратите внимание, что REWIND равно NULL для служащего SMITH и FORWARD равно NULL для служащего KING. Это объясняется тем, что эти два служащих имеют наименьшую и наибольшую заработные платы соответственно. Требование, предъявленное в разделе «Задача», гласит, что в случае существования в столбцах FORWARD или REWIND значений NULL результаты должны «замыкаться», т. е. для наибольшего значения SAL в FORWARD должно быть помещено наименьшее значение SAL таблицы и для наименьшего значения SAL в REWIND должно быть помещено наибольшее значение SAL таблицы. Оконные функции MIN OVER и MAX OVER, если для них не заданы сегмент или окно (т. е. за оператором OVER следуют пустые круглые

скобки), возвратят наименьшую и наибольшую заработные платы таблицы соответственно. Результаты представлены ниже:

```
select ename,sal,
       nvl(lead(sal)over(order by sal),min(sal)over()) forward,
       nvl(lag(sal)over(order by sal),max(sal)over()) rewind
  from emp
```

```
ENAME             SAL    FORWARD     REWIND
---------- ---------- ---------- ----------
SMITH              800        950       5000
JAMES              950       1100        800
ADAMS             1100       1250        950
WARD              1250       1250       1100
MARTIN            1250       1300       1250
MILLER            1300       1500       1250
TURNER            1500       1600       1300
ALLEN             1600       2450       1500
CLARK             2450       2850       1600
BLAKE             2850       2975       2450
JONES             2975       3000       2850
SCOTT             3000       3000       2975
FORD              3000       5000       3000
KING              5000        800       3000
```

Другое полезное свойство LAG OVER и LEAD OVER – возможность задавать, как далеко вперед или назад требуется уйти. В примере данного рецепта мы заглядываем только на одну строку вперед или назад. Так же просто перейти на три строки вперед и пять строк назад; просто задаем значения 3 и 5, как показано ниже:

```
select ename,sal,
       lead(sal,3)over(order by sal) forward,
       lag(sal,5)over(order by sal) rewind
  from emp
```

```
ENAME             SAL    FORWARD     REWIND
---------- ---------- ---------- ----------
SMITH              800       1250
JAMES              950       1250
ADAMS             1100       1300
WARD              1250       1500
MARTIN            1250       1600
MILLER            1300       2450        800
TURNER            1500       2850        950
ALLEN             1600       2975       1100
CLARK             2450       3000       1250
BLAKE             2850       3000       1250
JONES             2975       5000       1300
SCOTT             3000                  1500
FORD              3000                  1600
KING              5000                  2450
```

Ранжирование результатов

Задача

Требуется ранжировать заработные платы в таблице EMP, учитывая при этом дублирующиеся значения. Должно быть получено следующее результирующее множество:

```
RNK        SAL
--- ----------
  1        800
  2        950
  3       1100
  4       1250
  4       1250
  5       1300
  6       1500
  7       1600
  8       2450
  9       2850
 10       2975
 11       3000
 11       3000
 12       5000
```

Решение

Ранжирующие функции чрезвычайно упрощают ранжирующие запросы. Особенно полезны в данном случае три функции: DENSE_RANK OVER, ROW_NUMBER OVER и RANK OVER.

DB2, Oracle и SQL Server

Поскольку требуется учитывать дубликаты, используйте ранжирующую функцию DENSE_RANK OVER:

```
1 select dense_rank() over(order by sal) rnk, sal
2   from emp
```

MySQL и PostgreSQL

Пока не введены ранжирующие функции, для ранжирования заработных плат используйте скалярный подзапрос:

```
1  select (select count(distinct b.sal)
2            from emp b
3           where b.sal <= a.sal) as rnk,
4         a.sal
5    from emp a
```

Обсуждение

DB2, Oracle и SQL Server

Здесь всю работу выполняет ранжирующая функция DENSE_RANK OVER. В скобках после ключевого слова OVER располагается конструкция ORDER BY, определяющая порядок ранжирования строк. В решении используется выражение ORDER BY SAL, таким образом, строки таблицы EMP ранжируются в порядке возрастания заработной платы.

MySQL и PostgreSQL

Результат решения с использованием скалярного подзапроса аналогичен результату решения с DENSE_RANK, потому что управляющий предикат в скалярном подзапросе определяется по SAL.

Исключение дубликатов

Задача

Требуется выбрать из таблицы EMP разные типы должностей, без дублирования. Результирующее множество должно быть таким:

```
JOB
---------
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

Решение

Все СУБД поддерживают ключевое слово DISTINCT, и его применение, вероятно, является самым простым способом исключения дубликатов из результирующего множества. Однако в этом рецепте также будут рассмотрены еще два метода, позволяющие избавиться от дубликатов.

DB2, Oracle и SQL Server

Конечно, традиционный метод с использованием DISTINCT и иногда GROUP BY (как видно далее в решении для MySQL/PostgreSQL) подходит для этих СУБД. Ниже представлено альтернативное решение, в котором применяется ранжирующая функция ROW_NUMBER OVER:

```
1  select job
2    from (
3  select job,
4         row_number()over(partition by job order by job) rn
5    from emp
6         ) x
7   where rn = 1
```

MySQL и PostgreSQL

Чтобы исключить дубликаты из результирующего множества, используйте ключевое слово DISTINCT:

```
select distinct job
  from emp
```

Кроме того, устранение дубликатов может обеспечить оператор GROUP BY:

```
select job
  from emp
 group by job
```

Обсуждение

DB2, Oracle и SQL Server

Это решение строится на несколько ином подходе к ранжирующим функциям с сегментированием. Применение PARTITION BY в конструкции OVER функции ROW_NUMBER обеспечивает, что для каждой новой должности порядковый номер, возвращаемый ROW_NUMBER, будет сбрасываться до исходного значения, 1. Ниже приведены результаты выполнения вложенного запроса X:

```
select job,
       row_number()over(partition by job order by job) rn
  from emp
```

```
JOB               RN
--------- ----------
ANALYST            1
ANALYST            2
CLERK              1
CLERK              2
CLERK              3
CLERK              4
MANAGER            1
MANAGER            2
MANAGER            3
PRESIDENT          1
SALESMAN           1
SALESMAN           2
SALESMAN           3
SALESMAN           4
```

Каждой строке присваивается порядковый номер. Для каждой должности нумерация начинается заново с 1. Чтобы отсеять дубликаты, надо просто выбрать строки, в которых RN равно 1.

При использовании функции ROW_NUMBER OVER присутствие оператора ORDER BY обязательно (кроме DB2), но не оказывает влияния на результат. Какой из дубликатов будет возвращен, не важно, поскольку возвращается по одной строке для каждой из должностей.

MySQL и PostgreSQL

Первое решение показывает, как с помощью ключевого слова DISTINCT исключить дубликаты из результирующего множества. Помните, что DISTINCT применяется ко всему списку SELECT. Дополнительные столбцы могут и, безусловно, изменят результирующее множество. Рассмотрим, чем отличаются два приведенных ниже запроса:

```
select distinct job
  from emp

JOB
---------
ANALYST
CLERK
MANAGER
PRESIDENT
SALESMAN
```

```
select distinct job, deptno
  from emp

JOB           DEPTNO
--------- ----------
ANALYST           20
CLERK             10
CLERK             20
CLERK             30
MANAGER           10
MANAGER           20
MANAGER           30
PRESIDENT         10
SALESMAN          30
```

После добавления DEPTNO в список SELECT из таблицы EMP будут выбраны уникальные пары значений JOB/DEPTNO.

Второе решение реализует исключение дубликатов с помощью GROUP BY. Хотя GROUP BY используется таким образом довольно часто, необходимо помнить, что GROUP BY и DISTINCT – это два очень разных оператора, и они не являются взаимозаменяемыми. Я включил GROUP BY в это решение, чтобы наше обсуждение было полным, поскольку, несомненно, вы когда-нибудь столкнетесь с таким его применением.

Ход конем

Задача

Требуется получить множество, содержащее имя каждого служащего, отдел, в котором он работает, его заработную плату, дату его приема на работу и заработную плату сотрудника, принятого на работу последним в отделе. Должно быть получено следующее результирующее множество:

```
DEPTNO ENAME             SAL HIREDATE    LATEST_SAL
------ ---------- ---------- ----------- ----------
    10 MILLER           1300 23-JAN-1982       1300
    10 KING             5000 17-NOV-1981       1300
    10 CLARK            2450 09-JUN-1981       1300
    20 ADAMS            1100 12-JAN-1983       1100
    20 SCOTT            3000 09-DEC-1982       1100
    20 FORD             3000 03-DEC-1981       1100
    20 JONES            2975 02-APR-1981       1100
    20 SMITH             800 17-DEC-1980       1100
```

```
30 JAMES          950 03-DEC-1981        950
30 MARTIN        1250 28-SEP-1981        950
30 TURNER        1500 08-SEP-1981        950
30 BLAKE         2850 01-MAY-1981        950
30 WARD          1250 22-FEB-1981        950
30 ALLEN         1600 20-FEB-1981        950
```

Значения столбца LATEST_SAL определяются в результате «хода конем», поскольку схема поиска их в таблице аналогична схеме перемещения шахматного коня. Чтобы найти эти значения, надо сделать «ход конем»: перепрыгнуть в строку, затем развернуться и перепрыгнуть в другой столбец (рис.11.1). Чтобы найти верные значения LATEST_SAL, необходимо сначала для каждого отдела найти (перепрыгнуть в) строку с самой поздней датой HIREDATE и затем выбрать (перепрыгнуть в) столбец SAL этой строки.

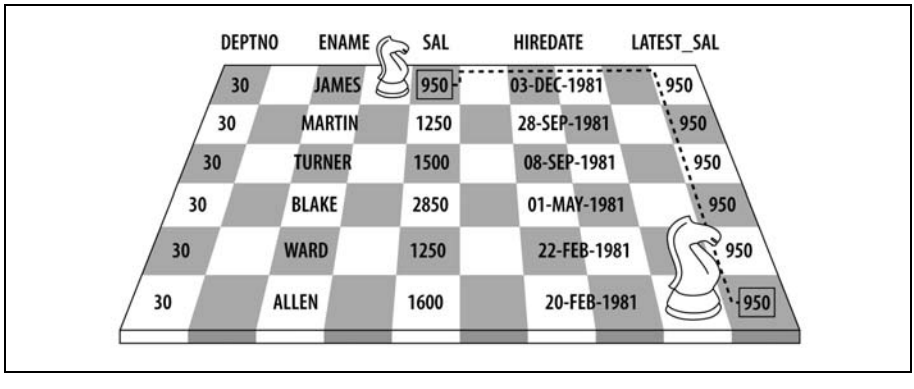


Рис. 11.1. *Ход конем*

Термин «ход конем» придумал один мой очень смышленый сотрудник, Кей Янг. Я дал ему проверить рецепты и признался, что никак не могу придумать хорошее название для этого раздела. Поскольку здесь приходится сначала сформировать строку, а затем «перепрыгнуть» и взять значение из другой, он предложил термин «ход конем».

Решение

DB2 и SQL Server

Чтобы найти для каждого отдела значение SAL служащего, принятого на работу последним, используем в подзапросе выражение CASE. Для всех остальных заработных плат возвращаем нуль. С помощью оконной функции MAX OVER во внешнем запросе получаем значения SAL, не равные нулю, для отдела каждого служащего:

```
1 select deptno,
2        ename,
```

```
 3        sal,
 4        hiredate,
 5        max(latest_sal)over(partition by deptno) latest_sal
 6   from (
 7 select deptno,
 8        ename,
 9        sal,
10        hiredate,
11        case
12          when hiredate = max(hiredate)over(partition by deptno)
13          then sal else 0
14        end latest_sal
15   from emp
16        ) x
17  order by 1, 4 desc
```

MySQL и PostgreSQL

Используем скалярный подзапрос с двумя уровнями вложенности. Сначала находим для всех отделов HIREDATE служащего, принятого на работу последним. Затем используем агрегатную функцию MAX (на случай дублирования значений), чтобы выбрать значение SAL каждого такого служащего:

```
 1  select e.deptno,
 2         e.ename,
 3         e.sal,
 4         e.hiredate,
 5         (select max(d.sal)
 6            from emp d
 7           where d.deptno   = e.deptno
 8             and d.hiredate =
 9                 (select max(f.hiredate)
10                    from emp f
11                   where f.deptno = e.deptno)) as latest_sal
12    from emp e
13   order by 1, 4 desc
```

Oracle

Используем оконную функцию MAX OVER, чтобы получить наибольшее значение SAL для каждого DEPTNO. Применяя в конструкции KEEP функции DENSE_RANK и LAST и при этом упорядочивая по HIREDATE, возвращаем наибольшее значение SAL для самой поздней даты HIREDATE в заданном DEPTNO:

```
1  select deptno,
2         ename,
3         sal,
4         hiredate,
5         max(sal)
6           keep(dense_rank last order by hiredate)
```

```
7           over(partition by deptno) latest_sal
8    from emp
9   order by 1, 4 desc
```

Обсуждение

DB2 и SQL Server

Первый шаг – использовать оконную функцию MAX OVER в выражении CASE, чтобы найти в каждом отделе служащего, который был принят на работу самым последним или позже всех. Если значение HIREDATE служащего совпадает со значением, возвращенным MAX OVER, с помощью выражения CASE возвращаем значение SAL этого служащего; в противном случае возвращаем 0. Результаты показаны ниже:

```
select deptno,
       ename,
       sal,
       hiredate,
       case
           when hiredate = max(hiredate)over(partition by deptno)
           then sal else 0
       end latest_sal
  from emp
```

```
DEPTNO ENAME             SAL HIREDATE    LATEST_SAL
------ ---------- ---------- ----------- ----------
    10 CLARK            2450 09-JUN-1981          0
    10 KING             5000 17-NOV-1981          0
    10 MILLER           1300 23-JAN-1982       1300
    20 SMITH             800 17-DEC-1980          0
    20 ADAMS            1100 12-JAN-1983       1100
    20 FORD             3000 03-DEC-1981          0
    20 SCOTT            3000 09-DEC-1982          0
    20 JONES            2975 02-APR-1981          0
    30 ALLEN            1600 20-FEB-1981          0
    30 BLAKE            2850 01-MAY-1981          0
    30 MARTIN           1250 28-SEP-1981          0
    30 JAMES             950 03-DEC-1981        950
    30 TURNER           1500 08-SEP-1981          0
    30 WARD             1250 22-FEB-1981          0
```

Поскольку LATEST_SAL будет содержать 0 или SAL служащих, принятых на работу последними, приведенный выше запрос можно поместить во вложенный запрос и опять применить MAX OVER, но на этот раз, чтобы получить наибольшее отличное от нуля значение LATEST_SAL для каждого DEPTNO:

```
select deptno,
       ename,
       sal,
       hiredate,
```

```
       max(latest_sal)over(partition by deptno) latest_sal
  from (
select deptno,
       ename,
       sal,
       hiredate,
       case
           when hiredate = max(hiredate)over(partition by deptno)
           then sal else 0
       end latest_sal
  from emp
       ) x
 order by 1, 4 desc
```

```
DEPTNO ENAME             SAL HIREDATE    LATEST_SAL
------ ---------- ---------- ----------- ----------
    10 MILLER           1300 23-JAN-1982       1300
    10 KING             5000 17-NOV-1981       1300
    10 CLARK            2450 09-JUN-1981       1300
    20 ADAMS            1100 12-JAN-1983       1100
    20 SCOTT            3000 09-DEC-1982       1100
    20 FORD             3000 03-DEC-1981       1100
    20 JONES            2975 02-APR-1981       1100
    20 SMITH             800 17-DEC-1980       1100
    30 JAMES             950 03-DEC-1981        950
    30 MARTIN           1250 28-SEP-1981        950
    30 TURNER           1500 08-SEP-1981        950
    30 BLAKE            2850 01-MAY-1981        950
    30 WARD             1250 22-FEB-1981        950
    30 ALLEN            1600 20-FEB-1981        950
```

MySQL и PostgreSQL

Первый шаг – использовать скалярный подзапрос, чтобы найти для каждого отдела HIREDATE служащего, принятого на работу самым последним:

```
select e.deptno,
       e.ename,
       e.sal,
       e.hiredate,
       (select max(f.hiredate)
          from emp f
         where f.deptno = e.deptno) as last_hire
  from emp e
 order by 1, 4 desc
```

```
DEPTNO ENAME             SAL HIREDATE    LAST_HIRE
------ ---------- ---------- ----------- -----------
    10 MILLER           1300 23-JAN-1982 23-JAN-1982
    10 KING             5000 17-NOV-1981 23-JAN-1982
    10 CLARK            2450 09-JUN-1981 23-JAN-1982
    20 ADAMS            1100 12-JAN-1983 12-JAN-1983
```

```
    20 SCOTT           3000 09-DEC-1982 12-JAN-1983
    20 FORD            3000 03-DEC-1981 12-JAN-1983
    20 JONES           2975 02-APR-1981 12-JAN-1983
    20 SMITH            800 17-DEC-1980 12-JAN-1983
    30 JAMES            950 03-DEC-1981 03-DEC-1981
    30 MARTIN          1250 28-SEP-1981 03-DEC-1981
    30 TURNER          1500 08-SEP-1981 03-DEC-1981
    30 BLAKE           2850 01-MAY-1981 03-DEC-1981
    30 WARD            1250 22-FEB-1981 03-DEC-1981
    30 ALLEN           1600 20-FEB-1981 03-DEC-1981
```

Следующий шаг – найти в каждом отделе SAL служащих, дата приема на работу которых совпадает с датой LAST_HIRE. С помощью агрегатной функции MAX выбираем наибольшее значение SAL (если в один день было нанято несколько служащих):

```
select e.deptno,
       e.ename,
       e.sal,
       e.hiredate,
       (select max(d.sal)
          from emp d
         where d.deptno   = e.deptno
           and d.hiredate =
               (select max(f.hiredate)
                  from emp f
                 where f.deptno = e.deptno)) as latest_sal
  from emp e
 order by 1, 4 desc
```

```
DEPTNO ENAME             SAL HIREDATE    LATEST_SAL
------ ---------- ---------- ----------- ----------
    10 MILLER           1300 23-JAN-1982       1300
    10 KING             5000 17-NOV-1981       1300
    10 CLARK            2450 09-JUN-1981       1300
    20 ADAMS            1100 12-JAN-1983       1100
    20 SCOTT            3000 09-DEC-1982       1100
    20 FORD             3000 03-DEC-1981       1100
    20 JONES            2975 02-APR-1981       1100
    20 SMITH             800 17-DEC-1980       1100
    30 JAMES             950 03-DEC-1981        950
    30 MARTIN           1250 28-SEP-1981        950
    30 TURNER           1500 08-SEP-1981        950
    30 BLAKE            2850 01-MAY-1981        950
    30 WARD             1250 22-FEB-1981        950
    30 ALLEN            1600 20-FEB-1981        950
```

Oracle

Пользователям Oracle 8*i* Database подойдет решение для DB2. Те, кто работает с Oracle 9*i* Database и более поздними версиями, могут использовать решение, представленное ниже. Ключ к решению для Oracle –

оператор KEEP. Он позволяет ранжировать возвращенный сегмент/группу строк и работать с первой или последней строкой группы. Рассмотрим решение без KEEP:

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal) over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

```
DEPTNO ENAME             SAL HIREDATE    LATEST_SAL
------ ---------- ---------- ----------- ----------
    10 MILLER           1300 23-JAN-1982       5000
    10 KING             5000 17-NOV-1981       5000
    10 CLARK            2450 09-JUN-1981       5000
    20 ADAMS            1100 12-JAN-1983       3000
    20 SCOTT            3000 09-DEC-1982       3000
    20 FORD             3000 03-DEC-1981       3000
    20 JONES            2975 02-APR-1981       3000
    20 SMITH             800 17-DEC-1980       3000
    30 JAMES             950 03-DEC-1981       2850
    30 MARTIN           1250 28-SEP-1981       2850
    30 TURNER           1500 08-SEP-1981       2850
    30 BLAKE            2850 01-MAY-1981       2850
    30 WARD             1250 22-FEB-1981       2850
    30 ALLEN            1600 20-FEB-1981       2850
```

Вместо того чтобы возвращать SAL служащего, принятого на работу последним, MAX OVER без KEEP возвращает самую высокую заработную плату для каждого отдела. В данном рецепте в конструкции KEEP задано ORDER BY HIREDATE, что обеспечивает упорядочение заработных плат каждого отдела по HIREDATE. Затем функция DENSE_RANK ранжирует все HIREDATE в порядке по возрастанию. Наконец, функция LAST определяет, к какой строке применять агрегатную функцию: к «последней» строке соответственно рангам, назначенным DENSE_RANK. В данном случае агрегатная функция MAX применяется к столбцу SAL для строки с «последней» датой HIREDATE. По сути, для каждого отдела выбирается значение SAL, соответствующее HIREDATE с наивысшим рангом.

Ранжирование строк для каждого DEPTNO осуществляется по одному столбцу (HIREDATE), а агрегация (MAX) выполняется по другому столбцу (SAL). Эта способность назначать ранги по одной величине и агрегировать по другой очень удобна, поскольку позволяет избежать применения дополнительных объединений и вложенных запросов, как в других решениях. Наконец, применив оператор OVER после KEEP, можно возвратить значение SAL, «выбранное» KEEP для каждой строки сегмента.

Или можно упорядочить строки по HIREDATE в порядке по убыванию и выбрать первое значение SAL. Сравните два приведенных ниже запроса, в результате их выполнения возвращается одно и то же результирующее множество:

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal)
         keep(dense_rank last order by hiredate)
         over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

```
DEPTNO ENAME             SAL HIREDATE    LATEST_SAL
------ ---------- ---------- ----------- ----------
    10 MILLER           1300 23-JAN-1982       1300
    10 KING             5000 17-NOV-1981       1300
    10 CLARK            2450 09-JUN-1981       1300
    20 ADAMS            1100 12-JAN-1983       1100
    20 SCOTT            3000 09-DEC-1982       1100
    20 FORD             3000 03-DEC-1981       1100
    20 JONES            2975 02-APR-1981       1100
    20 SMITH             800 17-DEC-1980       1100
    30 JAMES             950 03-DEC-1981        950
    30 MARTIN           1250 28-SEP-1981        950
    30 TURNER           1500 08-SEP-1981        950
    30 BLAKE            2850 01-MAY-1981        950
    30 WARD             1250 22-FEB-1981        950
    30 ALLEN            1600 20-FEB-1981        950
```

```
select deptno,
       ename,
       sal,
       hiredate,
       max(sal)
         keep(dense_rank first order by hiredate desc)
         over(partition by deptno) latest_sal
  from emp
 order by 1, 4 desc
```

```
DEPTNO ENAME             SAL HIREDATE    LATEST_SAL
------ ---------- ---------- ----------- ----------
    10 MILLER           1300 23-JAN-1982       1300
    10 KING             5000 17-NOV-1981       1300
    10 CLARK            2450 09-JUN-1981       1300
    20 ADAMS            1100 12-JAN-1983       1100
    20 SCOTT            3000 09-DEC-1982       1100
    20 FORD             3000 03-DEC-1981       1100
    20 JONES            2975 02-APR-1981       1100
    20 SMITH             800 17-DEC-1980       1100
```

```
30 JAMES            950 03-DEC-1981        950
30 MARTIN          1250 28-SEP-1981        950
30 TURNER          1500 08-SEP-1981        950
30 BLAKE           2850 01-MAY-1981        950
30 WARD            1250 22-FEB-1981        950
30 ALLEN           1600 20-FEB-1981        950
```

Формирование простых прогнозов

Задача

Исходя из текущих данных требуется получить дополнительные строки и столбцы, представляющие будущие действия. Например, рассмотрим следующее результирующее множество:

```
ID ORDER_DATE  PROCESS_DATE
-- ----------- ------------
 1 25-SEP-2005  27-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 3 27-SEP-2005  29-SEP-2005
```

Для каждой строки результирующего множества требуется возвратить три строки (строка плюс две дополнительные строки для каждого заказа). Кроме дополнительных строк, должны быть добавлены столбцы с предполагаемыми датами обработки заказов.

Из представленного выше результирующего множества видно, что обработка заказа занимает два дня. Для примера предположим, что следующий этап после обработки заказа – контроль, и последний этап – поставка. Контроль выполняется на следующий день после обработки, и поставка осуществляется на следующий день после контроля. Необходимо получить результирующее множество, отражающее всю процедуру. В итоге представленное выше результирующее множество должно быть преобразовано в следующее:

```
ID ORDER_DATE  PROCESS_DATE VERIFIED    SHIPPED
-- ----------- ------------ ----------- -----------
 1 25-SEP-2005  27-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005 01-OCT-2005
```

Решение

Ход решения – с помощью декартова произведения получить по две дополнительные строки на каждый заказ и затем просто использовать выражения CASE для создания необходимых значений столбцов.

DB2 и SQL Server

Чтобы сформировать строки, необходимые для декартова произведения, используйте рекурсивный оператор WITH. Решения для DB2 и SQL Server практически идентичны и отличаются лишь функциями, применяемыми для получения текущей даты. DB2 использует функцию CURRENT_DATE, а SQL Server – функцию GETDATE. Ниже показано решение для SQL Server:

```
with nrows(n) as (
 select 1 from t1 union all
 select n+1 from nrows where n+1 <= 3
 )
 select id,
        order_date,
        process_date,
        case when nrows.n >= 2
             then process_date+1
             else null
        end as verified,
        case when nrows.n = 3
             then process_date+2
             else null
        end  as shipped
   from (
 select nrows.n id,
        getdate()+nrows.n   as order_date,
        getdate()+nrows.n+2 as process_date
   from nrows
        ) orders, nrows
  order by 1
```

Oracle

Чтобы получить строки, необходимые для декартова произведения, используйте иерархический оператор CONNECT BY. Оператор WITH обеспечит возможность использовать результаты CONNECT BY без его повторного вызова:

```
with nrows as (
 select level n
   from dual
connect by level <= 3
 )
 select id,
        order_date,
        process_date,
        case when nrows.n >= 2
             then process_date+1
             else null
        end as verified,
        case when nrows.n = 3
             then process_date+2
```

```
15              else null
16         end  as shipped
17    from (
18  select nrows.n id,
19         sysdate+nrows.n   as order_date,
20         sysdate+nrows.n+2 as process_date
21    from nrows
22         ) orders, nrows
```

PostgreSQL

Существует множество способов создания декартова произведения; в данном решении используется функция PostgreSQL GENERATE_SERIES:

```
 1  select id,
 2         order_date,
 3         process_date,
 4         case when gs.n >= 2
 5              then process_date+1
 6              else null
 7         end as verified,
 8         case when gs.n = 3
 9              then process_date+2
10              else null
11         end  as shipped
12    from (
13  select gs.id,
14         current_date+gs.id   as order_date,
15         current_date+gs.id+2 as process_date
16    from generate_series(1,3) gs (id)
17         ) orders,
18           generate_series(1,3)gs(n)
```

MySQL

MySQL не поддерживает функцию для автоматического формирования строк.

Обсуждение

DB2 и SQL Server

Результирующее множество, представленное в разделе «Задача», получено посредством вложенного запроса ORDERS и показано ниже:

```
with nrows(n) as (
select 1 from t1 union all
select n+1 from nrows where n+1 <= 3
)
select nrows.n id,
       getdate()+nrows.n   as order_date,
       getdate()+nrows.n+2 as process_date
```

```
  from nrows

ID ORDER_DATE  PROCESS_DATE
-- ----------- ------------
 1 25-SEP-2005  27-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 3 27-SEP-2005  29-SEP-2005
```

Приведенный выше запрос для составления трех строк, представляющих заказы, которые подлежат обработке, просто использует оператор WITH. NROWS возвращает значения 1, 2 и 3. Эти числа добавляются к результату GETDATE (CURRENT_DATE для DB2) для получения дат выполнения заказов. Поскольку в разделе «Задача» определено, что обработка заказа занимает два дня, для получения даты окончания выполнения заказа в приведенном выше запросе к значению ORDER_DATE добавляется два дня (к GETDATE прибавляем значение, возвращенное NROWS, и еще два дня).

Получив базовое результирующее множество, перейдем к созданию декартова произведения, поскольку стоит задача получить по три строки для каждого заказа. Для этого используем рекурсивное представление NROWS:

```
with nrows(n) as (
select 1 from t1 union all
select n+1 from nrows where n+1 <= 3
)
select nrows.n,
       orders.*
  from (
select nrows.n id,
       getdate()+nrows.n   as order_date,
       getdate()+nrows.n+2 as process_date
  from nrows
       ) orders, nrows
 order by 2,1

  N  ID ORDER_DATE  PROCESS_DATE
--- --- ----------- ------------
  1   1 25-SEP-2005  27-SEP-2005
  2   1 25-SEP-2005  27-SEP-2005
  3   1 25-SEP-2005  27-SEP-2005
  1   2 26-SEP-2005  28-SEP-2005
  2   2 26-SEP-2005  28-SEP-2005
  3   2 26-SEP-2005  28-SEP-2005
  1   3 27-SEP-2005  29-SEP-2005
  2   3 27-SEP-2005  29-SEP-2005
  3   3 27-SEP-2005  29-SEP-2005
```

Теперь, имея по три строки для каждого заказа, просто с помощью выражения CASE создаем дополнительные столбцы для представления статуса контроля и поставки.

Первая строка каждого заказа в столбцах VERIFIED (контроль прошел) и SHIPPED (поставлен) должна содержать значения NULL. Во второй строке значение NULL установлено для столбца SHIPPED. В третьей строке каждого заказа ни в одном столбце не должно быть значения NULL. Окончательный результат показан ниже:

```
with nrows(n) as (
select 1 from t1 union all
select n+1 from nrows where n+1 <= 3
)
select id,
       order_date,
       process_date,
       case when nrows.n >= 2
            then process_date+1
            else null
       end as verified,
       case when nrows.n = 3
            then process_date+2
            else null
       end  as shipped
  from (
select nrows.n id,
       getdate()+nrows.n   as order_date,
       getdate()+nrows.n+2 as process_date
  from nrows
       ) orders, nrows
 order by 1

ID ORDER_DATE  PROCESS_DATE VERIFIED    SHIPPED
-- ----------- ------------ ----------- -----------
 1 25-SEP-2005  27-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005 01-OCT-2005
```

Окончательное результирующее множество представляет весь процесс обработки заказа, от дня получения до дня, когда он должен быть поставлен.

Oracle

Результирующее множество, представленное в разделе «Задача», получено посредством вложенного запроса ORDERS и показано ниже:

```
with nrows as (
select level n
```

```
  from dual
connect by level <= 3
)
select nrows.n id,
       sysdate+nrows.n   order_date,
       sysdate+nrows.n+2 process_date
  from nrows

ID ORDER_DATE  PROCESS_DATE
-- ----------- ------------
 1 25-SEP-2005  27-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 3 27-SEP-2005  29-SEP-2005
```

Приведенный выше запрос для составления трех строк, представляющих заказы, которые подлежат обработке, просто использует оператор CONNECT BY. Используем оператор WITH, чтобы обращаться к строкам, возвращенным CONNECT BY, через NROWS.N. CONNECT BY возвращает значения 1, 2 и 3. Эти числа добавляются к результату SYSDATE для представления дат получения заказов. Поскольку в разделе «Задача» определено, что обработка заказа занимает два дня, для получения даты окончания выполнения заказа в приведенном выше запросе к значению ORDER_DATE добавляется два дня (к SYSDATE прибавляем значение, возвращенное GENERATE_SERIES, и еще два дня).

Получив базовое результирующее множество, перейдем к созданию декартова произведения, поскольку стоит задача получить по три строки для каждого заказа. Для этого используем функцию NROWS:

```
with nrows as (
select level n
  from dual
connect by level <= 3
)
select nrows.n,
       orders.*
  from (
select nrows.n id,
       sysdate+nrows.n   order_date,
       sysdate+nrows.n+2 process_date
  from nrows
       ) orders, nrows

  N  ID ORDER_DATE  PROCESS_DATE
--- --- ----------- ------------
  1   1 25-SEP-2005  27-SEP-2005
  2   1 25-SEP-2005  27-SEP-2005
  3   1 25-SEP-2005  27-SEP-2005
  1   2 26-SEP-2005  28-SEP-2005
  2   2 26-SEP-2005  28-SEP-2005
  3   2 26-SEP-2005  28-SEP-2005
```

```
  1   3 27-SEP-2005  29-SEP-2005
  2   3 27-SEP-2005  29-SEP-2005
  3   3 27-SEP-2005  29-SEP-2005
```

Теперь, имея по три строки для каждого заказа, просто с помощью выражения CASE создаем дополнительные столбцы для представления статуса контроля и поставки.

Первая строка каждого заказа в столбцах VERIFIED и SHIPPED должна содержать значения NULL. Во второй строке значение NULL установлено для столбца SHIPPED. В третьей строке каждого заказа ни в одном столбце не должно быть значения NULL. Окончательный результат показан ниже:

```
with nrows as (
select level n
  from dual
connect by level <= 3
)
select id,
       order_date,
       process_date,
       case when nrows.n >= 2
            then process_date+1
            else null
       end as verified,
       case when nrows.n = 3
            then process_date+2
            else null
       end  as shipped
  from (
select nrows.n id,
       sysdate+nrows.n   order_date,
       sysdate+nrows.n+2 process_date
  from nrows
       ) orders, nrows

ID ORDER_DATE  PROCESS_DATE VERIFIED    SHIPPED
-- ----------- ------------ ----------- -----------
 1 25-SEP-2005  27-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005 01-OCT-2005
```

Окончательное результирующее множество представляет весь процесс обработки заказа, от дня получения до дня, когда он должен быть поставлен.

PostgreSQL

Результирующее множество, представленное в разделе «Задача», получено посредством вложенного запроса ORDERS и показано ниже:

```
select gs.id,
       current_date+gs.id   as order_date,
       current_date+gs.id+2 as process_date
  from generate_series(1,3) gs (id)

ID ORDER_DATE  PROCESS_DATE
-- ----------- ------------
 1 25-SEP-2005  27-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 3 27-SEP-2005  29-SEP-2005
```

Приведенный выше запрос для составления трех строк, представляющих заказы, которые подлежат обработке, просто использует оператор GENERATE_SERIES. GENERATE_SERIES возвращает значения 1, 2 и 3. Эти числа добавляются к CURRENT_DATE для представления дат получения заказов. Поскольку в разделе «Задача» определено, что обработка заказа занимает два дня, для получения даты окончания выполнения заказа в приведенном выше запросе к значению ORDER_DATE добавляется два дня (к CURRENT_DATE прибавляем значение, возвращенное GENERATE_SERIES, и еще два дня).

Получив базовое результирующее множество, перейдем к созданию декартова произведения, поскольку стоит задача получить по три строки для каждого заказа. Для этого используем функцию GENERATE_SERIES:

```
select gs.n,
       orders.*
  from (
select gs.id,
       current_date+gs.id   as order_date,
       current_date+gs.id+2 as process_date
  from generate_series(1,3) gs (id)
       ) orders,
         generate_series(1,3)gs(n)

  N  ID ORDER_DATE  PROCESS_DATE
--- --- ----------- ------------
  1   1 25-SEP-2005  27-SEP-2005
  2   1 25-SEP-2005  27-SEP-2005
  3   1 25-SEP-2005  27-SEP-2005
  1   2 26-SEP-2005  28-SEP-2005
  2   2 26-SEP-2005  28-SEP-2005
  3   2 26-SEP-2005  28-SEP-2005
  1   3 27-SEP-2005  29-SEP-2005
  2   3 27-SEP-2005  29-SEP-2005
  3   3 27-SEP-2005  29-SEP-2005
```

Теперь, имея по три строки для каждого заказа, просто с помощью выражения CASE создаем дополнительные столбцы для представления статуса контроля и поставки.

Первая строка каждого заказа в столбцах VERIFIED и SHIPPED должна содержать значения NULL. Во второй строке значение NULL установлено для столбца SHIPPED. В третьей строке каждого заказа ни в одном столбце не должно быть значения NULL. Окончательный результат показан ниже:

```
select id,
       order_date,
       process_date,
       case when gs.n >= 2
            then process_date+1
            else null
       end as verified,
       case when gs.n = 3
            then process_date+2
            else null
       end  as shipped
  from (
select gs.id,
       current_date+gs.id   as order_date,
       current_date+gs.id+2 as process_date
  from generate_series(1,3) gs(id)
       ) orders,
         generate_series(1,3)gs(n)

ID ORDER_DATE  PROCESS_DATE VERIFIED    SHIPPED
-- ----------- ------------ ----------- -----------
 1 25-SEP-2005  27-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005
 1 25-SEP-2005  27-SEP-2005 28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005
 2 26-SEP-2005  28-SEP-2005 29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005
 3 27-SEP-2005  29-SEP-2005 30-SEP-2005 01-OCT-2005
```

Окончательное результирующее множество представляет весь процесс обработки заказа, от дня получения до дня, когда он должен быть поставлен.

12

Составление отчетов и управление хранилищами данных

В данной главе представлены запросы, используемые для создания отчетов. При составлении отчетов обычно применяются специфическое форматирование и различные уровни агрегации. Другой объект рассмотрения данной главы – транспонирование или разворачивание результирующих множеств, преобразование строк в столбцы. Разворачивание – исключительно полезная техника для решения разнообразных задач. Освоив ее, вы найдете ей применение и за рамками вопросов, обсуждаемых здесь.

Разворачивание результирующего множества в одну строку

Задача

Требуется развернуть группу строк, превращая их значения в столбцы. Каждой группе строк должна соответствовать одна строка. Например, имеется результирующее множество, отражающее количество служащих в каждом отделе:

```
DEPTNO        CNT
------ ----------
    10          3
    20          5
    30          6
```

Необходимо переформатировать результат так, чтобы множество выглядело следующим образом:

```
DEPTNO_10  DEPTNO_20  DEPTNO_30
--------- ---------- ----------
        3          5          6
```

Решение

Транспонируйте результирующее множество с помощью выражения CASE и агрегатной функции SUM:

```
1  select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
2         sum(case when deptno=20 then 1 else 0 end) as deptno_20,
3         sum(case when deptno=30 then 1 else 0 end) as deptno_30
4    from emp
```

Обсуждение

Данный пример является превосходным введением в разворачивание таблиц. Принцип прост: к каждой строке, возвращенной запросом, применяем выражение CASE, чтобы разложить строки в столбцы. Затем, поскольку стоит конкретная задача пересчитать служащих в каждом отделе, с помощью агрегатной функции SUM подсчитываем количество экземпляров каждого значения DEPTNO. Если что-то не понятно, выполните запрос без агрегатной функции SUM и включите в него DEPTNO для удобства чтения:

```
select deptno,
       case when deptno=10 then 1 else 0 end as deptno_10,
       case when deptno=20 then 1 else 0 end as deptno_20,
       case when deptno=30 then 1 else 0 end as deptno_30
  from emp
 order by 1

DEPTNO  DEPTNO_10  DEPTNO_20  DEPTNO_30
------ ---------- ---------- ----------
    10          1          0          0
    10          1          0          0
    10          1          0          0
    20          0          1          0
    20          0          1          0
    20          0          1          0
    20          0          1          0
    20          0          1          0
    30          0          0          1
    30          0          0          1
    30          0          0          1
    30          0          0          1
    30          0          0          1
    30          0          0          1
```

Выражения CASE, так сказать, расставляют флаги, обозначая, к какому DEPTNO относится строка. На данный момент преобразование «строк в столбцы» уже выполнено. Осталось просто сложить значения, возвращенные в столбцах DEPTNO_10, DEPTNO_20 и DEPTNO_30, и сгруппировать их по DEPTNO. Ниже представлены результаты:

```
select deptno,
       sum(case when deptno=10 then 1 else 0 end) as deptno_10,
```

```
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
 group by deptno
```

```
DEPTNO  DEPTNO_10  DEPTNO_20  DEPTNO_30
------ ---------- ---------- ----------
    10          3          0          0
    20          0          5          0
    30          0          0          6
```

Если внимательно посмотреть на это результирующее множество, то станет ясно, что с логической точки зрения такой вывод имеет смысл: например, в столбце DEPTNO 10 для DEPTNO_10 указано 3 служащих и нуль для других отделов. Поскольку поставлена цель возвратить одну строку, последний шаг – убрать DEPTNO и GROUP BY и просто суммировать выражения CASE:

```
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
```

```
DEPTNO_10  DEPTNO_20  DEPTNO_30
--------- ---------- ----------
        3          5          6
```

Далее представлен другой подход, который иногда применяют для решения такого рода задач:

```
select max(case when deptno=10 then empcount else null end) as deptno_10
       max(case when deptno=20 then empcount else null end) as deptno_20,
       max(case when deptno=30 then empcount else null end) as deptno_30
  from (
select deptno, count(*) as empcount
  from emp
 group by deptno
       ) x
```

В этом подходе для подсчета количества служащих в отделе используется вложенный запрос. Выражения CASE основного запроса преобразуют строки в столбцы, обеспечивая следующие результаты:

```
DEPTNO_10  DEPTNO_20  DEPTNO_30
--------- ---------- ----------
        3       NULL       NULL
     NULL          5       NULL
     NULL       NULL          6
```

Затем функция MAX сворачивает столбцы в одну строку:

```
DEPTNO_10  DEPTNO_20  DEPTNO_30
--------- ---------- ----------
        3          5          6
```

Разворачивание результирующего множества в несколько строк

Задача

Требуется преобразовать строки в столбцы, создавая для каждого значения заданного столбца отдельный столбец. Однако в отличие от предыдущего рецепта выведено должно быть несколько строк.

Например, требуется выбрать всех служащих и их должности (JOB). В нашем распоряжении имеется следующее результирующее множество:

```
JOB       ENAME
--------- ----------
ANALYST   SCOTT
ANALYST   FORD
CLERK     SMITH
CLERK     ADAMS
CLERK     MILLER
CLERK     JAMES
MANAGER   JONES
MANAGER   CLARK
MANAGER   BLAKE
PRESIDENT KING
SALESMAN  ALLEN
SALESMAN  MARTIN
SALESMAN  TURNER
SALESMAN  WARD
```

Хотелось бы отформатировать это множество так, чтобы каждая должность была представлена отдельным столбцом:

```
CLERKS ANALYSTS MGRS  PREZ SALES
------ -------- ----- ---- ------
MILLER FORD     CLARK KING TURNER
JAMES  SCOTT    BLAKE      MARTIN
ADAMS           JONES      WARD
SMITH                      ALLEN
```

Решение

В отличие от первого рецепта данной главы данное результирующее множество состоит из нескольких строк. Техника, применявшаяся в предыдущем рецепте, здесь не подходит, потому что обеспечит возвращение MAX(ENAME) для каждого JOB, т. е. одного ENAME для каждого JOB (т. е. мы получим одну строку, как в первом рецепте). Чтобы решить поставленную задачу, необходимо сделать каждое сочетание JOB/ENAME уникальным. Тогда при использовании агрегатной функции для удаления значений NULL не будет утрачено ни одно значение ENAME.

DB2, Oracle и SQL Server

Используя ранжирующую функцию ROW_NUMBER OVER, сделайте каждое сочетание JOB/ENAME уникальным. Разверните результирующее множество с помощью выражения CASE и агрегатной функции MAX, группируя при этом по значению, возвращенному ранжирующей функцией:

```
select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
  from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
  from emp
       ) x
 group by rn
```

PostgreSQL и MySQL

Скалярным подзапросом ранжируйте всех служащих по EMPNO. Разверните результирующее множество с помощью выражения CASE и агрегатной функции MAX, группируя при этом по значению, возвращенному скалярным подзапросом:

```
select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
  from (
select e.job,
       e.ename,
       (select count(*) from emp d
         where e.job=d.job and e.empno < d.empno) as rnk
  from emp e
       ) x
 group by rnk
```

Обсуждение

DB2, Oracle и SQL Server

Первый шаг – с помощью ранжирующей функции ROW_NUMBER OVER сделать каждое сочетание JOB/ENAME уникальным:

```
select job,
       ename,
       row_number()over(partition by job order by ename) rn
  from emp
```

```
JOB       ENAME              RN
--------- ---------- ----------
ANALYST   FORD                1
ANALYST   SCOTT               2
CLERK     ADAMS               1
CLERK     JAMES               2
CLERK     MILLER              3
CLERK     SMITH               4
MANAGER   BLAKE               1
MANAGER   CLARK               2
MANAGER   JONES               3
PRESIDENT KING                1
SALESMAN  ALLEN               1
SALESMAN  MARTIN              2
SALESMAN  TURNER              3
SALESMAN  WARD                4
```

Присвоение каждому ENAME уникального для данной должности «номера строки» предотвращает появление любых проблем, которые могли бы возникнуть в случае существования двух служащих с одинаковыми именем и должностью. Целью является обеспечение возможности группировки по номеру строки (по столбцу RN) без исключения служащих из результирующего множества из-за применения функции MAX. Данный шаг – самый важный в решении поставленной задачи. Если не выполнить этот первый шаг, внешний запрос в результате агрегации удалит нужные строки. Рассмотрим, как выглядело бы результирующее множество без использования функции ROW_NUMBER OVER, если бы применялась техника, представленная в первом рецепте:

```
select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
  from emp
```

```
CLERKS     ANALYSTS   MGRS       PREZ       SALES
---------- ---------- ---------- ---------- ----------
SMITH      SCOTT      JONES      KING       WARD
```

К сожалению, для каждого значения JOB возвращена только одна строка: служащий с наивысшим рангом. При разворачивании результирующего множества функции MIN или MAX должны использоваться только как средства для удаления значений NULL, без ограничения возвращаемых значений ENAME. Как этого добиться, станет понятно в ходе обсуждения.

Следующий шаг – использовать выражение CASE для распределения значений ENAME по соответствующим столбцам (JOB):

```
select rn,
       case when job='CLERK'
            then ename else null end as clerks,
       case when job='ANALYST'
            then ename else null end as analysts,
       case when job='MANAGER'
            then ename else null end as mgrs,
       case when job='PRESIDENT'
            then ename else null end as prez,
       case when job='SALESMAN'
            then ename else null end as sales
  from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
  from emp
       ) x
```

```
RN CLERKS     ANALYSTS   MGRS       PREZ       SALES
-- ---------- ---------- ---------- ---------- ----------
 1            FORD
 2            SCOTT
 1 ADAMS
 2 JAMES
 3 MILLER
 4 SMITH
 1                       BLAKE
 2                       CLARK
 3                       JONES
 1                                  KING
 1                                             ALLEN
 2                                             MARTIN
 3                                             TURNER
 4                                             WARD
```

На данный момент строки транспонированы в столбцы, осталось лишь удалить значения NULL, чтобы сделать результирующее множество более удобным для восприятия. Удаляем значения NULL с помощью агрегатной функции MAX и группируем результаты по RN. (Можно исполь-

зовать и функцию MIN. Выбор MAX произволен, поскольку в каждой группе всегда осуществляется агрегация всего одного значения.) Сочетания значений RN/JOB/ENAME уникальны. Группировка по RN в сочетании с выражениями CASE, вложенными в вызовы MAX, гарантирует, что каждый вызов MAX обеспечит выбор единственного имени из группы, все остальные значения которой являются значениями NULL:

```
select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
  from (
select job,
       ename,
       row_number()over(partition by job order by ename) rn
  from emp
       ) x
 group by rn
```

```
CLERKS ANALYSTS MGRS  PREZ SALES
------ -------- ----- ---- ------
MILLER FORD     CLARK KING TURNER
JAMES  SCOTT    BLAKE      MARTIN
ADAMS           JONES      WARD
SMITH                      ALLEN
```

Методика использования ROW_NUMBER OVER для создания уникальных сочетаний строк исключительно полезна для форматирования результатов запросов. Рассмотрим запрос, создающий разреженный отчет, в котором служащие распределены по DEPTNO и JOB:

```
select deptno dno, job,
       max(case when deptno=10
                then ename else null end) as d10,
       max(case when deptno=20
                then ename else null end) as d20,
       max(case when deptno=30
                then ename else null end) as d30,
       max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as anals,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
```

```
       max(case when job='SALESMAN'
                then ename else null end) as sales
  from (
select deptno,
       job,
       ename,
       row_number()over(partition by job order by ename) rn_job,
       row_number()over(partition by job order by ename) rn_deptno
  from emp
       ) x
 group by deptno, job, rn_deptno, rn_job
 order by 1
```

```
DNO JOB       D10    D20   D30    CLERKS ANALS MGRS  PREZ SALES
--- --------- ------ ----- ------ ------ ----- ----- ---- ------
 10 CLERK     MILLER              MILLER
 10 MANAGER   CLARK                           CLARK
 10 PRESIDENT KING                                  KING
 20 ANALYST           FORD                FORD
 20 ANALYST           SCOTT               SCOTT
 20 CLERK             ADAMS        ADAMS
 20 CLERK             SMITH        SMITH
 20 MANAGER           JONES                      JONES
 30 CLERK                    JAMES  JAMES
 30 MANAGER                  BLAKE               BLAKE
 30 SALESMAN                 ALLEN                      ALLEN
 30 SALESMAN                 MARTIN                     MARTIN
 30 SALESMAN                 TURNER                     TURNER
 30 SALESMAN                 WARD                       WARD
```

Просто меняя значения, по которым происходит группировка (следовательно, и не участвующие в агрегации элементы списка SELECT), можно создавать отчеты разных форматов. Стоит потратить немного времени и поэкспериментировать, изменяя эти значения, чтобы понять, как меняются форматы в зависимости от того, что входит в конструкцию GROUP BY.

PostgreSQL и MySQL

Подход к решению для этих СУБД аналогичен используемому для всех остальных: создание уникальных пар JOB/ENAME. Первый шаг – с помощью скалярного подзапроса снабдить каждое сочетание JOB/ENAME «порядковым номером», или «рангом»:

```
select e.job,
       e.ename,
       (select count(*) from emp d
         where e.job=d.job and e.empno < d.empno) as rnk
  from emp e
```

```
JOB       ENAME             RNK
--------- ---------- ----------
```

```
CLERK     SMITH              3
SALESMAN  ALLEN              3
SALESMAN  WARD               2
MANAGER   JONES              2
SALESMAN  MARTIN             1
MANAGER   BLAKE              1
MANAGER   CLARK              0
ANALYST   SCOTT              1
PRESIDENT KING               0
SALESMAN  TURNER             0
CLERK     ADAMS              2
CLERK     JAMES              1
ANALYST   FORD               0
CLERK     MILLER             0
```

Присвоение каждому сочетанию JOB/ENAME уникального «ранга» делает каждую строку уникальной. Даже если есть служащие, имеющие одинаковые имена и занимающие одну должность, не будет двух служащих с одним рангом для данной должности. Этот шаг является самым важным при решении задачи. Если не выполнить этот первый шаг, внешний запрос в результате агрегации удалит нужные строки. Рассмотрим, как выглядело бы результирующее множество без присвоения ранга каждому сочетанию JOB/ENAME, если бы применялась техника, представленная в первом рецепте:

```
select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
  from emp
```

```
CLERKS     ANALYSTS   MGRS       PREZ       SALES
---------- ---------- ---------- ---------- ----------
SMITH      SCOTT      JONES      KING       WARD
```

К сожалению, для каждого значения JOB возвращена только одна строка: служащий с наивысшим рангом. При разворачивании результирующего множества функции MIN или MAX должны использоваться только как средства для удаления значений NULL, без ограничения возвращаемых значений ENAME.

Теперь, когда ясен смысл назначения рангов, можно идти далее. Следующий шаг – использовать выражение CASE для распределения значений ENAME по соответствующим столбцам (JOB):

```
select rnk,
       case when job='CLERK'
```

```
            then ename else null end as clerks,
       case when job='ANALYST'
            then ename else null end as analysts,
       case when job='MANAGER'
            then ename else null end as mgrs,
       case when job='PRESIDENT'
            then ename else null end as prez,
       case when job='SALESMAN'
            then ename else null end as sales
  from (
select e.job,
       e.ename,
       (select count(*) from emp d
         where e.job=d.job and e.empno < d.empno) as rnk
  from emp e
       ) x

RNK CLERKS ANALYSTS MGRS  PREZ SALES
--- ------ -------- ----- ---- ----------
  3 SMITH
  3                            ALLEN
  2                            WARD
  2                JONES
  1                            MARTIN
  1                BLAKE
  0                CLARK
  1        SCOTT
  0                       KING
  0                            TURNER
  2 ADAMS
  1 JAMES
  0        FORD
  0 MILLER
```

На данный момент строки транспонированы в столбцы, осталось лишь удалить значения NULL, чтобы сделать результирующее множество более удобным для восприятия. Удаляем значения NULL с помощью агрегатной функции MAX и группируем результаты по RNK. (Выбор MAX произволен. Можно использовать и функцию MIN.) Сочетания значений RN/JOB/ENAME уникальны, поэтому агрегатная функция просто удалит значения NULL:

```
select max(case when job='CLERK'
                then ename else null end) as clerks,
       max(case when job='ANALYST'
                then ename else null end) as analysts,
       max(case when job='MANAGER'
                then ename else null end) as mgrs,
       max(case when job='PRESIDENT'
                then ename else null end) as prez,
       max(case when job='SALESMAN'
                then ename else null end) as sales
```

```
  from (
select e.job,
       e.ename,
       (select count(*) from emp d
         where e.job=d.job and e.empno < d.empno) as rnk
  from emp e
       ) x
 group by rnk
```

```
CLERKS ANALYSTS MGRS  PREZ SALES
------ -------- ----- ---- ------
MILLER FORD     CLARK KING TURNER
JAMES  SCOTT    BLAKE      MARTIN
ADAMS           JONES      WARD
SMITH                      ALLEN
```

Обратное разворачивание результирующего множества

Задача

Требуется преобразовать столбцы в строки. Рассмотрим следующее результирующее множество:

```
DEPTNO_10  DEPTNO_20  DEPTNO_30
---------- ---------- ----------
         3          5          6
```

Оно должно быть преобразовано к такому виду:

```
DEPTNO COUNTS_BY_DEPT
------ --------------
    10              3
    20              5
    30              6
```

Решение

Чтобы получить требуемое результирующее множество, можно просто выполнить операции COUNT и GROUP BY для таблицы EMP. Однако главное здесь – понимать, что данные не хранятся как строки; возможно, данные денормализованы и агрегированные значения хранятся как множество столбцов.

Чтобы преобразовать столбцы в строки, используйте декартово произведение. Сколько столбцов требуется преобразовать в строки, должно быть известно заранее, потому что кардинальность табличного выражения, используемого для создания декартова произведения, должна, как минимум, равняться числу транспонируемых столбцов.

Мы не будем создавать денормализованную таблицу данных. В решении данного рецепта применим решение из первого рецепта главы и создадим «широкое» результирующее множество. Вот полное решение:

```
 1  select dept.deptno,
 2         case dept.deptno
 3              when 10 then emp_cnts.deptno_10
 4              when 20 then emp_cnts.deptno_20
 5              when 30 then emp_cnts.deptno_30
 6         end as counts_by_dept
 7    from (
 8  select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
 9         sum(case when deptno=20 then 1 else 0 end) as deptno_20,
10         sum(case when deptno=30 then 1 else 0 end) as deptno_30
11    from emp
12         ) emp_cnts,
13         (select deptno from dept where deptno <= 30) dept
```

Обсуждение

Вложенный запрос EMP_CNTS является денормализованным представлением, или «широким» результирующим множеством, которое требуется преобразовать в строки. Оно показано ниже:

```
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp

DEPTNO_10  DEPTNO_20  DEPTNO_30
---------- ---------- ----------
         3          5          6
```

Поскольку здесь три столбца, будет создано три строки. Начнем с декартова произведения между вложенным запросом EMP_CNTS и некоторым табличным выражением, имеющим, по крайней мере, три строки. В следующем коде для создания декартова произведения используется таблица DEPT. В DEPT четыре строки:

```
select dept.deptno,
       emp_cnts.deptno_10,
       emp_cnts.deptno_20,
       emp_cnts.deptno_30
  from (
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
       ) emp_cnts,
       (select deptno from dept where deptno <= 30) dept

DEPTNO  DEPTNO_10  DEPTNO_20  DEPTNO_30
------ ---------- ---------- ----------
    10          3          5          6
    20          3          5          6
    30          3          5          6
```

Декартово произведение позволяет получить по строке для каждого столбца вложенного запроса EMP_CNTS. Поскольку в окончательное результирующее множество должны войти только значения DEPTNO и количество служащих в соответствующем DEPTNO, используем выражение CASE для преобразования трех столбцов в один:

```
select dept.deptno,
       case dept.deptno
            when 10 then emp_cnts.deptno_10
            when 20 then emp_cnts.deptno_20
            when 30 then emp_cnts.deptno_30
       end as counts_by_dept
  from (
select sum(case when deptno=10 then 1 else 0 end) as deptno_10,
       sum(case when deptno=20 then 1 else 0 end) as deptno_20,
       sum(case when deptno=30 then 1 else 0 end) as deptno_30
  from emp
       ) emp_cnts,
       (select deptno from dept where deptno <= 30) dept

DEPTNO COUNTS_BY_DEPT
------ --------------
    10              3
    20              5
    30              6
```

Обратное разворачивание результирующего множества в один столбец

Задача

Требуется вывести все возвращаемые запросом столбцы в одном столбце. Например, стоит задача получить ENAME, JOB и SAL всех служащих 10-го отдела (DEPTNO 10), все три значения должны быть выведены в одном столбце в трех строках для каждого служащего, и значения для разных служащих должны быть разделены пустой строкой. Ожидается получить следующее результирующее множество:

```
EMPS
----------
CLARK
MANAGER
2450

KING
PRESIDENT
5000

MILLER
CLERK
1300
```

Решение

Ключ к решению – использовать декартово произведение и возвратить по четыре строки для каждого служащего. Это позволит разместить значения столбцов в отдельных строках и разделить значения, относящиеся к разным служащим, пустыми строками.

DB2, Oracle и SQL Server

С помощью ранжирующей функции ROW_NUMBER OVER присвойте каждой строке ранг на основании значений EMPNO (1–4). Затем используйте выражение CASE для преобразования трех столбцов в один:

```
 1  select case rn
 2              when 1 then ename
 3              when 2 then job
 4              when 3 then cast(sal as char(4))
 5         end emps
 6    from (
 7  select e.ename,e.job,e.sal,
 8         row_number()over(partition by e.empno
 9                               order by e.empno) rn
10    from emp e,
11         (select *
12            from emp where job='CLERK') four_rows
13   where e.deptno=10
14         ) x
```

PostgreSQL и MySQL

Данный рецепт призван обратить внимание на применение ранжирующих функций для ранжирования строк, которое затем используется при разворачивании таблицы. На момент написания данной книги ни PostgreSQL, ни MySQL не поддерживают ранжирующие функции.

Обсуждение

DB2, Oracle и SQL Server

Первый шаг – с помощью ранжирующей функции ROW_NUMBER OVER присвоить ранг каждому служащему DEPTNO 10:

```
select e.ename,e.job,e.sal,
       row_number()over(partition by e.empno
                             order by e.empno) rn
  from emp e
 where e.deptno=10
```

```
ENAME      JOB              SAL         RN
---------- --------- ---------- ----------
CLARK      MANAGER         2450          1
KING       PRESIDENT       5000          1
MILLER     CLERK           1300          1
```

Пока что ранги ничего не значат. Сегментирование выполнялось по EMPNO, поэтому всем трем служащим DEPTNO 10 присвоен ранг 1. Как только будет введено декартово произведение, появятся разные ранги, что можно видеть в следующих результатах:

```
select e.ename,e.job,e.sal,
       row_number()over(partition by e.empno
                            order by e.empno) rn
  from emp e,
       (select *
          from emp where job='CLERK') four_rows
 where e.deptno=10
```

```
ENAME      JOB              SAL         RN
---------- --------- ---------- ----------
CLARK      MANAGER          2450          1
CLARK      MANAGER          2450          2
CLARK      MANAGER          2450          3
CLARK      MANAGER          2450          4
KING       PRESIDENT        5000          1
KING       PRESIDENT        5000          2
KING       PRESIDENT        5000          3
KING       PRESIDENT        5000          4
MILLER     CLERK            1300          1
MILLER     CLERK            1300          2
MILLER     CLERK            1300          3
MILLER     CLERK            1300          4
```

Здесь следует остановиться и понять два ключевых момента:

- RN равен 1 теперь не для всех служащих; теперь это повторяющаяся последовательность значений от 1 до 4, потому что ранжирующие функции применяются после выполнения операторов FROM и WHERE. Таким образом, сегментирование по EMPNO обусловливает сброс RN в начальное значение (1) для записи нового служащего.
- Вложенный запрос FOUR_ROWS – это просто SQL-выражение, возвращающее четыре строки. Это все, что оно делает. Должно быть получено по строке для каждого столбца (ENAME, JOB, SAL) плюс строка-пробел.

На данный момент вся тяжелая работа выполнена, осталось лишь с помощью выражения CASE разместить знеачения ENAME, JOB и SAL всех служащих в один столбец (чтобы значения SAL могли использоваться в CASE, их необходимо привести к строковому типу):

```
select case rn
            when 1 then ename
            when 2 then job
            when 3 then cast(sal as char(4))
       end emps
  from (
select e.ename,e.job,e.sal,
```

```
      row_number()over(partition by e.empno
                           order by e.empno) rn
  from emp e,
       (select *
          from emp where job='CLERK') four_rows
 where e.deptno=10
       ) x
```

```
EMPS
----------
CLARK
MANAGER
2450

KING
PRESIDENT
5000

MILLER
CLERK
1300
```

Исключение повторяющихся значений из результирующего множества

Задача

При формировании отчета выдвинуто требование о том, что дублирующиеся значения в столбце должны отображаться лишь один раз. Например, из таблицы EMP требуется извлечь значения DEPTNO и ENAME, при этом необходимо сгруппировать вместе все строки для каждого значения DEPTNO и выводить каждое значение DEPTNO только один раз. Ожидается получить следующее результирующее множество:

```
DEPTNO ENAME
------ ---------
    10 CLARK
       KING
       MILLER
    20 SMITH
       ADAMS
       FORD
       SCOTT
       JONES
    30 ALLEN
       BLAKE
       MARTIN
       JAMES
       TURNER
       WARD
```

Решение

Это простая задача по форматированию, которая без труда решается применением оконной функции LAG OVER, предоставляемой Oracle. Можно прибегнуть к другим средствам, например, скалярным подзапросам и другим оконным функциям (именно они будут использоваться для остальных платформ), но LAG OVER наиболее удобна и уместна в данном случае.

DB2 и SQL Server

С помощью оконной функции MIN OVER можно найти наименьшее значение EMPNO для каждого DEPTNO, затем, используя выражение CASE, «стереть» значение DEPTNO из строк со всеми остальными EMPNO:

```
 1  select case when empno=min_empno
 2              then deptno else null
 3         end deptno,
 4         ename
 5    from (
 6  select deptno,
 7         min(empno)over(partition by deptno) min_empno,
 8         empno,
 9         ename
10    from emp
11         ) x
```

Oracle

С помощью оконной функции LAG OVER организуйте доступ к предыдущим относительно текущей строкам, чтобы найти первое значение DEPTNO для каждого сегмента:

```
1 select to_number(
2           decode(lag(deptno)over(order by deptno),
3                  deptno,null,deptno)
4        ) deptno, ename
5   from emp
```

PostgreSQL и MySQL

Данный рецепт описывает применение оконных функций для упрощения доступа к строкам, окружающим текущую. На момент написания данной книги эти производители не поддерживают оконные функции.

Обсуждение

DB2 и SQL Server

Первый шаг – с помощью оконной функции MIN OVER найти наименьшее значение EMPNO для каждого DEPTNO:

```
select deptno,
       min(empno)over(partition by deptno) min_empno,
```

```
       empno,
       ename
  from emp

DEPTNO  MIN_EMPNO      EMPNO ENAME
------ ---------- ---------- ----------
    10       7782       7782 CLARK
    10       7782       7839 KING
    10       7782       7934 MILLER
    20       7369       7369 SMITH
    20       7369       7876 ADAMS
    20       7369       7902 FORD
    20       7369       7788 SCOTT
    20       7369       7566 JONES
    30       7499       7499 ALLEN
    30       7499       7698 BLAKE
    30       7499       7654 MARTIN
    30       7499       7900 JAMES
    30       7499       7844 TURNER
    30       7499       7521 WARD
```

Следующий и последний шаг – посредством выражения CASE обеспечить однократное отображение DEPTNO. Если значение EMPNO служащего соответствует MIN_EMPNO, возвращается DEPTNO; в противном случае возвращается NULL:

```
select case when empno=min_empno
            then deptno else null
       end deptno,
       ename
  from (
select deptno,
       min(empno)over(partition by deptno) min_empno,
       empno,
       ename
  from emp
       ) x

DEPTNO ENAME
------ ----------
    10 CLARK
       KING
       MILLER
    20 SMITH
       ADAMS
       FORD
       SCOTT
       JONES
    30 ALLEN
       BLAKE
       MARTIN
       JAMES
```

```
       TURNER
       WARD
```

Oracle

Первый шаг – с помощью оконной функции LAG OVER возвратить предыдущее значение DEPTNO для каждой строки:

```
select lag(deptno)over(order by deptno) lag_deptno,
       deptno,
       ename
  from emp

LAG_DEPTNO     DEPTNO ENAME
---------- ---------- ----------
                   10 CLARK
        10         10 KING
        10         10 MILLER
        10         20 SMITH
        20         20 ADAMS
        20         20 FORD
        20         20 SCOTT
        20         20 JONES
        20         30 ALLEN
        30         30 BLAKE
        30         30 MARTIN
        30         30 JAMES
        30         30 TURNER
        30         30 WARD
```

Взглянув на представленное выше результирующее множество, можно сразу увидеть, где значения DEPTNO и LAG_ DEPTNO совпадают. Для этих строк DEPTNO должно быть присвоено значение NULL. Делается это с помощью функции DECODE (функция TO_NUMBER включена, чтобы привести значение DEPTNO к числовому типу):

```
select to_number(
           decode(lag(deptno)over(order by deptno),
                  deptno,null,deptno)
       ) deptno, ename
  from emp

DEPTNO ENAME
------ ----------
    10 CLARK
       KING
       MILLER
    20 SMITH
       ADAMS
       FORD
       SCOTT
       JONES
    30 ALLEN
```

```
BLAKE
MARTIN
JAMES
TURNER
WARD
```

Разворачивание результирующего множества для упрощения вычислений

Задача

Требуется выполнить вычисления, в которых участвуют данные нескольких строк. Чтобы упростить задачу, эти строки надо развернуть и превратить в столбцы, так чтобы все необходимые значения располагались в одной строке.

В данных, используемых в этой книге для примера, DEPTNO 20 – отдел с самой высокой совокупной заработной платой, в чем можно убедиться, выполнив следующий запрос:

```
select deptno, sum(sal) as sal
  from emp
 group by deptno

DEPTNO        SAL
------ ----------
    10       8750
    20      10875
    30       9400
```

Надо вычислить разность между заработными платами DEPTNO 20 и DEPTNO 10 и заработными платами DEPTNO 20 и DEPTNO 30.

Решение

С помощью агрегатной функции SUM и выражения CASE транспонируйте результаты. Затем включите свои выражения в список оператора SELECT:

```
1  select d20_sal - d10_sal as d20_10_diff,
2         d20_sal - d30_sal as d20_30_diff
3    from (
4  select sum(case when deptno=10 then sal end) as d10_sal,
5         sum(case when deptno=20 then sal end) as d20_sal,
6         sum(case when deptno=30 then sal end) as d30_sal
7    from emp
8         ) totals_by_dept
```

Обсуждение

Первый шаг – посредством выражений CASE развернуть таблицу и перенести заработные платы для каждого DEPTNO из строк в столбцы:

```
select case when deptno=10 then sal end as d10_sal,
       case when deptno=20 then sal end as d20_sal,
       case when deptno=30 then sal end as d30_sal
  from emp

D10_SAL    D20_SAL    D30_SAL
------- ---------- ----------
                800
                          1600
                          1250
               2975
                          1250
                          2850
   2450
               3000
   5000
                          1500
               1100
                           950
               3000
   1300
```

Следующий шаг – суммировать все заработные платы для каждого DEPTNO, применяя агрегатную функцию SUM к каждому выражению CASE:

```
select sum(case when deptno=10 then sal end) as d10_sal,
       sum(case when deptno=20 then sal end) as d20_sal,
       sum(case when deptno=30 then sal end) as d30_sal
  from emp

D10_SAL    D20_SAL    D30_SAL
------- ---------- ----------
   8750      10875       9400
```

Заключительный шаг – просто оформить приведенный выше SQL как вложенный запрос и найти разности.

Создание блоков данных фиксированного размера

Задача

Требуется организовать данные в одинаковые по размеру блоки с предопределенным количеством элементов в каждом блоке. Общее число блоков может быть неизвестно, но каждый из них должен гарантированно содержать пять элементов. Например, необходимо организовать служащих из таблицы EMP в группы по пять на основании значения EMPNO, как показано ниже:

```
GRP      EMPNO ENAME
--- ---------- -------
  1       7369 SMITH
  1       7499 ALLEN
```

```
1       7521 WARD
1       7566 JONES
1       7654 MARTIN
2       7698 BLAKE
2       7782 CLARK
2       7788 SCOTT
2       7839 KING
2       7844 TURNER
3       7876 ADAMS
3       7900 JAMES
3       7902 FORD
3       7934 MILLER
```

Решение

Решение данной задачи существенно упрощается, если СУБД обеспечивает функции для ранжирования строк. Когда строки ранжированы, для создания блоков по пять строк остается только выполнить деление и определить верхнюю границу для частного.

DB2, Oracle и SQL Server

Используйте ранжирующую функцию ROW_NUMBER OVER, чтобы ранжировать служащих по EMPNO. Затем, чтобы создать группы, разделите полученные ранги на 5 (для SQL Server будет использоваться не функция CEIL, а функция CEILING):

```
1 select ceil(row_number()over(order by empno)/5.0) grp,
2        empno,
3        ename
4   from emp
```

PostgreSQL и MySQL

С помощью скалярного подзапроса ранжируйте строки по EMPNO. Затем разделите полученные ранги на 5, чтобы создать группы:

```
1  select ceil(rnk/5.0) as grp,
2         empno, ename
3    from (
4  select e.empno, e.ename,
5         (select count(*) from emp d
6           where e.empno < d.empno)+1 as rnk
7    from emp e
8         ) x
9   order by grp
```

Обсуждение

DB2, Oracle и SQL Server

Ранжирующая функция ROW_NUMBER OVER присваивает ранги или «порядковые номера» строкам, сортированным по столбцу EMPNO:

```
select row_number()over(order by empno) rn,
       empno,
       ename
  from emp

RN      EMPNO ENAME
-- ---------- ----------
 1       7369 SMITH
 2       7499 ALLEN
 3       7521 WARD
 4       7566 JONES
 5       7654 MARTIN
 6       7698 BLAKE
 7       7782 CLARK
 8       7788 SCOTT
 9       7839 KING
10       7844 TURNER
11       7876 ADAMS
12       7900 JAMES
13       7902 FORD
14       7934 MILLER
```

Следующий шаг – применить функцию CEIL (или CEILING) после деления результата ROW_NUMBER OVER на пять. Деление на пять логически организует строки в группы по пять, т. е. пять значений, которые меньше или равны 1; пять значений, которые больше 1, но меньше или равны 2; оставшаяся группа (состоящая из четырех последних строк, поскольку 14, количество строк в таблице EMP, не кратно 5) соответствует значениям, которые больше 2, но меньше или равны 3.

Функция CEIL возвращает наименьшее целое число, которое больше, чем переданное в нее значение; это обеспечит создание групп целых чисел. Результаты деления и применения CEIL представлены ниже. Можно проследить порядок операций слева направо, от RN до DIVISION и GRP:

```
select row_number()over(order by empno) rn,
       row_number()over(order by empno)/5.0 division,
       ceil(row_number()over(order by empno)/5.0) grp,
       empno,
       ename
  from emp

RN   DIVISION GRP EMPNO ENAME
-- ---------- --- ----- ----------
 1         .2   1  7369 SMITH
 2         .4   1  7499 ALLEN
 3         .6   1  7521 WARD
 4         .8   1  7566 JONES
 5          1   1  7654 MARTIN
 6        1.2   2  7698 BLAKE
 7        1.4   2  7782 CLARK
```

```
 8        1.6   2  7788 SCOTT
 9        1.8   2  7839 KING
10          2   2  7844 TURNER
11        2.2   3  7876 ADAMS
12        2.4   3  7900 JAMES
13        2.6   3  7902 FORD
14        2.8   3  7934 MILLER
```

PostgreSQL и MySQL

Первый шаг – использовать скалярный подзапрос, чтобы ранжировать строки по EMPNO:

```
select (select count(*) from emp d
          where e.empno < d.empno)+1 as rnk,
       e.empno, e.ename
  from emp e
 order by 1
```

```
RNK      EMPNO ENAME
--- ---------- ----------
  1       7934 MILLER
  2       7902 FORD
  3       7900 JAMES
  4       7876 ADAMS
  5       7844 TURNER
  6       7839 KING
  7       7788 SCOTT
  8       7782 CLARK
  9       7698 BLAKE
 10       7654 MARTIN
 11       7566 JONES
 12       7521 WARD
 13       7499 ALLEN
 14       7369 SMITH
```

Следующий шаг – после деления RNK на 5 применить функцию CEIL. Деление на 5 логически организует строки в группы по пять, т. е. пять значений, меньших или равных 1; пять значений, больших 1, но меньших или равных 2; последней группе (состоящей из последних четырех строк, поскольку 14, количество строк в таблице EMP, не кратно 5) соответствуют значения, которые больше 2, но меньше или равны 3. Результаты деления и применения CEIL показаны ниже. Проследить порядок операций можно слева направо, от RNK до GRP:

```
select rnk,
       rnk/5.0 as division,
       ceil(rnk/5.0) as grp,
       empno, ename
  from (
select e.empno, e.ename,
       (select count(*) from emp d
         where e.empno < d.empno)+1 as rnk
```

```
  from emp e
       ) x
 order by 1

RNK   DIVISION GRP EMPNO ENAME
--- ---------- --- ----- -------
  1         .2   1  7934 MILLER
  2         .4   1  7902 FORD
  3         .6   1  7900 JAMES
  4         .8   1  7876 ADAMS
  5          1   1  7844 TURNER
  6        1.2   2  7839 KING
  7        1.4   2  7788 SCOTT
  8        1.6   2  7782 CLARK
  9        1.8   2  7698 BLAKE
 10          2   2  7654 MARTIN
 11        2.2   3  7566 JONES
 12        2.4   3  7521 WARD
 13        2.6   3  7499 ALLEN
 14        2.8   3  7369 SMITH
```

Создание заданного количества блоков

Задача

Требуется организовать данные в определенное число блоков. Например, записи служащих в таблице EMP должны быть разделены на четыре группы. Ниже представлено предполагаемое результирующее множество:

```
GRP EMPNO ENAME
--- ----- ---------
  1  7369 SMITH
  1  7499 ALLEN
  1  7521 WARD
  1  7566 JONES
  2  7654 MARTIN
  2  7698 BLAKE
  2  7782 CLARK
  2  7788 SCOTT
  3  7839 KING
  3  7844 TURNER
  3  7876 ADAMS
  4  7900 JAMES
  4  7902 FORD
  4  7934 MILLER
```

Эта задача обратная предыдущей, где число блоков было неизвестно, но было задано количество элементов в каждом из них. Особенность данного рецепта в том, что мы можем не знать, сколько элементов в каждом блоке, но количество блоков определено заранее.

Решение

Решение этой задачи не составляет труда, если используемая СУБД обеспечивает функции для создания «блоков» строк. Если СУБД не предоставляет таких функций, можно просто ранжировать строки и затем распределить их по блокам соответственно остатку от деления их ранга на n, где n – количество блоков, которое должно быть создано. Если доступна оконная функция NTILE, она будет использоваться для создания заданного числа блоков. NTILE разбивает упорядоченное множество на требуемое число сегментов. При этом, если количество записей не делится на это число нацело, записи «остатка» распределяются в доступные блоки, начиная с первого. Это видно из результирующего множества, которое требуется получить в данном рецепте: блоки 1 и 2 включают по 4 строки, блоки 3 и 4 – по три. Если СУБД не поддерживает NTILE, не надо беспокоиться о том, в какие блоки попадут те или иные строки. Основная цель данного рецепта – создание заданного количества блоков.

DB2

С помощью ранжирующей функции ROW_NUMBER OVER ранжируйте строки по EMPNO, затем соответственно остаткам от деления рангов на 4 организуйте четыре блока:

```
1  select mod(row_number()over(order by empno),4)+1 grp,
2         empno,
3         ename
4    from emp
5   order by 1
```

Oracle и SQL Server

Для этих баз данных подойдет решение для DB2, но в качестве альтернативы (и проще) для создания четырех блоков можно использовать оконную функцию NTILE:

```
1  select ntile(4)over(order by empno) grp,
2         empno,
3         ename
4    from emp
```

MySQL и PostgreSQL

Используя рефлексивное объединение, ранжируйте строки по EMPNO, затем соответственно остаткам от деления рангов на 4 организуйте необходимые блоки:

```
1  select mod(count(*),4)+1 as grp,
2         e.empno,
3         e.ename
4    from emp e, emp d
5   where e.empno >= d.empno
```

```
6   group by e.empno,e.ename
7   order by 1
```

Обсуждение

DB2

Первый шаг – с помощью ранжирующей функции ROW_NUMBER OVER ранжируем все строки по EMPNO:

```
select row_number()over(order by empno) grp,
       empno,
       ename
  from emp
```

```
GRP EMPNO ENAME
--- ----- ------
  1  7369 SMITH
  2  7499 ALLEN
  3  7521 WARD
  4  7566 JONES
  5  7654 MARTIN
  6  7698 BLAKE
  7  7782 CLARK
  8  7788 SCOTT
  9  7839 KING
 10  7844 TURNER
 11  7876 ADAMS
 12  7900 JAMES
 13  7902 FORD
 14  7934 MILLER
```

Когда все строки получили соответствующий ранг, создаем четыре блока с помощью функции вычисления остатка от деления MOD:

```
select mod(row_number()over(order by empno),4) grp,
       empno,
       ename
  from emp
```

```
GRP EMPNO ENAME
--- ----- ------
  1  7369 SMITH
  2  7499 ALLEN
  3  7521 WARD
  0  7566 JONES
  1  7654 MARTIN
  2  7698 BLAKE
  3  7782 CLARK
  0  7788 SCOTT
  1  7839 KING
  2  7844 TURNER
  3  7876 ADAMS
  0  7900 JAMES
```

```
1  7902 FORD
2  7934 MILLER
```

Последний шаг – добавить единицу к GRP, чтобы нумерация блоков начиналась не с 0, а с 1, и применить ORDER BY по GRP, чтобы сортировать строки по блокам.

Oracle и SQL Server

Всю работу выполняет функция NTILE. Просто передаем в нее число, представляющее требуемое количество блоков, и чудо происходит прямо на наших глазах.

MySQL и PostgreSQL

Первый шаг – сформировать декартово произведение таблицы EMP, так чтобы все значения EMPNO можно было сравнивать между собой (ниже показан лишь фрагмент декартова произведения, потому что оно включает 196 строк (14×14):

```
select e.empno,
       e.ename,
       d.empno,
       d.ename
  from emp e, emp d

EMPNO ENAME      EMPNO ENAME
----- ---------- ----------
 7369 SMITH       7369 SMITH
 7369 SMITH       7499 ALLEN
 7369 SMITH       7521 WARD
 7369 SMITH       7566 JONES
 7369 SMITH       7654 MARTIN
 7369 SMITH       7698 BLAKE
 7369 SMITH       7782 CLARK
 7369 SMITH       7788 SCOTT
 7369 SMITH       7839 KING
 7369 SMITH       7844 TURNER
 7369 SMITH       7876 ADAMS
 7369 SMITH       7900 JAMES
 7369 SMITH       7902 FORD
 7369 SMITH       7934 MILLER
 ...
```

Как видно из приведенного результирующего множества, значение EMPNO служащего SMITH можно сравнить с EMPNO всех остальных служащих таблицы EMP (все EMPNO можно сравнить между собой). Следующий шаг – ограничить декартово произведение только теми значениями EMPNO, которые больше или равны другому EMPNO. Результирующее множество частично (поскольку в нем 105 строк) показано ниже:

```
select e.empno,
       e.ename,
```

```
       d.empno,
       d.ename
  from emp e, emp d
 where e.empno >= d.empno

EMPNO ENAME           EMPNO ENAME
----- ---------- ---------- ----------
 7934 MILLER           7934 MILLER
 7934 MILLER           7902 FORD
 7934 MILLER           7900 JAMES
 7934 MILLER           7876 ADAMS
 7934 MILLER           7844 TURNER
 7934 MILLER           7839 KING
 7934 MILLER           7788 SCOTT
 7934 MILLER           7782 CLARK
 7934 MILLER           7698 BLAKE
 7934 MILLER           7654 MARTIN
 7934 MILLER           7566 JONES
 7934 MILLER           7521 WARD
 7934 MILLER           7499 ALLEN
 7934 MILLER           7369 SMITH
 ...
 7499 ALLEN            7499 ALLEN
 7499 ALLEN            7369 SMITH
 7369 SMITH            7369 SMITH
```

Чтобы показать, как предикат WHERE ограничил декартово произведение, из всего результирующего множества я выбрал только строки (из EMP E) для служащих MILLER, ALLEN и SMITH. Поскольку предикат WHERE, используемый для отсеивания по EMPNO, соответствует условию «больше чем или равно», мы гарантированно получим, по крайней мере, одну строку для каждого служащего, потому что каждое значение EMPNO равно самому себе. Но почему для служащего SMITH (в левой части результирующего множества) получена всего одна строка, тогда как для ALLEN их две и для MILLER их 14? Причина в процедуре сравнения значений EMPNO в предикате WHERE: выбираются значения «больше чем или равные» рассматриваемому. В случае со SMITH нет такого значения EMPNO, которое было бы меньше 7369, поэтому для SMITH возвращается только одна строка. В случае с ALLEN значение EMPNO служащего ALLEN, очевидно, равно самому себе (поэтому возвращена соответствующая строка), но 7499 к тому же больше 7369 (EMPNO служащего SMITH), поэтому для ALLEN возвращено две строки. Значение EMPNO служащего MILLER больше, чем EMPNO всех остальных служащих таблицы EMP (и, конечно, равно самому себе), поэтому для MILLER получаем 14 строк.

Теперь мы можем сравнить все EMPNO и выбрать строки, в которых одно значение больше другого. Используем агрегатную функцию COUNT, чтобы получить рефлексивное объединение, поскольку оно является наиболее выразительным результирующим множеством:

```
select count(*) as grp,
       e.empno,
       e.ename
  from emp e, emp d
 where e.empno >= d.empno
 group by e.empno,e.ename
 order by 1
```

```
GRP      EMPNO ENAME
--- ---------- ----------
  1       7369 SMITH
  2       7499 ALLEN
  3       7521 WARD
  4       7566 JONES
  5       7654 MARTIN
  6       7698 BLAKE
  7       7782 CLARK
  8       7788 SCOTT
  9       7839 KING
 10       7844 TURNER
 11       7876 ADAMS
 12       7900 JAMES
 13       7902 FORD
 14       7934 MILLER
```

Итак, строки ранжированы. Теперь, чтобы создать четыре блока, просто добавляем 1 к остатку от деления GRP на 4 (это обеспечит нумерацию сегментов не с 0, а с 1). С помощью оператора ORDER BY упорядочиваем блоки по GRP:

```
select mod(count(*),4)+1 as grp,
       e.empno,
       e.ename
  from emp e, emp d
 where e.empno >= d.empno
 group by e.empno,e.ename
 order by 1
```

```
GRP      EMPNO ENAME
--- ---------- ---------
  1       7900 JAMES
  1       7566 JONES
  1       7788 SCOTT
  2       7369 SMITH
  2       7902 FORD
  2       7654 MARTIN
  2       7839 KING
  3       7499 ALLEN
  3       7698 BLAKE
  3       7934 MILLER
  3       7844 TURNER
  4       7521 WARD
```

```
4       7782 CLARK
4       7876 ADAMS
```

Создание горизонтальных гистограмм

Задача

Требуется с помощью SQL создать горизонтальные гистограммы. Например, поставлена задача отобразить количество служащих в каждом отделе в виде горизонтальной гистограммы, в которой каждый служащий представлен экземпляром символа «*». Должно быть получено следующее результирующее множество:

```
DEPTNO CNT
------ ----------
    10 ***
    20 *****
    30 ******
```

Решение

Ключ к решению – с помощью агрегатной функции COUNT и группировки по DEPTNO найти количество служащих в каждом отделе. После этого передать значения, возвращенные COUNT, в строковую функцию, которая формирует ряды символов «*».

DB2

Для формирования гистограммы используйте функцию REPEAT (повторить):

```
1 select deptno,
2        repeat('*',count(*)) cnt
3   from emp
4  group by deptno
```

Oracle, PostgreSQL и MySQL

Для формирования необходимых строк символов «*» используйте функцию LPAD:

```
1 select deptno,
2        lpad('*',count(*),'*') as cnt
3   from emp
4  group by deptno
```

SQL Server

Гистограмма формируется с помощью функции REPLICATE:

```
1 select deptno,
2        replicate('*',count(*)) cnt
3   from emp
4  group by deptno
```

Обсуждение

Техника для всех баз данных одинакова. Единственное отличие состоит в строковых функциях, используемых для получения рядов символов «*». В данном обсуждении будем опираться на решение для Oracle, но объяснение правомочно для всех решений.

Первый шаг – подсчитываем количество служащих в каждом отделе:

```
select deptno,
       count(*)
  from emp
 group by deptno

DEPTNO   COUNT(*)
------ ----------
    10          3
    20          5
    30          6
```

Следующий шаг – возвращаем для каждого отдела соответствующее число символов «*», исходя из значения, возвращенного COUNT(*). Для этого просто передаем COUNT(*) как аргумент в строковую функцию LPAD:

```
select deptno,
       lpad('*',count(*),'*') as cnt
  from emp
 group by deptno

DEPTNO CNT
------ ----------
    10 ***
    20 *****
    30 ******
```

Пользователям PostgreSQL придется явно привести значение, возвращенное COUNT(*), к целому типу, как показано ниже:

```
select deptno,
       lpad('*',count(*)::integer,'*') as cnt
  from emp
 group by deptno

DEPTNO CNT
------ ----------
    10 ***
    20 *****
    30 ******
```

Приведение необходимо, потому что в PostgreSQL числовой аргумент LPAD обязательно должен быть целым числом.

Создание вертикальных гистограмм

Задача

Требуется создать гистограмму, в которой значения увеличиваются вдоль вертикальной оси снизу вверх. Например, поставлена задача отобразить количество служащих в каждом отделе в виде вертикальной гистограммы, в которой каждый служащий представлен экземпляром символа «*». Должно быть получено следующее результирующее множество:

```
D10 D20 D30
--- --- ---
        *
    *   *
    *   *
*   *   *
*   *   *
*   *   *
```

Решение

Техника, используемая для решения этой задачи, основана на втором рецепте данной главы, «Разворачивание результирующего множества в несколько строк».

DB2, Oracle и SQL Server

Используйте функцию ROW_NUMBER OVER, чтобы уникально идентифицировать каждый экземпляр «*» для каждого DEPTNO. С помощью агрегатной функции MAX разверните результирующее множество и сгруппируйте его по значениям, возвращенным ROW_NUMBER OVER (пользователи SQL Server не должны применять DESC в операторе ORDER BY):

```
 1  select max(deptno_10) d10,
 2         max(deptno_20) d20,
 3         max(deptno_30) d30
 4    from (
 5  select row_number()over(partition by deptno order by empno) rn,
 6         case when deptno=10 then '*' else null end deptno_10,
 7         case when deptno=20 then '*' else null end deptno_20,
 8         case when deptno=30 then '*' else null end deptno_30
 9    from emp
10         ) x
11   group by rn
12   order by 1 desc, 2 desc, 3 desc
```

PostgreSQL и MySQL

Используйте скалярный подзапрос, чтобы уникально идентифицировать каждый экземпляр «*» для каждого DEPTNO. Чтобы развернуть

результирующее множество, примените агрегатную функцию MAX к значениям, возвращенным вложенным запросом X, проводя при этом группировку по RNK. Пользователи MySQL не должны использовать DESC в операторе ORDER BY:

```
1  select max(deptno_10) as d10,
 2         max(deptno_20) as d20,
 3         max(deptno_30) as d30
 4    from (
 5  select case when e.deptno=10 then '*' else null end deptno_10,
 6         case when e.deptno=20 then '*' else null end deptno_20,
 7         case when e.deptno=30 then '*' else null end deptno_30,
 8         (select count(*) from emp d
 9           where e.deptno=d.deptno and e.empno < d.empno ) as rnk
10    from emp e
11         ) x
12   group by rnk
13   order by 1 desc, 2 desc, 3 desc
```

Обсуждение

DB2, Oracle и SQL Server

Первый шаг – с помощью ранжирующей функции ROW_NUMBER уникально идентифицировать каждый экземпляр «*» для каждого отдела. Возвращаем «*» для каждого служащего каждого отдела посредством выражения CASE:

```
select row_number()over(partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
  from emp

RN DEPTNO_10  DEPTNO_20  DEPTNO_30
-- ---------- ---------- ---------
 1 *
 2 *
 3 *
 1            *
 2            *
 3            *
 4            *
 5            *
 1                       *
 2                       *
 3                       *
 4                       *
 5                       *
 6                       *
```

Следующий и последний шаг – применить агрегатную функцию MAX к каждому выражению CASE, группируя по RN, чтобы удалить значе-

ния NULL из результирующего множества. Упорядочиваем результаты по возрастанию или по убыванию, в зависимости от того, как используемая СУБД сортирует значения NULL:

```
select max(deptno_10) d10,
       max(deptno_20) d20,
       max(deptno_30) d30
  from (
select row_number()over(partition by deptno order by empno) rn,
       case when deptno=10 then '*' else null end deptno_10,
       case when deptno=20 then '*' else null end deptno_20,
       case when deptno=30 then '*' else null end deptno_30
  from emp
       ) x
 group by rn
 order by 1 desc, 2 desc, 3 desc
```

```
D10 D20 D30
--- --- ---
        *
    *   *
    *   *
*   *   *
*   *   *
*   *   *
```

PostgreSQL и MySQL

Первый шаг – с помощью скалярного подзапроса уникально идентифицировать каждый экземпляр «*» каждого отдела. Скалярный подзапрос ранжирует служащих каждого отдела по EMPNO, поэтому дубликатов быть не может. Возвращаем «*» для каждого служащего каждого отдела посредством выражения CASE:

```
select case when e.deptno=10 then '*' else null end deptno_10,
       case when e.deptno=20 then '*' else null end deptno_20,
       case when e.deptno=30 then '*' else null end deptno_30,
       (select count(*) from emp d
         where e.deptno=d.deptno and e.empno < d.empno ) as rnk
  from emp e
```

```
DEPTNO_10  DEPTNO_20  DEPTNO_30         RNK
---------- ---------- ---------- ----------
           *                             4
                      *                  5
                      *                  4
           *                             3
                      *                  3
                      *                  2
*                                        2
           *                             2
*                                        1
                      *                  1
           *                             1
```

```
                    *                0
              *                      0
  *                                  0
```

Затем применяем к каждому выражению CASE агрегатную функцию MAX, группируя по RNK, чтобы удалить из результирующего множества все значения NULL. Упорядочиваем результаты по возрастанию или по убыванию, в зависимости от того, как используемая СУБД сортирует значения NULL.

```
select max(deptno_10) as d10,
       max(deptno_20) as d20,
       max(deptno_30) as d30
  from (
select case when e.deptno=10 then '*' else null end deptno_10,
       case when e.deptno=20 then '*' else null end deptno_20,
       case when e.deptno=30 then '*' else null end deptno_30,
       (select count(*) from emp d
         where e.deptno=d.deptno and e.empno < d.empno ) as rnk
  from emp e
       ) x
 group by rnk
 order by 1 desc, 2 desc, 3 desc
```

```
D10 D20 D30
--- --- ---
        *
    *   *
    *   *
*   *   *
*   *   *
*   *   *
```

Как возвратить столбцы, не перечисленные в операторе GROUP BY

Задача

Выполняется запрос с оператором GROUP BY, и в результирующем множестве также требуется возвратить столбцы, не входящие в конструкцию GROUP BY. Обычно это невозможно, поскольку такие столбцы для каждой строки предлагают множество решений, тогда как требуется представить одно значение на строку.

Скажем, неоходимо выбрать служащих, получающих наибольшую и наименьшую заработные платы в каждом отделе и для каждой должности. Должны быть представлены имя каждого такого служащего, отдел, в котором он работает, его должность и заработная плата. Ожидается получить следующее результирующее множество:

```
DEPTNO ENAME  JOB         SAL DEPT_STATUS     JOB_STATUS
------ ------ --------- ----- --------------- --------------
```

```
10 MILLER CLERK      1300 LOW SAL IN DEPT TOP SAL IN JOB
10 CLARK  MANAGER    2450                 LOW SAL IN JOB
10 KING   PRESIDENT  5000 TOP SAL IN DEPT TOP SAL IN JOB
20 SCOTT  ANALYST    3000 TOP SAL IN DEPT TOP SAL IN JOB
20 FORD   ANALYST    3000 TOP SAL IN DEPT TOP SAL IN JOB
20 SMITH  CLERK       800 LOW SAL IN DEPT LOW SAL IN JOB
20 JONES  MANAGER    2975                 TOP SAL IN JOB
30 JAMES  CLERK       950 LOW SAL IN DEPT
30 MARTIN SALESMAN   1250                 LOW SAL IN JOB
30 WARD   SALESMAN   1250                 LOW SAL IN JOB
30 ALLEN  SALESMAN   1600                 TOP SAL IN JOB
30 BLAKE  MANAGER    2850 TOP SAL IN DEPT
```

К сожалению, включение всех этих столбцов в оператор SELECT разрушит группировку. Рассмотрим такой пример. Служащий «KING» получает самую высокую заработную плату. Мы хотим убедиться в этом с помощью следующего запроса:

```
select ename,max(sal)
  from emp
 group by ename
```

Вместо того чтобы представить запись «KING» и его заработную плату, приведенный выше запрос возвратит все 14 строк таблицы EMP. Причина в группировке: MAX(SAL) применяется к каждому ENAME. Таким образом, кажется, что приведенный выше запрос можно интерпретировать как «найти служащего, получающего наивысшую заработную плату», а на самом деле он «находит наивысшую заработную плату для каждого значения ENAME в таблице EMP». В данном рецепте рассматривается, как можно включить в результирующее множество столбец ENAME, не указывая его в операторе GROUP BY.

Решение

Наибольшую и наименьшую заработные платы по DEPTNO и JOB находим с помощью вложенного запроса. Затем выбираем только тех служащих, которые получают такие зарплаты.

DB2, Oracle и SQL Server

С помощью оконных функций MAX OVER и MIN OVER найдите наибольшую и наименьшую заработные платы по DEPTNO и JOB. Затем выберите строки, в которых заработные платы соответствуют полученным наибольшим и наименьшим значениям:

```
1  select deptno,ename,job,sal,
2         case when sal = max_by_dept
3              then 'TOP SAL IN DEPT'
4              when sal = min_by_dept
5              then 'LOW SAL IN DEPT'
6         end dept_status,
7         case when sal = max_by_job
8              then 'TOP SAL IN JOB'
```

```
 9              when sal = min_by_job
10              then 'LOW SAL IN JOB'
11         end job_status
12    from (
13  select deptno,ename,job,sal,
14         max(sal)over(partition by deptno) max_by_dept,
15         max(sal)over(partition by job)    max_by_job,
16         min(sal)over(partition by deptno) min_by_dept,
17         min(sal)over(partition by job)    min_by_job
18    from emp
19         ) emp_sals
20   where sal in (max_by_dept,max_by_job,
21                 min_by_dept,min_by_job)
```

PostgreSQL и MySQL

С помощью скалярных подзапросов найдите наибольшую и наименьшую заработные платы по DEPTNO и JOB. Затем выберите строки только тех служащих, которые получают такие заработные платы:

```
 1  select deptno,ename,job,sal,
 2         case when sal = max_by_dept
 3              then 'TOP SAL IN DEPT'
 4              when sal = min_by_dept
 5              then 'LOW SAL IN DEPT'
 6         end as dept_status,
 7         case when sal = max_by_job
 8              then 'TOP SAL IN JOB'
 9              when sal = min_by_job
10              then 'LOW SAL IN JOB'
11         end as job_status
12    from (
13  select e.deptno,e.ename,e.job,e.sal,
14         (select max(sal) from  emp d
15           where d.deptno = e.deptno) as max_by_dept,
16         (select max(sal) from  emp d
17           where d.job = e.job) as max_by_job,
18         (select min(sal) from  emp d
19           where d.deptno = e.deptno) as min_by_dept,
20         (select min(sal) from  emp d
21           where d.job = e.job) as min_by_job
22    from emp e
23         ) x
24   where sal in (max_by_dept,max_by_job,
25                 min_by_dept,min_by_job)
```

Обсуждение

DB2, Oracle и SQL Server

Первый шаг – с помощью оконных функций MAX OVER и MIN OVER находим наибольшие и наименьшие заработные платы по DEPTNO и JOB.

```
select deptno,ename,job,sal,
       max(sal)over(partition by deptno) maxDEPT,
       max(sal)over(partition by job)    maxJOB,
       min(sal)over(partition by deptno) minDEPT,
       min(sal)over(partition by job)    minJOB
  from emp
```

```
DEPTNO ENAME  JOB         SAL MAXDEPT MAXJOB MINDEPT MINJOB
------ ------ --------- ----- ------- ------ ------- ------
    10 MILLER CLERK      1300    5000   1300    1300    800
    10 CLARK  MANAGER    2450    5000   2975    1300   2450
    10 KING   PRESIDENT  5000    5000   5000    1300   5000
    20 SCOTT  ANALYST    3000    3000   3000     800   3000
    20 FORD   ANALYST    3000    3000   3000     800   3000
    20 SMITH  CLERK       800    3000   1300     800    800
    20 JONES  MANAGER    2975    3000   2975     800   2450
    20 ADAMS  CLERK      1100    3000   1300     800    800
    30 JAMES  CLERK       950    2850   1300     950    800
    30 MARTIN SALESMAN   1250    2850   1600     950   1250
    30 TURNER SALESMAN   1500    2850   1600     950   1250
    30 WARD   SALESMAN   1250    2850   1600     950   1250
    30 ALLEN  SALESMAN   1600    2850   1600     950   1250
    30 BLAKE  MANAGER    2850    2850   2975     950   2450
```

Теперь каждую заработную плату можно сравнить с наибольшей и наименьшей для отдела (по DEPTNO) и должности (по JOB). Обратите внимание, что группировка (включение нескольких столбцов в оператор SELECT) не оказывает влияния на возвращаемые функциями MIN OVER и MAX OVER значения. В этом прелесть оконных функций: агрегат вычисляется для заданной «группы» или сегмента и возвращается в каждой строке соответствующей группы. Последний шаг – просто поместить оконные функции во вложенный запрос и выбрать только те строки, которые соответствуют возвращаемым ими значениям. Для отображения «статуса» служащих в окончательном результирующем множестве используйте простое выражение CASE:

```
select deptno,ename,job,sal,
       case when sal = max_by_dept
            then 'TOP SAL IN DEPT'
            when sal = min_by_dept
            then 'LOW SAL IN DEPT'
       end dept_status,
       case when sal = max_by_job
            then 'TOP SAL IN JOB'
            when sal = min_by_job
            then 'LOW SAL IN JOB'
       end job_status
  from (
select deptno,ename,job,sal,
       max(sal)over(partition by deptno) max_by_dept,
       max(sal)over(partition by job) max_by_job,
       min(sal)over(partition by deptno) min_by_dept,
```

```
       min(sal)over(partition by job) min_by_job
  from emp
       ) x
 where sal in (max_by_dept,max_by_job,
               min_by_dept,min_by_job)
```

```
DEPTNO ENAME  JOB         SAL DEPT_STATUS     JOB_STATUS
------ ------ --------- ----- --------------- --------------
    10 MILLER CLERK      1300 LOW SAL IN DEPT TOP SAL IN JOB
    10 CLARK  MANAGER    2450                 LOW SAL IN JOB
    10 KING   PRESIDENT  5000 TOP SAL IN DEPT TOP SAL IN JOB
    20 SCOTT  ANALYST    3000 TOP SAL IN DEPT TOP SAL IN JOB
    20 FORD   ANALYST    3000 TOP SAL IN DEPT TOP SAL IN JOB
    20 SMITH  CLERK       800 LOW SAL IN DEPT LOW SAL IN JOB
    20 JONES  MANAGER    2975                 TOP SAL IN JOB
    30 JAMES  CLERK       950 LOW SAL IN DEPT
    30 MARTIN SALESMAN   1250                 LOW SAL IN JOB
    30 WARD   SALESMAN   1250                 LOW SAL IN JOB
    30 ALLEN  SALESMAN   1600                 TOP SAL IN JOB
    30 BLAKE  MANAGER    2850 TOP SAL IN DEPT
```

PostgreSQL и MySQL

Первый шаг – с помощью скалярных подзапросов находим наибольшие и наименьшие заработные платы по DEPTNO и JOB.

```
select e.deptno,e.ename,e.job,e.sal,
       (select max(sal) from  emp d
         where d.deptno = e.deptno) as maxDEPT,
       (select max(sal) from  emp d
         where d.job = e.job) as maxJOB,
       (select min(sal) from  emp d
         where d.deptno = e.deptno) as minDEPT,
       (select min(sal) from  emp d
         where d.job = e.job) as minJOB
  from emp e
```

```
DEPTNO ENAME  JOB         SAL MAXDEPT MAXJOB MINDEPT MINJOB
------ ------ --------- ----- ------- ------ ------- ------
    20 SMITH  CLERK       800    3000   1300     800    800
    30 ALLEN  SALESMAN   1600    2850   1600     950   1250
    30 WARD   SALESMAN   1250    2850   1600     950   1250
    20 JONES  MANAGER    2975    3000   2975     800   2450
    30 MARTIN SALESMAN   1250    2850   1600     950   1250
    30 BLAKE  MANAGER    2850    2850   2975     950   2450
    10 CLARK  MANAGER    2450    5000   2975    1300   2450
    20 SCOTT  ANALYST    3000    3000   3000     800   3000
    10 KING   PRESIDENT  5000    5000   5000    1300   5000
    30 TURNER SALESMAN   1500    2850   1600     950   1250
    20 ADAMS  CLERK      1100    3000   1300     800    800
    30 JAMES  CLERK       950    2850   1300     950    800
    20 FORD   ANALYST    3000    3000   3000     800   3000
    10 MILLER CLERK      1300    5000   1300    1300    800
```

Теперь все заработные платы, представленные в таблице EMP, можно сравнить с наибольшими и наименьшими для отдела (по DEPTNO) и должности (по JOB). Последний шаг – поместить скалярные подзапросы во вложенный запрос и просто выбрать служащих, заработные платы которых соответствуют возвращаемым скалярными подзапросами. Для отображения «статуса» служащих в окончательном результирующем множестве используйте выражение CASE:

```
select deptno,ename,job,sal,
       case when sal = max_by_dept
            then 'TOP SAL IN DEPT'
            when sal = min_by_dept
            then 'LOW SAL IN DEPT'
       end as dept_status,
       case when sal = max_by_job
            then 'TOP SAL IN JOB'
            when sal = min_by_job
            then 'LOW SAL IN JOB'
       end as job_status
  from (
select e.deptno,e.ename,e.job,e.sal,
       (select max(sal) from  emp d
         where d.deptno = e.deptno) as max_by_dept,
       (select max(sal) from  emp d
         where d.job = e.job) as max_by_job,
       (select min(sal) from  emp d
         where d.deptno = e.deptno) as min_by_dept,
       (select min(sal) from  emp d
         where d.job = e.job) as min_by_job
  from emp e
       ) x
 where sal in (max_by_dept,max_by_job,
               min_by_dept,min_by_job)
```

```
DEPTNO ENAME  JOB         SAL DEPT_STATUS     JOB_STATUS
------ ------ --------- ----- --------------- --------------
    10 CLARK  MANAGER    2450                 LOW SAL IN JOB
    10 KING   PRESIDENT  5000 TOP SAL IN DEPT TOP SAL IN JOB
    10 MILLER CLERK      1300 LOW SAL IN DEPT TOP SAL IN JOB
    20 SMITH  CLERK       800 LOW SAL IN DEPT LOW SAL IN JOB
    20 FORD   ANALYST    3000 TOP SAL IN DEPT TOP SAL IN JOB
    20 SCOTT  ANALYST    3000 TOP SAL IN DEPT TOP SAL IN JOB
    20 JONES  MANAGER    2975                 TOP SAL IN JOB
    30 ALLEN  SALESMAN   1600                 TOP SAL IN JOB
    30 BLAKE  MANAGER    2850 TOP SAL IN DEPT
    30 MARTIN SALESMAN   1250                 LOW SAL IN JOB
    30 JAMES  CLERK       950 LOW SAL IN DEPT
    30 WARD   SALESMAN   1250                 LOW SAL IN JOB
```

Вычисление простых подсумм

Задача

В данном рецепте под «простой подсуммой» подразумевается результирующее множество, содержащее значения, полученные в результате агрегации одного столбца, и общую сумму таблицы. В качестве примера возьмем результирующее множество, содержащее суммы заработных плат таблицы EMP по должностям (JOB), а также сумму всех заработных плат таблицы EMP. Суммы зарплат по JOB – это подсуммы, а сумма всех заработных плат таблицы EMP – это общая сумма. Такое результирующее множество выглядело бы так:

```
JOB              SAL
--------- ----------
ANALYST         6000
CLERK           4150
MANAGER         8275
PRESIDENT       5000
SALESMAN        5600
TOTAL          29025
```

Решение

Расширение ROLLUP оператора GROUP BY идеально справляется с этой задачей. Если СУБД не поддерживает ROLLUP, задачу можно решить, хотя и немного сложнее, с помощью скалярного подзапроса или запроса UNION.

DB2 и Oracle

Используя агрегатную функцию SUM, просуммируйте заработные платы; с помощью расширения ROLLUP оператора GROUP BY организуйте результаты в подсуммы (по JOB) и найдите общую сумму (для всей таблицы):

```
1 select case grouping(job)
2             when 0 then job
3             else 'TOTAL'
4        end job,
5        sum(sal) sal
6   from emp
7  group by rollup(job)
```

SQL Server и MySQL

Используя агрегатную функцию SUM, просуммируйте заработные платы; с помощью WITH ROLLUP организуйте результаты в подсуммы (по JOB) и найдите общую сумму (для всей таблицы). Затем посредством функции COALESCE задайте имя «TOTAL» для строки общей суммы (в противном случае в столбце JOB этой строки будет располагаться значение NULL):

```
1 select coalesce(job,'TOTAL') job,
2        sum(sal) sal
3   from emp
4  group by job with rollup
```

Для SQL Server для определения уровня агрегации можно использовать вместо COALESCE функцию GROUPING, показанную в рецепте Oracle/DB2.

PostgreSQL

Используя агрегатную функцию SUM, просуммируйте заработные платы по DEPTNO. Затем посредством оператора UNION ALL объедините этот запрос с запросом, вычисляющим сумму всех заработных плат таблицы:

```
1 select job, sum(sal) as sal
2   from emp
3  group by job
4  union all
5 select 'TOTAL', sum(sal)
6   from emp
```

Обсуждение

DB2 и Oracle

Первый шаг – с помощью агрегатной функции SUM, группируя по столбцу JOB, найти суммы заработных плат для каждой должности (JOB):

```
select job, sum(sal) sal
  from emp
 group by job
```

```
JOB         SAL
--------- -----
ANALYST    6000
CLERK      4150
MANAGER    8275
PRESIDENT  5000
SALESMAN   5600
```

Следующий шаг – использовать расширение ROLLUP оператора GROUP BY для формирования общей суммы всех заработных плат помимо подсумм для каждой JOB:

```
select job, sum(sal) sal
  from emp
 group by rollup(job)
```

```
JOB           SAL
--------- -------
ANALYST      6000
CLERK        4150
```

```
MANAGER      8275
PRESIDENT    5000
SALESMAN     5600
            29025
```

Последний шаг – применить к столбцу JOB функцию GROUPING для отображения имени поля общей суммы. Если значение JOB – NULL, функция GROUPING возвратит 1, свидетельствующую о том, что значение SAL является общей суммой, созданной ROLLUP. Если значение JOB не NULL, функция GROUPING возвратит 0, свидетельствующий о том, что значение SAL является результатом GROUP BY, а не ROLLUP. Поместите вызов GROUPING(JOB) в выражение CASE, которое будет возвращать либо название должности, либо имя «TOTAL» соответственно:

```
select case grouping(job)
            when 0 then job
            else 'TOTAL'
       end job,
       sum(sal) sal
  from emp
 group by rollup(job)
```

```
JOB              SAL
--------- ----------
ANALYST          6000
CLERK            4150
MANAGER          8275
PRESIDENT        5000
SALESMAN         5600
TOTAL           29025
```

SQL Server и MySQL

Первый шаг – использовать агрегатную функцию SUM, группируя результаты по JOB, чтобы получить суммы заработных плат по должностям:

```
select job, sum(sal) sal
  from emp
 group by job
```

```
JOB         SAL
--------- -----
ANALYST    6000
CLERK      4150
MANAGER    8275
PRESIDENT  5000
SALESMAN   5600
```

Следующий шаг – использовать расширение ROLLUP оператора GROUP BY, чтобы помимо подсумм для каждой должности найти общую сумму всех заработных плат:

```
select job, sum(sal) sal
  from emp
 group by job with rollup
```

```
JOB           SAL
--------- -------
ANALYST      6000
CLERK        4150
MANAGER      8275
PRESIDENT    5000
SALESMAN     5600
            29025
```

Последний шаг – применить функцию COALESCE к столбцу JOB. Если значение JOB – NULL, значение SAL является общей суммой, созданной ROLLUP. Если значение JOB не NULL, значение SAL является результатом «обычного» GROUP BY, а не ROLLUP:

```
select coalesce(job,'TOTAL') job,
       sum(sal) sal
  from emp
 group by job with rollup
```

```
JOB              SAL
--------- ----------
ANALYST          6000
CLERK            4150
MANAGER          8275
PRESIDENT        5000
SALESMAN         5600
TOTAL           29025
```

PostgreSQL

Первый шаг – сгруппировать результаты, возвращаемые агрегатной функцией SUM, по должностям:

```
select job, sum(sal) sal
  from emp
 group by job
```

```
JOB          SAL
--------- -----
ANALYST    6000
CLERK      4150
MANAGER    8275
PRESIDENT  5000
SALESMAN   5600
```

Последний шаг – используя оператор UNION ALL, найти общую сумму результатов, возвращенных первым запросом:

```
select job, sum(sal) as sal
  from emp
 group by job
```

```
 union all
select 'TOTAL', sum(sal)
  from emp
```

```
JOB           SAL
--------- -------
ANALYST      6000
CLERK        4150
MANAGER      8275
PRESIDENT    5000
SALESMAN     5600
TOTAL       29025
```

Вычисление подсумм для всех возможных сочетаний

Задача

Требуется найти суммы всех заработных плат по отделам (группировка по столбцу DEPTNO), по должностям (группировка по столбцу JOB) и для каждого сочетания JOB/DEPTNO. Должна быть также вычислена общая сумма всех заработных плат таблицы EMP. Необходимо получить следующее результирующее множество:

```
DEPTNO JOB       CATEGORY                  SAL
------ --------- -------------------- -------
    10 CLERK     TOTAL BY DEPT AND JOB    1300
    10 MANAGER   TOTAL BY DEPT AND JOB    2450
    10 PRESIDENT TOTAL BY DEPT AND JOB    5000
    20 CLERK     TOTAL BY DEPT AND JOB    1900
    30 CLERK     TOTAL BY DEPT AND JOB     950
    30 SALESMAN  TOTAL BY DEPT AND JOB    5600
    30 MANAGER   TOTAL BY DEPT AND JOB    2850
    20 MANAGER   TOTAL BY DEPT AND JOB    2975
    20 ANALYST   TOTAL BY DEPT AND JOB    6000
       CLERK     TOTAL BY JOB             4150
       ANALYST   TOTAL BY JOB             6000
       MANAGER   TOTAL BY JOB             8275
       PRESIDENT TOTAL BY JOB             5000
       SALESMAN  TOTAL BY JOB             5600
    10           TOTAL BY DEPT            8750
    30           TOTAL BY DEPT            9400
    20           TOTAL BY DEPT           10875
                 GRAND TOTAL FOR TABLE   29025
```

Решение

Расширения оператора GROUP BY, появившиеся в последние годы, значительно упрощают решение этой задачи. Если используемая платформа не поддерживает расширения, позволяющие определять подсуммы различных уровней, придется вычислять их самостоятельно (посредством рефлексивных объединений или скалярных подзапросов).

DB2

Для DB2 потребуется приводить (с помощью функции CAST) результаты, возвращаемые GROUPING, к типу данных CHAR(1):

```
select deptno,
       job,
       case cast(grouping(deptno) as char(1))||
            cast(grouping(job) as char(1))
            when '00' then 'TOTAL BY DEPT AND JOB'
            when '10' then 'TOTAL BY JOB'
            when '01' then 'TOTAL BY DEPT'
            when '11' then 'TOTAL FOR TABLE'
       end category,
       sum(sal)
  from emp
 group by cube(deptno,job)
 order by grouping(job),grouping(deptno)
```

Oracle

Используйте расширение CUBE оператора GROUP BY в сочетании с оператором конкатенации ||:

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
            when '00' then 'TOTAL BY DEPT AND JOB'
            when '10' then 'TOTAL BY JOB'
            when '01' then 'TOTAL BY DEPT'
            when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by cube(deptno,job)
 order by grouping(job),grouping(deptno)
```

SQL Server

Используйте расширение CUBE оператора GROUP BY. Для SQL Server потребуется привести (CAST) результаты, возвращаемые GROUPING, к типу CHAR(1) и использовать оператор конкатенации + (а не оператор ||, применяемый в Oracle):

```
select deptno,
       job,
       case cast(grouping(deptno)as char(1))+
            cast(grouping(job)as char(1))
            when '00' then 'TOTAL BY DEPT AND JOB'
            when '10' then 'TOTAL BY JOB'
            when '01' then 'TOTAL BY DEPT'
            when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
```

```
11    from emp
12   group by deptno,job with cube
13   order by grouping(job),grouping(deptno)
```

PostgreSQL и MySQL

Суммы для разных столбцов и их сочетаний формируются с помощью многократного применения оператора UNION ALL:

```
 1  select deptno, job,
 2         'TOTAL BY DEPT AND JOB' as category,
 3         sum(sal) as sal
 4    from emp
 5   group by deptno, job
 6   union all
 7  select null, job, 'TOTAL BY JOB', sum(sal)
 8    from emp
 9   group by job
10   union all
11  select deptno, null, 'TOTAL BY DEPT', sum(sal)
12    from emp
13   group by deptno
14   union all
15  select null,null,'GRAND TOTAL FOR TABLE', sum(sal)
16    from emp
```

Обсуждение

Oracle, DB2 и SQL Server

Решения для всех трех СУБД, по сути, одинаковые. Первый шаг – найти суммарные заработные платы для каждого сочетания JOB и DEPTNO, применяя агрегатную функцию SUM и группируя значения по DEPTNO и JOB:

```
select deptno, job, sum(sal) sal
  from emp
 group by deptno, job
```

```
DEPTNO JOB           SAL
------ --------- -------
    10 CLERK        1300
    10 MANAGER      2450
    10 PRESIDENT    5000
    20 CLERK        1900
    20 ANALYST      6000
    20 MANAGER      2975
    30 CLERK         950
    30 MANAGER      2850
    30 SALESMAN     5600
```

Следующий шаг – вычислить подсуммы по JOB и DEPTNO и общую сумму для всей таблицы. С помощью расширения CUBE оператора

GROUP BY осуществляем агрегацию значений SAL по DEPTNO, JOB и затем для всей таблицы:

```
select deptno,
       job,
       sum(sal) sal
  from emp
 group by cube(deptno,job)
```

```
DEPTNO JOB           SAL
------ --------- -------
                   29025
       CLERK        4150
       ANALYST      6000
       MANAGER      8275
       SALESMAN     5600
       PRESIDENT    5000
    10               8750
    10 CLERK         1300
    10 MANAGER       2450
    10 PRESIDENT     5000
    20              10875
    20 CLERK         1900
    20 ANALYST       6000
    20 MANAGER       2975
    30               9400
    30 CLERK          950
    30 MANAGER       2850
    30 SALESMAN      5600
```

Далее используем функцию GROUPING в сочетании с выражением CASE, чтобы представить результаты в более выразительном формате. GROUPING(JOB) возвращает значения 1 или 0 в зависимости от того, получены ли значения SAL оператором GROUP BY или его расширением CUBE. Если значение возвращено CUBE, получаем 1, в противном случае – 0. Аналогично для GROUPING(DEPTNO). Из первого шага решения видим, что группировка выполняется по DEPTNO и JOB. Таким образом, в результате вызова GROUPING для строки, представляющей сочетание DEPTNO и JOB, должен быть возвращен 0. Запрос ниже подтверждает это:

```
select deptno,
       job,
       grouping(deptno) is_deptno_subtotal,
       grouping(job) is_job_subtotal,
       sum(sal) sal
  from emp
 group by cube(deptno,job)
 order by 3,4
```

```
DEPTNO JOB       IS_DEPTNO_SUBTOTAL IS_JOB_SUBTOTAL      SAL
------ --------- ------------------ --------------- -------
    10 CLERK                      0               0    1300
```

```
    10 MANAGER                   0            0    2450
    10 PRESIDENT                 0            0    5000
    20 CLERK                     0            0    1900
    30 CLERK                     0            0     950
    30 SALESMAN                  0            0    5600
    30 MANAGER                   0            0    2850
    20 MANAGER                   0            0    2975
    20 ANALYST                   0            0    6000
    10                           0            1    8750
    20                           0            1   10875
    30                           0            1    9400
       CLERK                     1            0    4150
       ANALYST                   1            0    6000
       MANAGER                   1            0    8275
       PRESIDENT                 1            0    5000
       SALESMAN                  1            0    5600
                                 1            1   29025
```

Заключительный шаг – использовать выражение CASE для определения категории строки на основании значений, возвращенных в результате конкатенации GROUPING(JOB) и GROUPING(DEPTNO):

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
            when '00' then 'TOTAL BY DEPT AND JOB'
            when '10' then 'TOTAL BY JOB'
            when '01' then 'TOTAL BY DEPT'
            when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by cube(deptno,job)
 order by grouping(job),grouping(deptno)
```

```
DEPTNO JOB       CATEGORY                  SAL
------ --------- -------------------- -------
    10 CLERK     TOTAL BY DEPT AND JOB    1300
    10 MANAGER   TOTAL BY DEPT AND JOB    2450
    10 PRESIDENT TOTAL BY DEPT AND JOB    5000
    20 CLERK     TOTAL BY DEPT AND JOB    1900
    30 CLERK     TOTAL BY DEPT AND JOB     950
    30 SALESMAN  TOTAL BY DEPT AND JOB    5600
    30 MANAGER   TOTAL BY DEPT AND JOB    2850
    20 MANAGER   TOTAL BY DEPT AND JOB    2975
    20 ANALYST   TOTAL BY DEPT AND JOB    6000
       CLERK     TOTAL BY JOB             4150
       ANALYST   TOTAL BY JOB             6000
       MANAGER   TOTAL BY JOB             8275
       PRESIDENT TOTAL BY JOB             5000
       SALESMAN  TOTAL BY JOB             5600
    10           TOTAL BY DEPT            8750
```

```
    30          TOTAL BY DEPT           9400
    20          TOTAL BY DEPT          10875
                GRAND TOTAL FOR TABLE  29025
```

В этом решении для Oracle при подготовке к конкатенации результаты, возвращаемые функциями GROUPING, неявно преобразуются в символьный тип данных. Пользователям DB2 и SQL Server придется явно приводить (используя функцию CAST) результаты функций GROUPING к типу CHAR(1), как показано в решении. Кроме того, в SQL Server для объединения результатов двух вызовов GROUPING в одну строку используется оператор конкатенации +, а не ||.

Пользователям Oracle и DB2 доступно дополнительное исключительно полезное расширение GROUP BY под названием GROUPING SETS. С помощью GROUPING SETS можно, например, имитировать вывод, создаваемый CUBE, как это сделано ниже (пользователям DB2 и SQL Server потребуется применить к значениям, возвращаемым функцией GROUPING, явные операторы CAST, как в решении с расширением CUBE):

```
select deptno,
       job,
       case grouping(deptno)||grouping(job)
            when '00' then 'TOTAL BY DEPT AND JOB'
            when '10' then 'TOTAL BY JOB'
            when '01' then 'TOTAL BY DEPT'
            when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by grouping sets ((deptno),(job),(deptno,job),())
```

```
DEPTNO JOB       CATEGORY                 SAL
------ --------- -------------------- -------
    10 CLERK     TOTAL BY DEPT AND JOB    1300
    20 CLERK     TOTAL BY DEPT AND JOB    1900
    30 CLERK     TOTAL BY DEPT AND JOB     950
    20 ANALYST   TOTAL BY DEPT AND JOB    6000
    10 MANAGER   TOTAL BY DEPT AND JOB    2450
    20 MANAGER   TOTAL BY DEPT AND JOB    2975
    30 MANAGER   TOTAL BY DEPT AND JOB    2850
    30 SALESMAN  TOTAL BY DEPT AND JOB    5600
    10 PRESIDENT TOTAL BY DEPT AND JOB    5000
       CLERK     TOTAL BY JOB             4150
       ANALYST   TOTAL BY JOB             6000
       MANAGER   TOTAL BY JOB             8275
       SALESMAN  TOTAL BY JOB             5600
       PRESIDENT TOTAL BY JOB             5000
    10           TOTAL BY DEPT            8750
    20           TOTAL BY DEPT           10875
    30           TOTAL BY DEPT            9400
                 GRAND TOTAL FOR TABLE   29025
```

GROUPING SETS замечателен тем, что позволяет задавать группы. Оператор GROUPING SETS в предыдущем запросе обусловливает создание групп по DEPTNO, по JOB, по сочетанию DEPTNO и JOB, и, наконец, пустые круглые скобки соответствуют запросу на получение общей суммы. GROUPING SETS обеспечивает колоссальную гибкость для создания отчетов с разными уровнями агрегации. Например, чтобы в предыдущем примере исключить из результирующего множества общую сумму (GRAND TOTAL), надо просто убрать из списка оператора GROUPING SETS пустые круглые скобки:

```
/* нет общей суммы */

select deptno,
       job,
       case grouping(deptno)||grouping(job)
            when '00' then 'TOTAL BY DEPT AND JOB'
            when '10' then 'TOTAL BY JOB'
            when '01' then 'TOTAL BY DEPT'
            when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by grouping sets ((deptno),(job),(deptno,job))

DEPTNO JOB       CATEGORY                    SAL
------ --------- -------------------- ----------
    10 CLERK     TOTAL BY DEPT AND JOB       1300
    20 CLERK     TOTAL BY DEPT AND JOB       1900
    30 CLERK     TOTAL BY DEPT AND JOB        950
    20 ANALYST   TOTAL BY DEPT AND JOB       6000
    10 MANAGER   TOTAL BY DEPT AND JOB       2450
    20 MANAGER   TOTAL BY DEPT AND JOB       2975
    30 MANAGER   TOTAL BY DEPT AND JOB       2850
    30 SALESMAN  TOTAL BY DEPT AND JOB       5600
    10 PRESIDENT TOTAL BY DEPT AND JOB       5000
       CLERK     TOTAL BY JOB                4150
       ANALYST   TOTAL BY JOB                6000
       MANAGER   TOTAL BY JOB                8275
       SALESMAN  TOTAL BY JOB                5600
       PRESIDENT TOTAL BY JOB                5000
    10           TOTAL BY DEPT               8750
    20           TOTAL BY DEPT              10875
    30           TOTAL BY DEPT               9400
```

Можно также убрать какую-то из подсумм, например по DEPTNO, просто опуская (DEPTNO) в списке GROUPING SETS:

```
/* нет подсумм по DEPTNO */

select deptno,
       job,
       case grouping(deptno)||grouping(job)
            when '00' then 'TOTAL BY DEPT AND JOB'
```

```
            when '10' then 'TOTAL BY JOB'
            when '01' then 'TOTAL BY DEPT'
            when '11' then 'GRAND TOTAL FOR TABLE'
       end category,
       sum(sal) sal
  from emp
 group by grouping sets ((job),(deptno,job),())
 order by 3
```

```
DEPTNO JOB       CATEGORY                    SAL
------ --------- -------------------- ----------
                 GRAND TOTAL FOR TABLE      29025
    10 CLERK     TOTAL BY DEPT AND JOB       1300
    20 CLERK     TOTAL BY DEPT AND JOB       1900
    30 CLERK     TOTAL BY DEPT AND JOB        950
    20 ANALYST   TOTAL BY DEPT AND JOB       6000
    20 MANAGER   TOTAL BY DEPT AND JOB       2975
    30 MANAGER   TOTAL BY DEPT AND JOB       2850
    30 SALESMAN  TOTAL BY DEPT AND JOB       5600
    10 PRESIDENT TOTAL BY DEPT AND JOB       5000
    10 MANAGER   TOTAL BY DEPT AND JOB       2450
       CLERK     TOTAL BY JOB                4150
       SALESMAN  TOTAL BY JOB                5600
       PRESIDENT TOTAL BY JOB                5000
       MANAGER   TOTAL BY JOB                8275
       ANALYST   TOTAL BY JOB                6000
```

Как видите, с GROUPING SETS очень просто манипулировать суммами и подсуммами, представляя данные в разных ракурсах.

PostgreSQL и MySQL

Первый шаг – использовать агрегатную функцию SUM и группировать значения по DEPTNO и JOB:

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
```

```
DEPTNO JOB       CATEGORY                   SAL
------ --------- -------------------- -------
    10 CLERK     TOTAL BY DEPT AND JOB    1300
    10 MANAGER   TOTAL BY DEPT AND JOB    2450
    10 PRESIDENT TOTAL BY DEPT AND JOB    5000
    20 CLERK     TOTAL BY DEPT AND JOB    1900
    20 ANALYST   TOTAL BY DEPT AND JOB    6000
    20 MANAGER   TOTAL BY DEPT AND JOB    2975
    30 CLERK     TOTAL BY DEPT AND JOB     950
    30 MANAGER   TOTAL BY DEPT AND JOB    2850
    30 SALESMAN  TOTAL BY DEPT AND JOB    5600
```

Далее с помощью UNION ALL находим суммы всех заработных плат по должностям (JOB):

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
```

```
DEPTNO JOB       CATEGORY                 SAL
------ --------- -------------------- -------
    10 CLERK     TOTAL BY DEPT AND JOB    1300
    10 MANAGER   TOTAL BY DEPT AND JOB    2450
    10 PRESIDENT TOTAL BY DEPT AND JOB    5000
    20 CLERK     TOTAL BY DEPT AND JOB    1900
    20 ANALYST   TOTAL BY DEPT AND JOB    6000
    20 MANAGER   TOTAL BY DEPT AND JOB    2975
    30 CLERK     TOTAL BY DEPT AND JOB     950
    30 MANAGER   TOTAL BY DEPT AND JOB    2850
    30 SALESMAN  TOTAL BY DEPT AND JOB    5600
       ANALYST   TOTAL BY JOB             6000
       CLERK     TOTAL BY JOB             4150
       MANAGER   TOTAL BY JOB             8275
       PRESIDENT TOTAL BY JOB             5000
       SALESMAN  TOTAL BY JOB             5600
```

Следующий шаг – с помощью UNION ALL находим суммы всех заработных плат по отделам (DEPTNO):

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
 union all
select deptno, null, 'TOTAL BY DEPT', sum(sal)
  from emp
 group by deptno
```

```
DEPTNO JOB       CATEGORY                 SAL
------ --------- -------------------- -------
    10 CLERK     TOTAL BY DEPT AND JOB    1300
    10 MANAGER   TOTAL BY DEPT AND JOB    2450
    10 PRESIDENT TOTAL BY DEPT AND JOB    5000
    20 CLERK     TOTAL BY DEPT AND JOB    1900
```

```
    20 ANALYST   TOTAL BY DEPT AND JOB    6000
    20 MANAGER   TOTAL BY DEPT AND JOB    2975
    30 CLERK     TOTAL BY DEPT AND JOB     950
    30 MANAGER   TOTAL BY DEPT AND JOB    2850
    30 SALESMAN  TOTAL BY DEPT AND JOB    5600
       ANALYST   TOTAL BY JOB             6000
       CLERK     TOTAL BY JOB             4150
       MANAGER   TOTAL BY JOB             8275
       PRESIDENT TOTAL BY JOB             5000
       SALESMAN  TOTAL BY JOB             5600
    10           TOTAL BY DEPT            8750
    20           TOTAL BY DEPT           10875
    30           TOTAL BY DEPT            9400
```

Заключительный шаг – посредством UNION ALL вычисляем сумму всех заработных плат таблицы EMP:

```
select deptno, job,
       'TOTAL BY DEPT AND JOB' as category,
       sum(sal) as sal
  from emp
 group by deptno, job
 union all
select null, job, 'TOTAL BY JOB', sum(sal)
  from emp
 group by job
 union all
select deptno, null, 'TOTAL BY DEPT', sum(sal)
  from emp
 group by deptno
 union all
select null,null, 'GRAND TOTAL FOR TABLE', sum(sal)
  from emp

DEPTNO JOB       CATEGORY                  SAL
------ --------- -------------------- -------
    10 CLERK     TOTAL BY DEPT AND JOB    1300
    10 MANAGER   TOTAL BY DEPT AND JOB    2450
    10 PRESIDENT TOTAL BY DEPT AND JOB    5000
    20 CLERK     TOTAL BY DEPT AND JOB    1900
    20 ANALYST   TOTAL BY DEPT AND JOB    6000
    20 MANAGER   TOTAL BY DEPT AND JOB    2975
    30 CLERK     TOTAL BY DEPT AND JOB     950
    30 MANAGER   TOTAL BY DEPT AND JOB    2850
    30 SALESMAN  TOTAL BY DEPT AND JOB    5600
       ANALYST   TOTAL BY JOB             6000
       CLERK     TOTAL BY JOB             4150
       MANAGER   TOTAL BY JOB             8275
       PRESIDENT TOTAL BY JOB             5000
       SALESMAN  TOTAL BY JOB             5600
    10           TOTAL BY DEPT            8750
    20           TOTAL BY DEPT           10875
```

```
        30          TOTAL BY DEPT           9400
                    GRAND TOTAL FOR TABLE   29025
```

Как выявить строки, в которых представлены не подсуммы

Задача

При создании отчета использовалось расширение CUBE оператора GROUP BY. Как отличить строки, сформированные обычным оператором GROUP BY, и строки, являющиеся результатом выполнения CUBE или ROLLUP?

Ниже представлено результирующее множество, возвращаемое запросом, в котором для анализа заработных плат таблицы EMP используется расширение CUBE оператора GROUP BY:

```
DEPTNO JOB           SAL
------ --------- -------
                   29025
       CLERK        4150
       ANALYST      6000
       MANAGER      8275
       SALESMAN     5600
       PRESIDENT    5000
    10               8750
    10 CLERK         1300
    10 MANAGER       2450
    10 PRESIDENT     5000
    20              10875
    20 CLERK         1900
    20 ANALYST       6000
    20 MANAGER       2975
    30               9400
    30 CLERK          950
    30 MANAGER       2850
    30 SALESMAN      5600
```

Этот отчет включает сумму всех заработных плат по DEPTNO и JOB (для каждой должности по отделам), сумму всех заработных плат по DEPTNO (по отделам), сумму всех заработных плат по JOB (по должностям) и, наконец, общую сумму заработных плат (сумму всех заработных плат таблицы EMP). Уровни агрегации должны быть четко обозначены. Для каждого значения должно быть точно определено, к какой категории оно относится (т. е. представляет ли данное значение столбца SAL сумму по DEPTNO? По JOB? Общую сумму?) Требуется получить следующее результирующее множество:

```
DEPTNO JOB           SAL DEPTNO_SUBTOTALS JOB_SUBTOTALS
------ --------- ------- ---------------- -------------
                   29025                1             1
```

```
    CLERK        4150              1              0
    ANALYST      6000              1              0
    MANAGER      8275              1              0
    SALESMAN     5600              1              0
    PRESIDENT    5000              1              0
10                8750              0              1
10  CLERK         1300              0              0
10  MANAGER       2450              0              0
10  PRESIDENT     5000              0              0
20                10875             0              1
20  CLERK         1900              0              0
20  ANALYST       6000              0              0
20  MANAGER       2975              0              0
30                9400              0              1
30  CLERK          950              0              0
30  MANAGER       2850              0              0
30  SALESMAN      5600              0              0
```

Решение

Чтобы обозначить, какие из значений получены благодаря выполнению CUBE или ROLLUP, а какие из них являются *супераграгатными*, используйте функцию GROUPING. Ниже представлен пример для DB2 и Oracle:

```
1  select deptno, job, sum(sal) sal,
2         grouping(deptno) deptno_subtotals,
3         grouping(job) job_subtotals
4    from emp
5   group by cube(deptno,job)
```

Решение для SQL Server отличается от решения для DB2 и Oracle только записью операторов CUBE/ROLLUP:

```
1  select deptno, job, sum(sal) sal,
2         grouping(deptno) deptno_subtotals,
3         grouping(job) job_subtotals
4    from emp
5   group by deptno,job with cube
```

Этот рецепт посвящен использованию CUBE и GROUPING при работе с подсуммами. На момент написания данной книги PostgreSQL и MySQL не поддерживают ни CUBE, ни GROUPING.

Обсуждение

Если DEPTNO_SUBTOTALS равно 1, то значение поля SAL представляет подсумму по DEPTNO, созданную CUBE. Если JOB_SUBTOTALS равно 1, то значение поля SAL представляет подсумму по JOB, созданную CUBE. Если в обоих столбцах, JOB_SUBTOTALS и DEPTNO_SUBTOTALS, располагается 1, значение SAL представляет общую сумму всех заработных плат, сформированную CUBE. Строки, в которых оба

значения, DEPTNO_SUBTOTALS и JOB_SUBTOTALS, равны 0, созданы в результате обычной агрегации (значение SAL представляет сумму заработных плат для каждого сочетания DEPTNO/JOB).

Использование выражений CASE для маркировки строк

Задача

Требуется представить значения столбца, скажем, столбца JOB таблицы EMP, в виде набора «логических» флагов. Например, необходимо получить следующее результирующее множество:

```
ENAME  IS_CLERK IS_SALES IS_MGR IS_ANALYST IS_PREZ
------ -------- -------- ------ ---------- -------
KING          0        0      0          0       1
SCOTT         0        0      0          1       0
FORD          0        0      0          1       0
JONES         0        0      1          0       0
BLAKE         0        0      1          0       0
CLARK         0        0      1          0       0
ALLEN         0        1      0          0       0
WARD          0        1      0          0       0
MARTIN        0        1      0          0       0
TURNER        0        1      0          0       0
SMITH         1        0      0          0       0
MILLER        1        0      0          0       0
ADAMS         1        0      0          0       0
JAMES         1        0      0          0       0
```

Подобное результирующее множество может использоваться для отладки и для представления данных иначе, чем предлагают обычные результирующие множества.

Решение

С помощью выражения CASE для каждого служащего определите должность (JOB) и возвратите 1 или 0, чтобы обозначить ее. Потребуется написать выражение CASE и, таким образом, создать столбец для каждой из возможных должностей:

```
 1  select ename,
 2         case when job = 'CLERK'
 3              then 1 else 0
 4         end as is_clerk,
 5         case when job = 'SALESMAN'
 6              then 1 else 0
 7         end as is_sales,
 8         case when job = 'MANAGER'
 9              then 1 else 0
10         end as is_mgr,
```

```
11         case when job = 'ANALYST'
12              then 1 else 0
13         end as is_analyst,
14         case when job = 'PRESIDENT'
15              then 1 else 0
16         end as is_prez
17    from emp
18   order by 2,3,4,5,6
```

Обсуждение

Код данного решения практически не требует пояснений. Если возникли трудности с его пониманием, просто добавьте в оператор SELECT столбец JOB:

```
select ename,
       job,
       case when job = 'CLERK'
            then 1 else 0
       end as is_clerk,
       case when job = 'SALESMAN'
            then 1 else 0
       end as is_sales,
       case when job = 'MANAGER'
            then 1 else 0
       end as is_mgr,
       case when job = 'ANALYST'
            then 1 else 0
       end as is_analyst,
       case when job = 'PRESIDENT'
            then 1 else 0
       end as is_prez
  from emp
 order by 2
```

```
ENAME  JOB       IS_CLERK IS_SALES IS_MGR IS_ANALYST IS_PREZ
------ --------- -------- -------- ------ ---------- -------
SCOTT  ANALYST          0        0      0          1       0
FORD   ANALYST          0        0      0          1       0
SMITH  CLERK            1        0      0          0       0
ADAMS  CLERK            1        0      0          0       0
MILLER CLERK            1        0      0          0       0
JAMES  CLERK            1        0      0          0       0
JONES  MANAGER          0        0      1          0       0
CLARK  MANAGER          0        0      1          0       0
BLAKE  MANAGER          0        0      1          0       0
KING   PRESIDENT        0        0      0          0       1
ALLEN  SALESMAN         0        1      0          0       0
MARTIN SALESMAN         0        1      0          0       0
TURNER SALESMAN         0        1      0          0       0
WARD   SALESMAN         0        1      0          0       0
```

Создание разреженной матрицы

Задача

Требуется создать разреженную матрицу, такую как представлена ниже, в которой транспонированы столбцы DEPTNO и JOB таблицы EMP:

```
D10        D20        D30        CLERKS MGRS  PREZ ANALS SALES
---------- ---------- ---------- ------ ----- ---- ----- ------
           SMITH                 SMITH
                      ALLEN                                ALLEN
                      WARD                                 WARD
           JONES                        JONES
                      MARTIN                               MARTIN
                      BLAKE              BLAKE
CLARK                                    CLARK
           SCOTT                                  SCOTT
KING                                           KING
                      TURNER                               TURNER
           ADAMS                 ADAMS
                      JAMES      JAMES
           FORD                                   FORD
MILLER                           MILLER
```

Решение

Для создания разреженной таблицы, в которой строки транспонированы в столбцы, используйте выражения CASE:

```
1  select case deptno when 10 then ename end as d10,
2         case deptno when 20 then ename end as d20,
3         case deptno when 30 then ename end as d30,
4         case job when 'CLERK'     then ename end as clerks,
5         case job when 'MANAGER'   then ename end as mgrs,
6         case job when 'PRESIDENT' then ename end as prez,
7         case job when 'ANALYST'   then ename end as anals,
8         case job when 'SALESMAN'  then ename end as sales
9    from emp
```

Обсуждение

Чтобы превратить строки DEPTNO и JOB в столбцы, просто используем выражение CASE, обрабатывающее значения, которые могут быть возвращены в этих строках. Вот и все. В дополнение к этому, если требуется «уплотнить» отчет и избавиться от строк со значениями NULL, необходимо определиться с принципом группировки. Например, с помощью ранжирующей функции ROW_NUMBER OVER присвойте ранги всем служащим, разделяя их по DEPTNO, и затем, применяя агрегатную функцию MAX, удалите некоторые значения NULL:

```
select max(case deptno when 10 then ename end) d10,
       max(case deptno when 20 then ename end) d20,
       max(case deptno when 30 then ename end) d30,
```

```
       max(case job when 'CLERK'     then ename end) clerks,
       max(case job when 'MANAGER'   then ename end) mgrs,
       max(case job when 'PRESIDENT' then ename end) prez,
       max(case job when 'ANALYST'   then ename end) anals,
       max(case job when 'SALESMAN'  then ename end) sales
  from (
select deptno, job, ename,
       row_number()over(partition by deptno order by empno) rn
  from emp
       ) x
 group by rn
```

```
D10        D20        D30        CLERKS MGRS  PREZ ANALS SALES
---------- ---------- ---------- ------ ----- ---- ----- ------
CLARK      SMITH      ALLEN      SMITH  CLARK            ALLEN
KING       JONES      WARD              JONES KING       WARD
MILLER     SCOTT      MARTIN     MILLER            SCOTT MARTIN
           ADAMS      BLAKE      ADAMS  BLAKE
           FORD       TURNER                       FORD  TURNER
                      JAMES      JAMES
```

Группировка строк по интервалам времени

Задача

Требуется обобщить данные по некоторому интервалу времени. Например, имеется журнал транзакций. Необходимо разбить период наблюдений на 5-секундные интервалы и показать, сколько транзакций имело место в каждый из этих интервалов. Строки таблицы TRX_LOG показаны ниже:

```
select trx_id,
       trx_date,
       trx_cnt
  from trx_log
```

```
TRX_ID TRX_DATE                TRX_CNT
------ -------------------- ----------
     1 28-JUL-2005 19:03:07         44
     2 28-JUL-2005 19:03:08         18
     3 28-JUL-2005 19:03:09         23
     4 28-JUL-2005 19:03:10         29
     5 28-JUL-2005 19:03:11         27
     6 28-JUL-2005 19:03:12         45
     7 28-JUL-2005 19:03:13         45
     8 28-JUL-2005 19:03:14         32
     9 28-JUL-2005 19:03:15         41
    10 28-JUL-2005 19:03:16         15
    11 28-JUL-2005 19:03:17         24
    12 28-JUL-2005 19:03:18         47
    13 28-JUL-2005 19:03:19         37
```

```
    14 28-JUL-2005 19:03:20         48
    15 28-JUL-2005 19:03:21         46
    16 28-JUL-2005 19:03:22         44
    17 28-JUL-2005 19:03:23         36
    18 28-JUL-2005 19:03:24         41
    19 28-JUL-2005 19:03:25         33
    20 28-JUL-2005 19:03:26         19
```

Должно быть получено следующее результирующее множество:

```
GRP TRX_START           TRX_END                 TOTAL
--- ------------------- ------------------- ----------
  1 28-JUL-2005 19:03:07 28-JUL-2005 19:03:11        141
  2 28-JUL-2005 19:03:12 28-JUL-2005 19:03:16        178
  3 28-JUL-2005 19:03:17 28-JUL-2005 19:03:21        202
  4 28-JUL-2005 19:03:22 28-JUL-2005 19:03:26        173
```

Решение

Группировать записи в блоки по пять строк. Такую логическую группировку можно реализовать несколькими способами. В данном рецепте она осуществляется путем деления значений TRX_ID на 5, т. е. используется техника, представленная ранее в разделе «Создание блоков данных фиксированного размера».

Когда «группы» установлены, с помощью агрегатных функций MIN, MAX и SUM определяются начальное время, конечное время и общее количество транзакций в каждой «группе» (для SQL Server используется функция CEILING, а не CEIL):

```
1  select ceil(trx_id/5.0) as grp,
2         min(trx_date)    as trx_start,
3         max(trx_date)    as trx_end,
4         sum(trx_cnt)     as total
5    from trx_log
6   group by ceil(trx_id/5.0)
```

Обсуждение

Первый и ключевой для всего решения шаг – сгруппировать строки. Сформировать логические группы можно путем деления уникальных идентификаторов записей на 5 и возвращения наименьшего целого числа, которое больше, чем остаток от деления. Например:

```
select trx_id,
       trx_date,
       trx_cnt,
       trx_id/5.0       as val,
       ceil(trx_id/5.0) as grp
  from trx_log
```

```
TRX_ID TRX_DATE             TRX_CNT    VAL GRP
------ -------------------- ------- ------ ---
```

```
 1 28-JUL-2005 19:03:07      44    .20   1
 2 28-JUL-2005 19:03:08      18    .40   1
 3 28-JUL-2005 19:03:09      23    .60   1
 4 28-JUL-2005 19:03:10      29    .80   1
 5 28-JUL-2005 19:03:11      27   1.00   1
 6 28-JUL-2005 19:03:12      45   1.20   2
 7 28-JUL-2005 19:03:13      45   1.40   2
 8 28-JUL-2005 19:03:14      32   1.60   2
 9 28-JUL-2005 19:03:15      41   1.80   2
10 28-JUL-2005 19:03:16      15   2.00   2
11 28-JUL-2005 19:03:17      24   2.20   3
12 28-JUL-2005 19:03:18      47   2.40   3
13 28-JUL-2005 19:03:19      37   2.60   3
14 28-JUL-2005 19:03:20      48   2.80   3
15 28-JUL-2005 19:03:21      46   3.00   3
16 28-JUL-2005 19:03:22      44   3.20   4
17 28-JUL-2005 19:03:23      36   3.40   4
18 28-JUL-2005 19:03:24      41   3.60   4
19 28-JUL-2005 19:03:25      33   3.80   4
20 28-JUL-2005 19:03:26      19   4.00   4
```

Последний шаг – применить соответствующие агрегатные функции и найти, сколько транзакций произошло в течение каждого пятисекундного интервала, а также время начала и завершения каждой транзакции:

```
select ceil(trx_id/5.0) as grp,
       min(trx_date)    as trx_start,
       max(trx_date)    as trx_end,
       sum(trx_cnt)     as total
  from trx_log
 group by ceil(trx_id/5.0)
```

```
GRP TRX_START           TRX_END                TOTAL
--- ------------------- ------------------- ----------
  1 28-JUL-2005 19:03:07 28-JUL-2005 19:03:11        141
  2 28-JUL-2005 19:03:12 28-JUL-2005 19:03:16        178
  3 28-JUL-2005 19:03:17 28-JUL-2005 19:03:21        202
  4 28-JUL-2005 19:03:22 28-JUL-2005 19:03:26        173
```

Если ваши данные немного отличаются от рассматриваемых (скажем, строки не имеют ID), всегда можно создать подобные группы путем деления секунд из значений TRX_DATE на 5. Затем для каждого значения TRX_DATE включаем часы и группируем по фактическому часу и логической «группе», GRP. Ниже представлен пример реализации такой техники (фигурирующие здесь функции Oracle TO_CHAR и TO_NUMBER необходимо заменить на функции работы с датами и форматирования символов, соответствующие используемой платформе):

```
select trx_date,trx_cnt,
       to_number(to_char(trx_date,'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60,'miss'))/5.0) grp
  from trx_log
```

```
TRX_DATE                TRX_CNT         HR         GRP
-------------------- ---------- ---------- ----------
28-JUL-2005 19:03:07         44         19         62
28-JUL-2005 19:03:08         18         19         62
28-JUL-2005 19:03:09         23         19         62
28-JUL-2005 19:03:10         29         19         62
28-JUL-2005 19:03:11         27         19         62
28-JUL-2005 19:03:12         45         19         63
28-JUL-2005 19:03:13         45         19         63
28-JUL-2005 19:03:14         32         19         63
28-JUL-2005 19:03:15         41         19         63
28-JUL-2005 19:03:16         15         19         63
28-JUL-2005 19:03:17         24         19         64
28-JUL-2005 19:03:18         47         19         64
28-JUL-2005 19:03:19         37         19         64
28-JUL-2005 19:03:20         48         19         64
28-JUL-2005 19:03:21         46         19         64
28-JUL-2005 19:03:22         44         19         65
28-JUL-2005 19:03:23         36         19         65
28-JUL-2005 19:03:24         41         19         65
28-JUL-2005 19:03:25         33         19         65
28-JUL-2005 19:03:26         19         19         65
```

Суть здесь в том, что группировка осуществляется для каждых 5 секунд независимо от фактических значений GRP. После этого уже можно применять агрегатные функции, так же как и в исходном решении:

```
select hr,grp,sum(trx_cnt) total
  from (
select trx_date,trx_cnt,
       to_number(to_char(trx_date,'hh24')) hr,
       ceil(to_number(to_char(trx_date-1/24/60/60,'miss'))/5.0) grp
  from trx_log
       ) x
 group by hr,grp
```

```
HR        GRP      TOTAL
-- ---------- ----------
19         62        141
19         63        178
19         64        202
19         65        173
```

Группировать транзакции по часу можно, если журнал транзакций охватывает большие промежутки времени. В DB2 и Oracle такой же результат можно получить с помощью оконной функции SUM OVER. Следующий запрос возвращает все строки таблицы TRX_LOG и, логически их группируя, вычисляет промежуточные суммы (столбец TRX_CNT) и общую сумму транзакций TOTAL (столбец TRX_CNT) в каждой строке «группы»:

```
select trx_id, trx_date, trx_cnt,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)
```

```
                        order by trx_date
                        range between unbounded preceding
                          and current row) runing_total,
       sum(trx_cnt)over(partition by ceil(trx_id/5.0)) total,
       case when mod(trx_id,5.0) = 0 then 'X' end grp_end
  from trx_log
```

```
TRX_ID TRX_DATE              TRX_CNT RUNING_TOTAL      TOTAL GRP_END
------ -------------------- ---------- ------------ ---------- -------
     1 28-JUL-2005 19:03:07         44           44        141
     2 28-JUL-2005 19:03:08         18           62        141
     3 28-JUL-2005 19:03:09         23           85        141
     4 28-JUL-2005 19:03:10         29          114        141
     5 28-JUL-2005 19:03:11         27          141        141 X
     6 28-JUL-2005 19:03:12         45           45        178
     7 28-JUL-2005 19:03:13         45           90        178
     8 28-JUL-2005 19:03:14         32          122        178
     9 28-JUL-2005 19:03:15         41          163        178
    10 28-JUL-2005 19:03:16         15          178        178 X
    11 28-JUL-2005 19:03:17         24           24        202
    12 28-JUL-2005 19:03:18         47           71        202
    13 28-JUL-2005 19:03:19         37          108        202
    14 28-JUL-2005 19:03:20         48          156        202
    15 28-JUL-2005 19:03:21         46          202        202 X
    16 28-JUL-2005 19:03:22         44           44        173
    17 28-JUL-2005 19:03:23         36           80        173
    18 28-JUL-2005 19:03:24         41          121        173
    19 28-JUL-2005 19:03:25         33          154        173
    20 28-JUL-2005 19:03:26         19          173        173 X
```

Агрегация разных групп/сегментов одновременно

Задача

Требуется осуществить агрегацию «в разных измерениях» одновременно. Например, необходимо получить результирующее множество, в котором для каждого сотрудника перечислены имя, отдел, количество служащих в отделе (включая его самого), количество служащих, занимающих ту же должность, что и он (также включая его самого), и общее число служащих в таблице EMP. Таким образом, результирующее множество должно иметь следующий вид:

```
ENAME  DEPTNO DEPTNO_CNT JOB       JOB_CNT  TOTAL
------ ------ ---------- --------- -------- ------
MILLER     10          3 CLERK            4     14
CLARK      10          3 MANAGER          3     14
KING       10          3 PRESIDENT        1     14
SCOTT      20          5 ANALYST          2     14
FORD       20          5 ANALYST          2     14
SMITH      20          5 CLERK            4     14
JONES      20          5 MANAGER          3     14
ADAMS      20          5 CLERK            4     14
```

```
JAMES      30          6 CLERK           4     14
MARTIN     30          6 SALESMAN        4     14
TURNER     30          6 SALESMAN        4     14
WARD       30          6 SALESMAN        4     14
ALLEN      30          6 SALESMAN        4     14
BLAKE      30          6 MANAGER         3     14
```

Решение

Оконные функции упрощают решение этой задачи. Если в вашем распоряжении нет оконных функций, можно использовать скалярные подзапросы.

DB2, Oracle и SQL Server

Используйте оконную функцию COUNT OVER, задавая разные *сегменты* или группы данных, для которых проводится агрегация:

```
select ename,
       deptno,
       count(*)over(partition by deptno) deptno_cnt,
       job,
       count(*)over(partition by job) job_cnt,
       count(*)over() total
  from emp
```

PostgreSQL и MySQL

Для выполнения операций агрегации COUNT разных групп строк используйте скалярные подзапросы в списке оператора SELECT:

```
1  select e.ename,
2         e.deptno,
3         (select count(*) from emp d
4           where d.deptno = e.deptno) as deptno_cnt,
5         job,
6         (select count(*) from emp d
7           where d.job = e.job) as job_cnt,
8         (select count(*) from emp) as total
9    from emp e
```

Обсуждение

DB2, Oracle и SQL Server

Данный пример действительно показывает мощь и преимущества оконных функций. Всего лишь задавая различные сегменты или группы данных, подлежащих агрегации, можно создавать чрезвычайно подробные отчеты без бесконечных рефлексивных объединений и без громоздких и, возможно, низкопроизводительных подзапросов в списке SELECT. Всю работу выполняет оконная функция COUNT OVER. Чтобы понять полученный результат, остановимся на операторе OVER каждой операции COUNT:

```
count(*)over(partition by deptno)
count(*)over(partition by job)
count(*)over()
```

Вспомним основные части оператора OVER: сегмент, определяемый ключевым словом PARTITION BY, и кадр или окно данных, определяемое ORDER BY. Посмотрим на первый оператор COUNT, в котором задано сегментирование по DEPTNO. Строки таблицы EMP будут сгруппированы по DEPTNO, и операция COUNT будет выполнена над всеми строками каждой группы. Поскольку кадр или окно данных не определено (нет оператора ORDER BY), пересчитываются все строки группы. Оператор PARTITION BY находит все уникальные значения DEPTNO, для каждого из них функция COUNT подсчитывает количество строк, имеющих это значение. В конкретном примере COUNT(*)OVER(PARTITION BY DEPTNO) оператор PARTITION BY выделяет сегменты или группы по значениям 10, 20 и 30.

То же самое происходит для второй функции COUNT с сегментированием по JOB. В последней COUNT сегменты не определены, просто указаны пустые круглые скобки. Пустые круглые скобки подразумевают «всю таблицу». Таким образом, тогда как две предыдущие операции COUNT обрабатывают заданные группы или сегменты данных, последняя COUNT подсчитывает все строки таблицы EMP.

Не забывайте, что оконные функции выполняются после предиката WHERE. Если бы вы применили к результирующему множеству некоторый фильтр, например исключающий всех служащих 10-го отдела (DEPTNO 10), значение TOTAL было бы не 14, а 11. Чтобы фильтровать результаты после выполнения оконных функций, необходимо поместить запрос с функцией во вложенный запрос и затем фильтровать результаты, возвращенные этим запросом.

PostgreSQL и MySQL

Для проведения различных подсчетов для каждого отдела и должности используйте несколько скалярных подзапросов в списке SELECT, обрабатывая каждую строку, возвращаемую основным запросом (строки из EMP E). Чтобы получить значение TOTAL, просто с помощью другого скалярного подзапроса пересчитайте всех служащих таблицы EMP.

Агрегация скользящего множества значений

Задача

Требуется выполнить скользящую агрегацию, например найти скользящую сумму заработных плат таблицы EMP. Будем вычислять сумму для каждого интервала в 90 дней, начиная с даты приема на работу (HIREDATE) первого служащего, чтобы увидеть динамику изменения

расходов для каждого 90-дневного периода между датами приема на работу первого и последнего служащих. Должно быть получено следующее результирующее множество:

```
HIREDATE        SAL SPENDING_PATTERN
----------- ------- ----------------
17-DEC-1980     800              800
20-FEB-1981    1600             2400
22-FEB-1981    1250             3650
02-APR-1981    2975             5825
01-MAY-1981    2850             8675
09-JUN-1981    2450             8275
08-SEP-1981    1500             1500
28-SEP-1981    1250             2750
17-NOV-1981    5000             7750
03-DEC-1981     950            11700
03-DEC-1981    3000            11700
23-JAN-1982    1300            10250
09-DEC-1982    3000             3000
12-JAN-1983    1100             4100
```

Решение

Возможность задавать скользящее окно в операторе сегментирования оконных функций сильно упрощает решение этой задачи, если используемая СУБД поддерживает такие функции. Ключ к решению – выполнить упорядочение по HIREDATE в оконной функции и затем задать окно в 90 дней, начиная с даты приема на работу первого служащего. В сумму войдут заработные платы служащих, принятых на работу в течение 90 дней до даты HIREDATE текущего служащего (зарплата текущего служащего включается в сумму). Если в распоряжении нет оконных функций, можно воспользоваться скалярными подзапросами, но тогда решение будет более сложным.

DB2 и Oracle

Для DB2 и Oracle используйте оконную функцию SUM OVER и сортировку по HIREDATE. В операторе сегментирования задайте диапазон 90 дней, чтобы в сумму были включены заработные платы всех служащих, принятых на работу в течение предыдущих 90 дней. Поскольку DB2 не позволяет задавать HIREDATE в операторе ORDER BY оконной функции (строка 3 в фрагменте кода ниже), можно сортировать по DAYS(HIREDATE):

```
1  select hiredate,
2         sal,
3         sum(sal)over(order by days(hiredate)
4                           range between 90 preceding
5                             and current row) spending_pattern
6    from emp e
```

Решение для Oracle более понятное, чем для DB2, потому что в Oracle в оконных функциях можно проводить сортировку по типам даты-времени:

```
select hiredate,
       sal,
       sum(sal)over(order by hiredate
                      range between 90 preceding
                        and current row) spending_pattern
  from emp e
```

MySQL, PostgreSQL и SQL Server

Чтобы для каждого служащего просуммировать заработные платы сотрудников, принятых на работу в течение 90 дней до дня найма рассматриваемого сотрудника, используйте скалярный подзапрос:

```
select e.hiredate,
       e.sal,
       (select sum(sal) from emp d
         where d.hiredate between e.hiredate-90
                              and e.hiredate) as spending_pattern
  from emp e
 order by 1
```

Обсуждение

DB2 и Oracle

Для DB2 и Oracle используется одно и то же решение. Единственное небольшое отличие в том, как задается HIREDATE в операторе ORDER BY оконной функции. На момент написания данной книги DB2 не допускает применения значений типа DATE в ORDER BY, если для определения окна данных используется числовое значение. (Например, если задано RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW, сортировка по дате допускается, а если RANGE BETWEEN 90 PRECEDING AND CURRENT ROW – нет.)

Чтобы понять, что делает запрос, представленный в решении, необходимо просто разобраться, как работает оператор сегментирования. Задаваемое окно данных обусловливает упорядочение заработных плат всех служащих по HIREDATE. Затем функция вычисляет сумму. Сумма вычисляется не для всех заработных плат. Происходит следующее:

1. Определяется заработная плата служащего, принятого на работу первым. Поскольку служащих, нанятых раньше него, нет, сумма в данной точке просто равна заработной плате первого служащего.
2. Определяется заработная плата следующего (соответственно HIREDATE) служащего. Она включается в скользящую сумму вместе с заработными платами других сотрудников, принятых на работу в течение 90 дней до дня найма рассматриваемого сотрудника.

Дата HIREDATE первого служащего – 17 декабря 1980 года, а HIREDATE служащего, поступившего следующим, – 20 февраля 1981. Второй служащий был принят на работу раньше, чем через 90 дней после первого служащего, таким образом, скользящая сумма для второго сотрудника – 2400 (1600 + 800). Чтобы не было трудностей с пониманием, откуда берутся значения поля SPENDING_PATTERN, рассмотрим следующий запрос и результирующее множество:

```
select distinct
       dense_rank()over(order by e.hiredate) window,
       e.hiredate current_hiredate,
       d.hiredate hiredate_within_90_days,
       d.sal sals_used_for_sum
  from emp e,
       emp d
 where d.hiredate between e.hiredate-90 and e.hiredate
```

```
WINDOW CURRENT_HIREDATE HIREDATE_WITHIN_90_DAYS SALS_USED_FOR_SUM
------ ---------------- ---------------------- -----------------
     1 17-DEC-1980      17-DEC-1980                           800
     2 20-FEB-1981      17-DEC-1980                           800
     2 20-FEB-1981      20-FEB-1981                          1600
     3 22-FEB-1981      17-DEC-1980                           800
     3 22-FEB-1981      20-FEB-1981                          1600
     3 22-FEB-1981      22-FEB-1981                          1250
     4 02-APR-1981      20-FEB-1981                          1600
     4 02-APR-1981      22-FEB-1981                          1250
     4 02-APR-1981      02-APR-1981                          2975
     5 01-MAY-1981      20-FEB-1981                          1600
     5 01-MAY-1981      22-FEB-1981                          1250
     5 01-MAY-1981      02-APR-1981                          2975
     5 01-MAY-1981      01-MAY-1981                          2850
     6 09-JUN-1981      02-APR-1981                          2975
     6 09-JUN-1981      01-MAY-1981                          2850
     6 09-JUN-1981      09-JUN-1981                          2450
     7 08-SEP-1981      08-SEP-1981                          1500
     8 28-SEP-1981      08-SEP-1981                          1500
     8 28-SEP-1981      28-SEP-1981                          1250
     9 17-NOV-1981      08-SEP-1981                          1500
     9 17-NOV-1981      28-SEP-1981                          1250
     9 17-NOV-1981      17-NOV-1981                          5000
    10 03-DEC-1981      08-SEP-1981                          1500
    10 03-DEC-1981      28-SEP-1981                          1250
    10 03-DEC-1981      17-NOV-1981                          5000
    10 03-DEC-1981      03-DEC-1981                           950
    10 03-DEC-1981      03-DEC-1981                          3000
    11 23-JAN-1982      17-NOV-1981                          5000
    11 23-JAN-1982      03-DEC-1981                           950
    11 23-JAN-1982      03-DEC-1981                          3000
    11 23-JAN-1982      23-JAN-1982                          1300
    12 09-DEC-1982      09-DEC-1982                          3000
```

```
        13 12-JAN-1983       09-DEC-1982                    3000
        13 12-JAN-1983       12-JAN-1983                    1100
```

В каждой сумме участвуют только строки с одинаковым значением поля WINDOW. Возьмем, к примеру, WINDOW 3. При вычислении суммы для этого окна используются заработные платы 800, 1600 и 1250, в сумме дающие 3650. Если взглянуть на окончательное результирующее множество в разделе «Задача», мы увидим, что значение SPENDING_PATTERN для 22 февраля 1981 (WINDOW 3) равно 3650. Убедиться в том, что приведенное выше рефлексивное объединение обеспечивает выбор соответствующих заработных плат для заданных окон, можно, просто просуммировав значения SALS_USED_FOR_SUM и проведя группировку по CURRENT_DATE. Результаты должны быть аналогичны результирующему множеству, представленному в разделе «Задача» (без дублирующейся строки для 3 декабря 1981):

```
select current_hiredate,
       sum(sals_used_for_sum) spending_pattern
  from (
select distinct
       dense_rank()over(order by e.hiredate) window,
       e.hiredate current_hiredate,
       d.hiredate hiredate_within_90_days,
       d.sal sals_used_for_sum
  from emp e,
       emp d
 where d.hiredate between e.hiredate-90 and e.hiredate
       ) x
 group by current_hiredate
```

```
CURRENT_HIREDATE SPENDING_PATTERN
---------------- ----------------
17-DEC-1980                  800
20-FEB-1981                 2400
22-FEB-1981                 3650
02-APR-1981                 5825
01-MAY-1981                 8675
09-JUN-1981                 8275
08-SEP-1981                 1500
28-SEP-1981                 2750
17-NOV-1981                 7750
03-DEC-1981                11700
23-JAN-1982                10250
09-DEC-1982                 3000
12-JAN-1983                 4100
```

MySQL, PostgreSQL и SQL Server

В этом решении сумма заработных плат для каждых 90 дней на основании значений HIREDATE вычисляется с помощью скалярного подзапроса (подойдет и рефлексивное объединение) с агрегатной функцией SUM. Если возникают затруднения с пониманием происходящего,

просто трансформируйте решение в рефлексивное объединение и проверьте, какие строки будут участвовать в вычислениях. Рассмотрим результирующее множество ниже, которое аналогично результату, приведенному в разделе «Решение»:

```
select e.hiredate,
       e.sal,
       sum(d.sal) as spending_pattern
  from emp e, emp d
 where d.hiredate
       between e.hiredate-90 and e.hiredate
 group by e.hiredate,e.sal
 order by 1
```

```
HIREDATE      SAL SPENDING_PATTERN
----------- ----- ----------------
17-DEC-1980   800              800
20-FEB-1981  1600             2400
22-FEB-1981  1250             3650
02-APR-1981  2975             5825
01-MAY-1981  2850             8675
09-JUN-1981  2450             8275
08-SEP-1981  1500             1500
28-SEP-1981  1250             2750
17-NOV-1981  5000             7750
03-DEC-1981   950            11700
03-DEC-1981  3000            11700
23-JAN-1982  1300            10250
09-DEC-1982  3000             3000
12-JAN-1983  1100             4100
```

Если до сих пор остаются вопросы, уберите агрегацию и начните с получения декартова произведения. Первый шаг – используя таблицу EMP, создать декартово произведение, чтобы каждое значение HIREDATE можно было сравнивать со всеми другими значениями HIREDATE. (Ниже показан лишь фрагмент результирующего множества, потому что декартово произведение таблицы EMP включает 196 строк (14×14)

```
select e.hiredate,
       e.sal,
       d.sal,
       d.hiredate
  from emp e, emp d
```

```
HIREDATE      SAL   SAL HIREDATE
----------- ----- ----- -----------
17-DEC-1980   800   800 17-DEC-1980
17-DEC-1980   800  1600 20-FEB-1981
17-DEC-1980   800  1250 22-FEB-1981
17-DEC-1980   800  2975 02-APR-1981
17-DEC-1980   800  1250 28-SEP-1981
17-DEC-1980   800  2850 01-MAY-1981
17-DEC-1980   800  2450 09-JUN-1981
```

```
17-DEC-1980   800  3000 09-DEC-1982
17-DEC-1980   800  5000 17-NOV-1981
17-DEC-1980   800  1500 08-SEP-1981
17-DEC-1980   800  1100 12-JAN-1983
17-DEC-1980   800   950 03-DEC-1981
17-DEC-1980   800  3000 03-DEC-1981
17-DEC-1980   800  1300 23-JAN-1982
20-FEB-1981  1600   800 17-DEC-1980
20-FEB-1981  1600  1600 20-FEB-1981
20-FEB-1981  1600  1250 22-FEB-1981
20-FEB-1981  1600  2975 02-APR-1981
20-FEB-1981  1600  1250 28-SEP-1981
20-FEB-1981  1600  2850 01-MAY-1981
20-FEB-1981  1600  2450 09-JUN-1981
20-FEB-1981  1600  3000 09-DEC-1982
20-FEB-1981  1600  5000 17-NOV-1981
20-FEB-1981  1600  1500 08-SEP-1981
20-FEB-1981  1600  1100 12-JAN-1983
20-FEB-1981  1600   950 03-DEC-1981
20-FEB-1981  1600  3000 03-DEC-1981
20-FEB-1981  1600  1300 23-JAN-1982
```

Если проанализировать это результирующее множество, можно заметить, что нет даты HIREDATE, на 90 дней раньше или соответствующей 17 декабря, кроме 17 декабря. Таким образом, сумма для данной строки должна составлять всего 800. Если посмотреть на следующую HIREDATE, 20 февраля, можно увидеть, что только одно значение HIREDATE попадает в 90-дневное окно (предыдущие 90 дней), и это 17 декабря. Если сложить значения SAL для 17 декабря и 20 февраля (потому что мы ищем HIREDATE, равные рассматриваемой HIREDATE или попадающие в 90-дневное окно до нее), получаем 2400, что является окончательным результатом для этой даты.

Разобравшись с тем, что происходит, применяем фильтр в предикате WHERE, чтобы получить результаты для каждой HIREDATE и HIREDATE, равной ей или попадающей в 90-дневное окно до этой даты:

```
select e.hiredate,
       e.sal,
       d.sal sal_to_sum,
       d.hiredate within_90_days
  from emp e, emp d
 where d.hiredate
       between e.hiredate-90 and e.hiredate
 order by 1
```

```
HIREDATE      SAL SAL_TO_SUM WITHIN_90_DAYS
----------- ----- ---------- --------------
17-DEC-1980   800        800    17-DEC-1980
20-FEB-1981  1600        800    17-DEC-1980
20-FEB-1981  1600       1600    20-FEB-1981
22-FEB-1981  1250        800    17-DEC-1980
```

```
22-FEB-1981  1250        1600    20-FEB-1981
22-FEB-1981  1250        1250    22-FEB-1981
02-APR-1981  2975        1600    20-FEB-1981
02-APR-1981  2975        1250    22-FEB-1981
02-APR-1981  2975        2975    02-APR-1981
01-MAY-1981  2850        1600    20-FEB-1981
01-MAY-1981  2850        1250    22-FEB-1981
01-MAY-1981  2850        2975    02-APR-1981
01-MAY-1981  2850        2850    01-MAY-1981
09-JUN-1981  2450        2975    02-APR-1981
09-JUN-1981  2450        2850    01-MAY-1981
09-JUN-1981  2450        2450    09-JUN-1981
08-SEP-1981  1500        1500    08-SEP-1981
28-SEP-1981  1250        1500    08-SEP-1981
28-SEP-1981  1250        1250    28-SEP-1981
17-NOV-1981  5000        1500    08-SEP-1981
17-NOV-1981  5000        1250    28-SEP-1981
17-NOV-1981  5000        5000    17-NOV-1981
03-DEC-1981   950        1500    08-SEP-1981
03-DEC-1981   950        1250    28-SEP-1981
03-DEC-1981   950        5000    17-NOV-1981
03-DEC-1981   950         950    03-DEC-1981
03-DEC-1981   950        3000    03-DEC-1981
03-DEC-1981  3000        1500    08-SEP-1981
03-DEC-1981  3000        1250    28-SEP-1981
03-DEC-1981  3000        5000    17-NOV-1981
03-DEC-1981  3000         950    03-DEC-1981
03-DEC-1981  3000        3000    03-DEC-1981
23-JAN-1982  1300        5000    17-NOV-1981
23-JAN-1982  1300         950    03-DEC-1981
23-JAN-1982  1300        3000    03-DEC-1981
23-JAN-1982  1300        1300    23-JAN-1982
09-DEC-1982  3000        3000    09-DEC-1982
12-JAN-1983  1100        3000    09-DEC-1982
12-JAN-1983  1100        1100    12-JAN-1983
```

Теперь, зная, какие значения SAL должны войти в скользящее окно для вычисления суммы, просто применяем агрегатную функцию SUM для получения более выразительного результирующего множества:

```
select e.hiredate,
       e.sal,
       sum(d.sal) as spending_pattern
  from emp e, emp d
 where d.hiredate
       between e.hiredate-90 and e.hiredate
 group by e.hiredate,e.sal
 order by 1
```

Если сравнить результирующее множество приведенного выше запроса и результирующее множество следующего запроса (который является первоначально предлагаемым решением), мы увидим, что они абсолютно одинаковые:

```
select e.hiredate,
       e.sal,
       (select sum(sal) from emp d
         where d.hiredate between e.hiredate-90
                              and e.hiredate) as spending_pattern
  from emp e
 order by 1
```

```
HIREDATE      SAL SPENDING_PATTERN
----------- ----- ----------------
17-DEC-1980   800              800
20-FEB-1981  1600             2400
22-FEB-1981  1250             3650
02-APR-1981  2975             5825
01-MAY-1981  2850             8675
09-JUN-1981  2450             8275
08-SEP-1981  1500             1500
28-SEP-1981  1250             2750
17-NOV-1981  5000             7750
03-DEC-1981   950            11700
03-DEC-1981  3000            11700
23-JAN-1982  1300            10250
09-DEC-1982  3000             3000
12-JAN-1983  1100             4100
```

Разворачивание результирующего множества, содержащего подсуммы

Задача

Требуется вычислить подсуммы, создать отчет и транспонировать его, чтобы обеспечить более наглядный результат. Например, поставлена задача создать отчет, представляющий руководителей каждого отдела и суммы заработных плат подчиненных каждого руководителя. Кроме того, необходимо получить две подсуммы: сумму всех заработных плат по отделам для служащих, работающих в чьем-то подчинении, и сумму всех заработных плат (сумму подсумм отдела). На данный момент имеется следующий отчет:

```
DEPTNO        MGR        SAL
------ ---------- ----------
    10       7782       1300
    10       7839       2450
    10                  3750
    20       7566       6000
    20       7788       1100
    20       7839       2975
    20       7902        800
    20                 10875
    30       7698       6550
    30       7839       2850
```

```
    30                  9400
                       24025
```

Необходимо сделать отчет более удобным для чтения и преобразовать приведенное выше результирующее множество в следующее:

```
MGR     DEPT10     DEPT20     DEPT30      TOTAL
---- ---------- ---------- ---------- ----------
7566          0       6000          0
7698          0          0       6550
7782       1300          0          0
7788          0       1100          0
7839       2450       2975       2850
7902          0        800          0
           3750      10875       9400      24025
```

Решение

Первый шаг – получить подсуммы, используя расширение ROLLUP оператора GROUP BY. Следующий шаг – выполнить классический разворот (с помощью агрегатной функции и выражения CASE) для создания необходимых столбцов отчета. Функция GROUPING обеспечивает возможность без труда определять значения, являющиеся подсуммами (т. е. полученные в результате выполнения ROLLUP). В зависимости от того, как сортируются значения NULL в используемой СУБД, может потребоваться добавить в решение оператор ORDER BY, чтобы получить такое же результирующее множество, как представлено выше.

DB2 и Oracle

Используйте расширение ROLLUP оператора GROUP BY и затем выражение CASE для представления данных в более удобном формате:

```
 1  select mgr,
 2         sum(case deptno when 10 then sal else 0 end) dept10,
 3         sum(case deptno when 20 then sal else 0 end) dept20,
 4         sum(case deptno when 30 then sal else 0 end) dept30,
 5         sum(case flag   when '11' then sal else null end) total
 6    from (
 7  select deptno,mgr,sum(sal) sal,
 8         cast(grouping(deptno) as char(1))||
 9         cast(grouping(mgr)    as char(1)) flag
10    from emp
11   where mgr is not null
12   group by rollup(deptno,mgr)
13         ) x
14   group by mgr
```

SQL Server

Используйте расширение ROLLUP оператора GROUP BY и затем выражение CASE для представления данных в более удобном формате:

```
 1  select mgr,
 2         sum(case deptno when 10 then sal else 0 end) dept10,
 3         sum(case deptno when 20 then sal else 0 end) dept20,
 4         sum(case deptno when 30 then sal else 0 end) dept30,
 5         sum(case flag   when '11' then sal else null end) total
 6    from (
 7  select deptno,mgr,sum(sal) sal,
 8         cast(grouping(deptno) as char(1))+
 9         cast(grouping(mgr)    as char(1)) flag
10    from emp
11   where mgr is not null
12   group by deptno,mgr with rollup
13         ) x
14   group by mgr
```

MySQL и PostgreSQL

Функция GROUPING не поддерживается ни одной из этих СУБД.

Обсуждение

Приведенные выше решения идентичны, за исключением строки конкатенации и описания GROUPING, поэтому промежуточные результаты обсудим на примере решения для SQL Server (все сказанное здесь будет правомочно и для DB2, и для Oracle).

Первый шаг – сформировать результирующее множество, суммируя значения SAL всех подчиненных каждого руководителя (MGR) для каждого отдела (DEPTNO). Идея в том, чтобы показать, сколько служащих подчиняется каждому руководителю в каждом отделе. Например, приведенный ниже запрос позволит сравнить заработные платы подчиненных руководителя KING из 10-го отдела с заработными платами подчиненных KING их 30-го отдела.

```
select deptno,mgr,sum(sal) sal
  from emp
 where mgr is not null
 group by mgr,deptno
 order by 1,2
```

```
DEPTNO        MGR        SAL
------ ---------- ----------
    10       7782       1300
    10       7839       2450
    20       7566       6000
    20       7788       1100
    20       7839       2975
    20       7902        800
    30       7698       6550
    30       7839       2850
```

Далее используем расширение ROLLUP оператора GROUP BY и создаем подсуммы для каждого DEPTNO и по всем служащим (которые находятся в чьем-либо подчинении):

```
select deptno,mgr,sum(sal) sal
  from emp
 where mgr is not null
 group by deptno,mgr with rollup

DEPTNO        MGR        SAL
------ ---------- ----------
    10       7782       1300
    10       7839       2450
    10                  3750
    20       7566       6000
    20       7788       1100
    20       7839       2975
    20       7902        800
    20                 10875
    30       7698       6550
    30       7839       2850
    30                  9400
                       24025
```

Когда подсуммы найдены, необходим способ определить, какое из значений является подсуммой (создано ROLLUP), а какое – результатом выполнения обычного GROUP BY. С помощью функции GROUPING создайте битовые карты, помогающие отличить подсуммы от обычных агрегатов:

```
select deptno,mgr,sum(sal) sal,
       cast(grouping(deptno) as char(1))+
       cast(grouping(mgr)    as char(1)) flag
  from emp
 where mgr is not null
 group by deptno,mgr with rollup

DEPTNO        MGR        SAL FLAG
------ ---------- ---------- ----
    10       7782       1300 00
    10       7839       2450 00
    10                  3750 01
    20       7566       6000 00
    20       7788       1100 00
    20       7839       2975 00
    20       7902        800 00
    20                 10875 01
    30       7698       6550 00
    30       7839       2850 00
    30                  9400 01
                       24025 11
```

Строки со значением 00 в поле FLAG являются результатом обычной агрегации. Строки со значением 01 в поле FLAG – результаты выполнения ROLLUP, осуществляющего агрегацию SAL по DEPTNO (поскольку DEPTNO указан в списке ROLLUP первым; если изменить порядок, например «GROUP BY MGR, DEPTNO WITH ROLLUP», результаты будут совершенно иными). Строка со значением 11 в поле FLAG – результат выполнения ROLLUP, суммирующего SAL по всем строкам.

Теперь у нас есть все необходимое для создания красивого отчета с помощью простых выражений CASE. Цель – сформировать отчет, представляющий заработные платы подчиненных всех руководителей по отделам. Если в каком-то отделе данному руководителю не подчиняется ни один служащий, должен быть возвращен нуль; в противном случае требуется возвратить сумму всех заработных плат подчиненных этого руководителя в данном отделе. Кроме того, должен быть добавлен столбец TOTAL, представляющий сумму всех заработных плат. Решение, удовлетворяющее всем этим требованиям, показано ниже:

```
select mgr,
       sum(case deptno when 10 then sal else 0 end) dept10,
       sum(case deptno when 20 then sal else 0 end) dept20,
       sum(case deptno when 30 then sal else 0 end) dept30,
       sum(case flag   when '11' then sal else null end) total
  from (
select deptno,mgr,sum(sal) sal,
       cast(grouping(deptno) as char(1))+
       cast(grouping(mgr)    as char(1)) flag
  from emp
 where mgr is not null
 group by deptno,mgr with rollup
       ) x
 group by mgr
 order by coalesce(mgr,9999)
```

```
MGR     DEPT10     DEPT20     DEPT30      TOTAL
---- ---------- ---------- ---------- ----------
7566          0       6000          0
7698          0          0       6550
7782       1300          0          0
7788          0       1100          0
7839       2450       2975       2850
7902          0        800          0
           3750      10875       9400      24025
```

13

Иерархические запросы

В данной главе приведены рецепты для отображения иерархических отношений, которые могут присутствовать в данных. Обычно хранить иерархические данные легче, чем извлекать и отображать их (в иерархическом виде). Именно в таких задачах особенно сильно ощущается недостаток гибкости SQL и его нерекурсивная природа. При работе с иерархическими запросами абсолютно необходимо использовать предоставляемые для этого вашей СУБД возможности; в противном случае запросы будут неэффективными либо придется создавать замысловатые модели данных. Скорее всего, рекурсивный оператор WITH будет добавлен в следующие версии PostgreSQL, поэтому пользователям PostgreSQL следует обращать внимание на решения для DB2.

В данной главе будут предложены рецепты, которые помогут отобразить иерархическую структуру данных, используя функции, предоставляемые СУБД. Рассмотрим сначала таблицу EMP и взаимосвязь между столбцами EMPNO и MGR:

```
select empno,mgr
  from emp
order by 2
```

```
     EMPNO        MGR
---------- ----------
      7788       7566
      7902       7566
      7499       7698
      7521       7698
      7900       7698
      7844       7698
      7654       7698
      7934       7782
      7876       7788
      7566       7839
```

```
7782        7839
7698        7839
7369        7902
7839
```

Если посмотреть внимательно, можно увидеть, что каждое значение из MGR также присутствует и в EMPNO. Это говорит о том, что каждый руководитель в таблице EMP также является служащим, запись о котором хранится в таблице EMP, а не где-то в другом месте. Отношение между MGR и EMPNO – это отношение родитель-потомок, потому что значение MGR является непосредственным родителем данного EMPNO (возможно также, что руководитель, в свою очередь, является подчиненным руководителя, который работает под чьим-то руководством, и т. д.; в результате создается *n*-уровневая иерархия). Если у служащего нет руководителя, для него в поле MGR содержится значение NULL.

Представление отношений родитель–потомок

Задача

При выводе записей потомков требуется включить и информацию о родителях. Например, необходимо представить имя каждого сотрудника, а также имя его руководителя. Должно быть получено следующее результирующее множество:

```
EMPS_AND_MGRS
------------------------------
FORD works for JONES
SCOTT works for JONES
JAMES works for BLAKE
TURNER works for BLAKE
MARTIN works for BLAKE
WARD works for BLAKE
ALLEN works for BLAKE
MILLER works for CLARK
ADAMS works for SCOTT
CLARK works for KING
BLAKE works for KING
JONES works for KING
SMITH works for FORD
```

Решение

Выполнить рефлексивное объединение EMP по MGR и EMPNO, чтобы найти имя руководителя для каждого служащего. Затем с помощью предоставляемых СУБД функций, реализующих конкатенацию, сформировать строки и организовать требуемое результирующее множество.

DB2, Oracle и PostgreSQL

Проведите рефлексивное объединение EMP. Затем используйте оператор конкатенации, двойную вертикальную черту (||):

```
1 select a.ename || ' works for ' || b.ename as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

MySQL

Проведите рефлексивное объединение EMP. Затем используйте функцию конкатенации CONCAT:

```
1 select concat(a.ename, ' works for ',b.ename) as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

SQL Server

Проведите рефлексивное объединение EMP. Затем используйте знак плюс (+) как оператор конкатенации:

```
1 select a.ename + ' works for ' + b.ename as emps_and_mgrs
2   from emp a, emp b
3  where a.mgr = b.empno
```

Обсуждение

Реализация всех решений, по сути, одинаковая. Разница лишь в средстве конкатенации строк. Таким образом, будем рассматривать сразу все решения.

Ключ к решению – объединение MGR и EMPNO. Первый шаг – создать декартово произведение, объединяя EMP с самой собой (возвращаемые в результате декартова произведения строки показаны ниже лишь частично):

```
select a.empno, b.empno
  from emp a, emp b

EMPNO        MGR
----- ----------
 7369       7369
 7369       7499
 7369       7521
 7369       7566
 7369       7654
 7369       7698
 7369       7782
 7369       7788
 7369       7839
 7369       7844
 7369       7876
 7369       7900
```

```
7369        7902
7369        7934
7499        7369
7499        7499
7499        7521
7499        7566
7499        7654
7499        7698
7499        7782
7499        7788
7499        7839
7499        7844
7499        7876
7499        7900
7499        7902
7499        7934
```

Как видите, декартово произведение возвращает все возможные сочетания EMPNO/EMPNO (это выглядит так, как будто руководителем служащего с EMPNO 7369 являются все остальные сотрудники, представленные в таблице, включая EMPNO 7369).

Следующий шаг – фильтруем результаты так, чтобы осталось по одной строке для каждого служащего и EMPNO его руководителя. Осуществляем это, проводя объединение по MGR и EMPNO:

```
1  select a.empno, b.empno mgr
2    from emp a, emp b
3   where a.mgr = b.empno

     EMPNO        MGR
---------- ----------
      7902       7566
      7788       7566
      7900       7698
      7844       7698
      7654       7698
      7521       7698
      7499       7698
      7934       7782
      7876       7788
      7782       7839
      7698       7839
      7566       7839
      7369       7902
```

Теперь, когда имеем по одной строке для каждого служащего и EMPNO его руководителя, получить имя руководителя можно, просто выбирая B.ENAME, а не B.EMPNO. Если даже после нескольких попыток возникают трудности с пониманием того, как все это происходит, можно вместо рефлексивного объединения прибегнуть к скалярному подзапросу:

```
select a.ename,
       (select b.ename
          from emp b
         where b.empno = a.mgr) as mgr
  from emp a
```

```
ENAME      MGR
---------- ----------
SMITH      FORD
ALLEN      BLAKE
WARD       BLAKE
JONES      KING
MARTIN     BLAKE
BLAKE      KING
CLARK      KING
SCOTT      JONES
KING
TURNER     BLAKE
ADAMS      SCOTT
JAMES      BLAKE
FORD       JONES
MILLER     CLARK
```

Решение в случае использования скалярного подзапроса эквивалентно решению с рефлексивным объединением, кроме одной строки: в результирующем множестве присутствует строка служащего KING, тогда как в результате рефлексивного объединения ее нет. Вы спросите: «Почему?» Вспомните, значение NULL ничему не равно, даже самому себе. В решении с рефлексивным объединением мы проводим эквиобъединение между EMPNO и MGR, в результате чего отсеиваются все служащие, имеющие значение NULL в поле MGR. Чтобы увидеть строку служащего KING при использовании метода с рефлексивным объединением, необходимо выполнить внешнее объединение, как показано в следующих двух запросах. В первом решении используется ANSI-синтаксис внешнего объединения, тогда как второе решение является примером синтаксиса Oracle. Оба запроса возвращают один и тот же результат, показанный после второго запроса:

```
/* ANSI */
select a.ename, b.ename mgr
  from emp a left join emp b
    on (a.mgr = b.empno)

/* Oracle */
select a.ename, b.ename mgr
  from emp a, emp b
 where a.mgr = b.empno (+)
```

```
ENAME      MGR
---------- ----------
FORD       JONES
SCOTT      JONES
```

```
JAMES      BLAKE
TURNER     BLAKE
MARTIN     BLAKE
WARD       BLAKE
ALLEN      BLAKE
MILLER     CLARK
ADAMS      SCOTT
CLARK      KING
BLAKE      KING
JONES      KING
SMITH      FORD
KING
```

Представление отношений потомок-родитель-прародитель

Задача

Служащий CLARK является подчиненным служащего KING. Представить это отношение можно, используя первый рецепт данной главы. А что если бы служащий CLARK, в свою очередь, был руководителем другого служащего? Рассмотрим следующий запрос:

```
select ename,empno,mgr
  from emp
 where ename in ('KING','CLARK','MILLER')
```

```
ENAME           EMPNO        MGR
---------- ---------- ----------
CLARK            7782       7839
KING             7839
MILLER           7934       7782
```

Как видите, служащий MILLER является подчиненным служащего CLARK, который, в свою очередь, подчиняется KING. Требуется показать всю иерархию от MILLER до KING. Должно быть получено следующее результирующее множество:

```
LEAF___BRANCH___ROOT
--------------------
MILLER-->CLARK-->KING
```

Чтобы показать эти отношения полностью, сверху вниз, одного рефлексивного объединения недостаточно. Можно было бы написать запрос с двумя рефлексивными объединениями, но на самом деле необходим общий подход для представления таких иерархий.

Решение

Этот рецепт отличается от первого тем, что здесь рассматриваются трехуровневые отношения, как предлагает заголовок. Если используемая СУБД не предоставляет функциональности для обхода данных

с древовидной структурой, можно решить эту задачу, используя технику из рецепта «Создание иерархического представления таблицы» с введением дополнительного рефлексивного объединения. DB2, SQL Server и Oracle предлагают функции для работы с иерархиями. Таким образом, рефлексивные объединения в решениях для этих СУБД не нужны, хотя, безусловно, возможны.

DB2 и SQL Server

С помощью рекурсивного оператора WITH найдите руководителя служащего MILLER, CLARK, затем руководителя служащего CLARK, KING. В этом решении используется оператор конкатенации строк SQL Server +:

```
 1    with x (tree,mgr,depth)
 2      as (
 3  select cast(ename as varchar(100)),
 4         mgr, 0
 5    from emp
 6   where ename = 'MILLER'
 7   union all
 8  select cast(x.tree+'-->'+e.ename as varchar(100)),
 9         e.mgr, x.depth+1
10    from emp e, x
11   where x.mgr = e.empno
12  )
13  select tree leaf___branch___root
14    from x
15   where depth = 2
```

Чтобы использовать это решение для DB2, в нем всего лишь надо заменить оператор конкатенации на соответствующий оператор DB2, ||. Но в любом случае, даже без замены оператора, решение будет работать как для DB2, так и для SQL Server.

Oracle

С помощью функции SYS_CONNECT_BY_PATH получите MILLER, его руководителя, CLARK, затем руководителя CLARK, KING. Для обхода дерева используйте оператор CONNECT BY:

```
1  select ltrim(
2           sys_connect_by_path(ename,'-->'),
3         '-->') leaf___branch___root
4    from emp
5   where level = 3
6   start with ename = 'MILLER'
7 connect by prior mgr = empno
```

PostgreSQL и MySQL

Чтобы возвратить служащего MILLER, его руководителя, CLARK, затем руководителя CLARK, KING, дважды выполните рефлексивное

объединение таблицы EMP. В этом решении используется оператор конкатенации PostgreSQL, двойная вертикальная черта (||):

```
1  select a.ename||'-->'||b.ename
2                ||'-->'||c.ename as leaf___branch___root
3    from emp a, emp b, emp c
4   where a.ename = 'MILLER'
5     and a.mgr = b.empno
6     and b.mgr = c.empno
```

Для MySQL просто используйте функцию CONCAT; это решение будет работать и для PostgreSQL.

Обсуждение

DB2 и SQL Server

Здесь обход дерева иерархии начинается с концевого узла и выполняется вверх до корневого узла (для тренировки попробуйте обойти дерево в обратном направлении). Верхняя часть оператора UNION ALL просто ищет строку служащего MILLER (концевой узел). В нижней части UNION ALL идет поиск служащего, который является руководителем MILLER, затем руководителя этого служащего и т. д. Процесс поиска «руководителя руководителя» продолжается до тех пор, пока не будет найден руководитель самого высокого уровня (корневой узел). В рекурсивных запросах DB2 использует поле DEPTH, представляющее уровень иерархии. DEPTH начинается со значения 0 и затем автоматически увеличивается на 1 при каждом выявлении следующего руководителя.

Интересный и всеобъемлющий рассказ об операторе WITH с упором на его рекурсивное использование можно найти в статье Джонатана Генника «Understanding the WITH Clause» по адресу *http://gennick.com/with.htm*.

Далее второй запрос UNION ALL объединяет рекурсивное представление X с таблицей EMP для определения отношений родитель-потомок. На данный момент имеем следующий запрос, использующий оператор конкатенации SQL Server:

```
  with x (tree,mgr,depth)
    as (
select cast(ename as varchar(100)),
       mgr, 0
  from emp
 where ename = 'MILLER'
 union all
select cast(e.ename as varchar(100)),
       e.mgr, x.depth+1
  from emp e, x
 where x.mgr = e.empno
```

```
)
select tree leaf___branch___root, depth
  from x
```

```
TREE            DEPTH
---------- ----------
MILLER              0
CLARK               1
KING                2
```

Итак, основная часть задачи выполнена: получена вся иерархия взаимоотношений снизу вверх, начиная с MILLER. Осталось только отформатировать. Поскольку обход дерева выполняется рекурсивно, просто соединяем текущее значение ENAME из EMP с предыдущим и получаем такое результирующее множество:

```
  with x (tree,mgr,depth)
    as (
select cast(ename as varchar(100)),
       mgr, 0
  from emp
 where ename = 'MILLER'
 union all
select cast(x.tree+'-->'+e.ename as varchar(100)),
       e.mgr, x.depth+1
  from emp e, x
 where x.mgr = e.empno
)
select depth, tree
  from x
```

```
DEPTH TREE
----- ----------------------
    0 MILLER
    1 MILLER-->CLARK
    2 MILLER-->CLARK-->KING
```

Заключительный шаг – выбрать только последнюю строку иерархии. Это можно сделать по-разному, но в решении для определения корневого узла используется значение DEPTH (очевидно, что если бы руководителем CLARK был не KING, фильтр по DEPTH пришлось бы изменить; более универсальное решение, не требующее применения такого фильтра, представлено в следующем рецепте).

Oracle

В решении для Oracle всю работу выполняет оператор CONNECT BY. Начиная с MILLER, проходим весь путь до KING без всяких объединений. Выражение в операторе CONNECT BY определяет отношения между данными и то, как будет выполняться обход дерева:

```
select ename
  from emp
```

```
  start with ename = 'MILLER'
connect by prior mgr = empno

ENAME
----------
MILLER
CLARK
KING
```

Ключевое слово PRIOR обеспечивает возможность доступа к значениям предыдущей записи иерархии. Таким образом, для любого данного EMPNO с помощью PRIOR MGR можно обратиться к номеру руководителя этого служащего. Конструкцию CONNECT BY PRIOR MGR = EMPNO можно рассматривать как представляющую объединение между, в данном случае, родителем и потомком.

Более подробную информацию об операторе CONNECT BY и связанных с ним возможностях можно найти в статьях Oracle Technology Network «Querying Hierarchies: Top-of-the-Line Support» по адресу *http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/gennick_connectby.html* и «New CONNECT BY Features in Oracle Database 10*g*» по адресу *http://www.oracle.com/technology/oramag/webcolumns/2003/techarticles/gennick_connectby_ 10g.html*.

На данный момент мы успешно отобразили всю иерархию, начиная с MILLER и заканчивая KING. Задача по большей части решена. Осталось только отформатировать. С помощью функции SYS_CONNECT_BY_PATH добавьте каждое значение ENAME в конец предшествующего ему значению:

```
 select sys_connect_by_path(ename,'-->') tree
   from emp
  start with ename = 'MILLER'
connect by prior mgr = empno

TREE
---------------------------
-->MILLER
-->MILLER-->CLARK
-->MILLER-->CLARK-->KING
```

Поскольку нам нужна только полная иерархия, фильтруем результаты по псевдостолбцу LEVEL (более универсальный подход показан в следующем рецепте):

```
 select sys_connect_by_path(ename,'-->') tree
   from emp
  where level = 3
  start with ename = 'MILLER'
connect by prior mgr = empno

TREE
```

```
----------------------------
-->MILLER-->CLARK-->KING
```

Заключительный шаг – используем функцию LTRIM для удаления символа «-->» в начале результирующего множества.

PostgreSQL и MySQL

Если СУБД не обеспечивает встроенной поддержки иерархических запросов, для получения всего дерева иерархии приходится выполнять рефлексивные объединения n раз (где n – количество узлов между концевым и корневым узлами, включая корневой узел; в этом примере CLARK относительно MILLER является узлом ветвления, а KING – корневым узлом, таким образом, расстояние – два узла, и $n = 2$). В данном решении просто используется техника из предыдущего рецепта и добавляется еще одно рефлексивное объединение:

```
select a.ename as leaf,
       b.ename as branch,
       c.ename as root
  from emp a, emp b, emp c
 where a.ename = 'MILLER'
   and a.mgr = b.empno
   and b.mgr = c.empno
```

```
LEAF       BRANCH     ROOT
---------- ---------- -----
MILLER     CLARK      KING
```

Следующий и последний шаг – отформатировать вывод, используя оператор конкатенации «||» для PostgreSQL или функцию CONCAT для MySQL. Недостаток такого запроса в том, что при изменении иерархии – например, если появляется еще один узел между CLARK и KING – для получения всего дерева в запрос придется вводить еще одно объединение. Вот почему при работе с иерархиями так удобно пользоваться встроенными функциями СУБД, если они есть.

Создание иерархического представления таблицы

Задача

Требуется получить результирующее множество, описывающее иерархию всей таблицы. Рассмотрим таблицу EMP. В ней у служащего KING нет руководителя, поэтому KING является корневым узлом. Необходимо представить, начиная с KING, всех служащих, подчиняющихся KING, и всех служащих (если таковые имеются), подчиняющихся подчиненным KING. В итоге должно быть получено следующее результирующее множество:

```
EMP_TREE
------------------------------
KING
```

```
KING - BLAKE
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK
KING - CLARK - MILLER
KING - JONES
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

Решение

DB2 и SQL Server

Используя рекурсивный оператор WITH, начинайте построение иерархии с KING и в итоге выведите всех служащих. В представленном далее решении используется оператор конкатенации DB2 «||». Для SQL Server используется оператор конкатенации «+». Во всем остальном решения для обеих СУБД ничем не отличаются:

```
 1  with x (ename,empno)
 2     as (
 3 select cast(ename as varchar(100)),empno
 4   from emp
 5  where mgr is null
 6  union all
 7 select cast(x.ename||' - '||e.ename as varchar(100)),
 8        e.empno
 9   from emp e, x
10  where e.mgr = x.empno
11 )
12 select ename as emp_tree
13   from x
14  order by 1
```

Oracle

Для описания иерархии используйте функцию CONNECT BY. С помощью функции SYS_CONNECT_BY_PATH отформатируйте результаты соответствующим образом:

```
1 select ltrim(
2          sys_connect_by_path(ename,' - '),
3        ' - ') emp_tree
4   from emp
5   start with mgr is null
6 connect by prior empno=mgr
7   order by 1
```

Это решение отличается от приведенного в предыдущем рецепте тем, что не включает фильтр по псевдостолбцу LEVEL. Без этого фильтра выводятся все возможные деревья (где PRIOR EMPNO=MGR).

PostgreSQL

Используйте три оператора UNION и несколько рефлексивных объединений:

```
 1  select emp_tree
 2    from (
 3  select ename as emp_tree
 4    from emp
 5   where mgr is null
 6  union
 7  select a.ename||' - '||b.ename
 8    from emp a
 9         join
10         emp b on (a.empno=b.mgr)
11   where a.mgr is null
12  union
13  select rtrim(a.ename||' - '||b.ename
14                       ||' - '||c.ename,' - ')
15    from emp a
16         join
17         emp b on (a.empno=b.mgr)
18         left join
19         emp c on (b.empno=c.mgr)
20   where a.ename = 'KING'
21  union
22  select rtrim(a.ename||' - '||b.ename||' - '||
23               c.ename||' - '||d.ename,' - ')
24    from emp a
25         join
26         emp b on (a.empno=b.mgr)
27         join
28         emp c on (b.empno=c.mgr)
29         left join
30         emp d on (c.empno=d.mgr)
31   where a.ename = 'KING'
32         ) x
33   where tree is not null
34   order by 1
```

MySQL

Используйте три оператора UNION и несколько рефлексивных объединений:

```
1  select emp_tree
2    from (
3  select ename as emp_tree
4    from emp
```

```
 5   where mgr is null
 6  union
 7  select concat(a.ename,' - ',b.ename)
 8    from emp a
 9         join
10         emp b on (a.empno=b.mgr)
11   where a.mgr is null
12  union
13  select concat(a.ename,' - ',
14                b.ename,' - ',c.ename)
15    from emp a
16         join
17         emp b on (a.empno=b.mgr)
18         left join
19         emp c on (b.empno=c.mgr)
20   where a.ename = 'KING'
21  union
22  select concat(a.ename,' - ',b.ename,' - ',
23                c.ename,' - ',d.ename)
24    from emp a
25         join
26         emp b on (a.empno=b.mgr)
27         join
28         emp c on (b.empno=c.mgr)
29         left join
30         emp d on (c.empno=d.mgr)
31   where a.ename = 'KING'
32         ) x
33   where tree is not null
34   order by 1
```

Обсуждение

DB2 и SQL Server

Первый шаг – определить корневую строку (служащий KING) в рекурсивном представлении X верхней части оператора UNION ALL. Следующий шаг – найти подчиненных KING и их подчиненных, если таковые имеются, через объединение рекурсивного представления X с таблицей EMP. Повторения выполняются до тех пор, пока не будут возвращены все служащие. Результирующее множество, возвращаемое рекурсивным представлением X и отличающееся от окончательного результата только отсутствием форматирования, показано ниже:

```
with x (ename,empno)
    as (
select cast(ename as varchar(100)),empno
  from emp
 where mgr is null
 union all
select cast(e.ename as varchar(100)),e.empno
  from emp e, x
```

```
 where e.mgr = x.empno
)
select ename emp_tree
  from x
```

```
EMP_TREE
-----------------
KING
JONES
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
WARD
MARTIN
TURNER
JAMES
CLARK
MILLER
```

Получены все строки иерархии (которые могут быть полезны), но без форматирования мы не можем сказать, кто из сотрудников является руководителями. Объединяя строку каждого сотрудника со строкой его руководителя, получаем более выразительный результат. Для этого просто используем

```
cast(x.ename+', '+e.ename as varchar(100))
```

в операторе SELECT нижней части UNION ALL в рекурсивном представлении X.

Оператор WITH исключительно полезен при решении такого типа задач, потому что иерархия может меняться (например, концевые узлы могут стать узлами ветвления), но при этом не придется менять запрос.

Oracle

Оператор CONNECT BY возвращает строки иерархической последовательности. Оператор START WITH определяет корневую строку. Если выполнить решение без SYS_CONNECT_BY_PATH, мы получим необходимые строки (что может быть полезным), но без форматирования, отображающего существующие взаимотношения:

```
select ename emp_tree
  from emp
 start with mgr is null
connect by prior empno = mgr
```

```
EMP_TREE
-----------------
KING
JONES
```

```
SCOTT
ADAMS
FORD
SMITH
BLAKE
ALLEN
WARD
MARTIN
TURNER
JAMES
CLARK
MILLER
```

Применяя псевдостолбец LEVEL и функцию LPAD, можно более ясно увидеть иерархию и в конечном счете понять, почему SYS_CONNECT_BY_PATH возвращает результаты, которые и требовалось получить, как показано ранее:

```
select lpad('.',2*level,'.')||ename emp_tree
   from emp
  start with mgr is null
connect by prior empno = mgr
```

```
EMP_TREE
------------------
..KING
....JONES
......SCOTT
........ADAMS
......FORD
........SMITH
....BLAKE
......ALLEN
......WARD
......MARTIN
......TURNER
......JAMES
....CLARK
......MILLER
```

По отступам в данном выводе можно понять, кто является руководителем, а кто – подчиненным. Например, KING не подчиняется никому. JONES подчиняется KING. SCOTT подчиняется JONES. ADAMS подчиняется SCOTT.

Если посмотреть на соответствующие строки решения, использующего SYS_CONNECT_BY_PATH, можно увидеть, что SYS_CONNECT_BY_PATH «накапливает» иерархию. Получая новый узел, мы видим и все предыдущие узлы:

```
KING
KING - JONES
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

Для Oracle 8*i* Database и более ранних версий можно использовать решение для PostgreSQL. Или, поскольку CONNECT BY доступен в более старых версиях Oracle, форматирование можно выполнять с помощью LEVEL и RPAD/LPAD (хотя, чтобы воспроизвести результат, обеспечиваемый SYS_CONNECT_BY_PATH, придется немного потрудиться).

PostgreSQL и MySQL

За исключением конкатенации строк в операторах SELECT, решения для PostgreSQL и MySQL одинаковые. Первый шаг – определить максимальное число узлов для каждой ветки. Это необходимо сделать самостоятельно до создания запроса. Если проанализировать данные таблицы EMP, можно увидеть, что служащие ADAM и SMITH являются концевыми узлами с самым глубоким уровнем вложенности (загляните в раздел обсуждения решения для Oracle, где найдете правильно отформатированное дерево иерархии EMP). Рассмотрим служащего ADAMS. Он является подчиненным SCOTT, который, в свою очередь, подчиняется JONES, который подчиняется KING. Таким образом, глубина равна 4. Чтобы представить иерархию с четырехкратной глубиной вложенности, необходимо выполнить рефлексивное объединение четырех экземпляров таблицы EMP и написать запрос с оператором UNION, состоящим из четырех частей. Результаты четырехкратного рефлексивного объединения (нижняя часть последнего UNION, если рассматривать сверху вниз) показаны ниже (с использованием синтаксиса PostgreSQL; пользователи MySQL просто заменяют оператор «||» вызовом функции CONCAT):

```
select rtrim(a.ename||' - '||b.ename||' - '||
             c.ename||' - '||d.ename,' - ') as max_depth_4
  from emp a
       join
       emp b on (a.empno=b.mgr)
       join
       emp c on (b.empno=c.mgr)
       left join
       emp d on (c.empno=d.mgr)
 where a.ename = 'KING'

MAX_DEPTH_4
-----------------------------
KING - JONES - FORD - SMITH
KING - JONES - SCOTT - ADAMS
KING - BLAKE - TURNER
KING - BLAKE - ALLEN
KING - BLAKE - WARD
KING - CLARK - MILLER
KING - BLAKE - MARTIN
KING - BLAKE - JAMES
```

Фильтр по A.ENAME гарантирует, что корневой строкой является KING и никакая другая строка. Если взглянуть на приведенное выше результирующее множество и сравнить его с конечным результатом, можно заметить отсутствие нескольких строк, представляющих третий уровень иерархии: KING–JONES–FORD и KING–JONES–SCOTT. Чтобы эти строки вошли в результирующее множество, необходимо написать еще один запрос, подобный приведенному выше, но в котором было бы на единицу меньше объединений (рефлексивные объединения всего трех экземпляров таблицы EMP, а не четырех). Результирующее множество этого запроса показано ниже:

```
select rtrim(a.ename||' - '||b.ename
                    ||' - '||c.ename,' - ') as max_depth_3
  from emp a
       join
       emp b on (a.empno=b.mgr)
       left join
       emp c on (b.empno=c.mgr)
 where a.ename = 'KING'

MAX_DEPTH_3
--------------------------
KING - BLAKE - ALLEN
KING - BLAKE - WARD
KING - BLAKE - MARTIN
KING - JONES - SCOTT
KING - BLAKE - TURNER
KING - BLAKE - JAMES
KING - JONES - FORD
KING - CLARK - MILLER
```

Как в предыдущем запросе, здесь фильтр A.ENAME гарантирует, что корневым узлом является KING. Можно заметить, что строки, возвращаемые данным запросом, частично повторяют строки, возвращаемые предыдущим четырехкратным объединением EMP. Чтобы избавиться от лишних строк, просто объединим (с помощью оператора UNION) два запроса:

```
select rtrim(a.ename||' - '||b.ename
                    ||' - '||c.ename,' - ') as partial_tree
  from emp a
       join
       emp b on (a.empno=b.mgr)
       left join
       emp c on (b.empno=c.mgr)
 where a.ename = 'KING'
union
select rtrim(a.ename||' - '||b.ename||' - '||
             c.ename||' - '||d.ename,' - ')
  from emp a
       join
       emp b on (a.empno=b.mgr)
```

```
       join
       emp c on (b.empno=c.mgr)
       left join
       emp d on (c.empno=d.mgr)
 where a.ename = 'KING'
```

```
PARTIAL_TREE
----------------------------
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK - MILLER
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

Сейчас дерево почти готово. Следующий шаг – выбрать строки, представляющие второй уровень иерархии, где KING – корневой узел (т. е. служащих, находящихся в прямом подчинении у KING). Запрос, возвращающий эти строки, показан ниже:

```
select a.ename||' - '||b.ename as max_depth_2
  from emp a
       join
       emp b on (a.empno=b.mgr)
 where a.mgr is null
```

```
MAX_DEPTH_2
---------------
KING - JONES
KING - BLAKE
KING - CLARK
```

Следующий шаг – объединить (с помощью оператора UNION) приведенный выше запрос с объединением PARTIAL_TREE:

```
select a.ename||' - '||b.ename as partial_tree
  from emp a
       join
       emp b on (a.empno=b.mgr)
 where a.mgr is null
union
select rtrim(a.ename||' - '||b.ename
                    ||' - '||c.ename,' - ')
  from emp a
       join
       emp b on (a.empno=b.mgr)
       left join
       emp c on (b.empno=c.mgr)
 where a.ename = 'KING'
union
```

```
select rtrim(a.ename||' - '||b.ename||' - '||
             c.ename||' - '||d.ename,' - ')
  from emp a
       join
       emp b on (a.empno=b.mgr)
       join
       emp c on (b.empno=c.mgr)
       left join
       emp d on (c.empno=d.mgr)
 where a.ename = 'KING'
```

```
PARTIAL_TREE
---------------------------------
KING - BLAKE
KING - BLAKE - ALLEN
KING - BLAKE - JAMES
KING - BLAKE - MARTIN
KING - BLAKE - TURNER
KING - BLAKE - WARD
KING - CLARK
KING - CLARK - MILLER
KING - JONES
KING - JONES - FORD
KING - JONES - FORD - SMITH
KING - JONES - SCOTT
KING - JONES - SCOTT - ADAMS
```

Заключительный шаг – с помощью операции UNION вставить строку служащего KING в верхушку PARTIAL_TREE и получить требуемое результирующее множество.

Выбор всех дочерних строк для заданной строки

Задача

Требуется найти всех служащих, которые прямо или косвенно (т. е. являются подчиненными того, кто подчиняется JONES) подчиняются JONES. Список подчиненных JONES показан ниже (JONES включен в результирующее множество):

```
ENAME
---------
JONES
SCOTT
ADAMS
FORD
SMITH
```

Решение

Здесь очень пригодится возможность переходить в самый верх или самый низ дерева иерархии. Для этого решения не требуется специаль-

ного форматирования. Цель – просто выбрать всех служащих, работающих под руководством JONES, включая самого JONES. Подобные задачи, на самом деле, демонстрируют ценность таких рекурсивных расширений SQL, как операторы CONNECT BY для Oracle и WITH для SQL Server/DB2.

DB2 и SQL Server

С помощью рекурсивного оператора WITH найдите всех подчиненных JONES. Начинайте с JONES, задав WHERE ENAME = 'JONES' в первом из двух объединяемых запросов.

```
 1    with x (ename,empno)
 2      as (
 3  select ename,empno
 4    from emp
 5   where ename = 'JONES'
 6   union all
 7  select e.ename, e.empno
 8    from emp e, x
 9   where x.empno = e.mgr
10  )
11  select ename
12    from x
```

Oracle

Используйте оператор CONNECT BY и задайте START WITH ENAME = 'JONES', чтобы найти всех подчиненных JONES:

```
1  select ename
2    from emp
3   start with ename = 'JONES'
4  connect by prior empno = mgr
```

PostgreSQL и MySQL

Необходимо заранее знать количество узлов в дереве. Следующий запрос показывает, как определить глубину иерархии:

```
/* находим EMPNO служащего JONES */
select ename,empno,mgr
  from emp
 where ename = 'JONES'

ENAME           EMPNO        MGR
---------- ---------- ----------
JONES            7566       7839

/* есть ли служащие, находящиеся в прямом подчинении у JONES? */
select count(*)
  from emp
 where mgr = 7566

  COUNT(*)
```

```
----------
         2

/* у JONES двое подчиненных, найдем их EMPNO */
select ename,empno,mgr
  from emp
 where mgr = 7566

ENAME           EMPNO        MGR
---------- ---------- ----------
SCOTT            7788       7566
FORD             7902       7566

/* есть ли подчиненные у SCOTT или FORD? */
select count(*)
  from emp
 where mgr in (7788,7902)

  COUNT(*)
----------
         2

/* у SCOTT и FORD двое подчиненных, находим их EMPNO */
select ename,empno,mgr
  from emp
 where mgr in (7788,7902)

ENAME           EMPNO        MGR
---------- ---------- ----------
SMITH            7369       7902
ADAMS            7876       7788

/* есть ли подчиненные у SMITH или ADAMS? */
select count(*)
  from emp
 where mgr in (7369,7876)

  COUNT(*)
----------
         0
```

Иерархия, начинающаяся со служащего JONES, заканчивается служащими SMITH и ADAMS, таким образом, имеем трехуровневую иерархию. Теперь, когда известна глубина, можно приступать к обходу иерархии сверху вниз.

Сначала дважды выполним рефлексивное объединение таблицы EMP. Затем произведем обратное разворачивание вложенного представления X, чтобы преобразовать три столбца и две строки в один столбец и шесть строк (в PostgreSQL как альтернативу запросу к сводной таблице T100 можно использовать GENERATE_SERIES(1,6):

```
1  select distinct
2         case t100.id
3              when 1 then root
```

```
 4              when 2 then branch
 5              else        leaf
 6         end as JONES_SUBORDINATES
 7    from (
 8  select a.ename as root,
 9         b.ename as branch,
10         c.ename as leaf
11    from emp a, emp b, emp c
12   where a.ename = 'JONES'
13     and a.empno = b.mgr
14     and b.empno = c.mgr
15         ) x,
16         t100
17   where t100.id <= 6
```

В качестве альтернативы можно использовать представления и объединять (UNION) результаты. Если созданы следующие представления:

```
create view v1
as
select ename,mgr,empno
  from emp
 where ename = 'JONES'

create view v2
as
select ename,mgr,empno
  from emp
 where mgr = (select empno from v1)

create view v3
as
select ename,mgr,empno
  from emp
 where mgr in (select empno from v2)
```

решение будет таким:

```
select ename from v1
 union
select ename from v2
 union
select ename from v3
```

Обсуждение

DB2 и SQL Server

Рекурсивный оператор WITH упрощает решение этой задачи. Первая часть оператора WITH, верхняя часть UNION ALL возвращает строку служащего JONES. Необходимо получить ENAME, чтобы увидеть имя, и EMPNO, чтобы использовать его для объединения. Нижняя часть UNION ALL рекурсивно объединяет EMP.MGR с X.EMPNO. Ус-

ловие объединения будет применяться до тех пор, пока не будет исчерпано результирующее множество.

Oracle

Оператор START WITH указывает запросу сделать JONES корневым узлом. Условие в операторе CONNECT BY управляет обходом дерева, который будет продолжаться до тех пор, пока условие истинно.

PostgreSQL и MySQL

Техника, используемая здесь, аналогична применяемой во втором рецепте данной главы, «Представление отношений потомок-родитель-прародитель». Основной недостаток в том, что заранее должна быть известна глубина иерархии.

Определение узлов: ветвления, концевого, корневого

Задача

Требуется определить, узлом какого типа является данная строка: концевым узлом, узлом ветвления или корневым узлом. В данном примере концевым узлом считается служащий, не являющийся руководителем. Узел ветвления – это служащий, являющийся как руководителем, так и подчиненным. Корневой узел – служащий, не имеющий руководителя. Для отражения статуса каждой строки в иерархии используется 1 (TRUE) или 0 (FALSE). Должно быть получено следующее результирующее множество:

```
ENAME      IS_LEAF  IS_BRANCH    IS_ROOT
---------- ---------- ---------- ----------
KING                0          0          1
JONES               0          1          0
SCOTT               0          1          0
FORD                0          1          0
CLARK               0          1          0
BLAKE               0          1          0
ADAMS               1          0          0
MILLER              1          0          0
JAMES               1          0          0
TURNER              1          0          0
ALLEN               1          0          0
WARD                1          0          0
MARTIN              1          0          0
SMITH               1          0          0
```

Решение

Важно понимать, что таблица EMP смоделирована как древовидная иерархия, не рекурсивная, значение поля MGR корневых узлов – NULL. Если бы в EMP использовалась рекурсивная структура данных, корне-

вые узлы были бы автореферентными (т. е. ссылались бы сами на себя, и значением поля MGR для служащего KING было бы значение поля EMPNO этого же самого KING). Автореферентность показалась мне лишенной здравого смысла, и поэтому в поле MGR корневых узлов я поместил значение NULL. Значительно упростят работу с древовидными иерархиями и сделают их потенциально более эффективными, чем рекурсивные иерархии, операторы CONNECT BY для пользователей Oracle и WITH для пользователей DB2/SQL Server. Применяя CONNECT BY или WITH при работе с рекурсивной иерархией, будьте начеку: все может закончиться циклом. Чтобы этого не произошло, в коде должны быть предприняты соответствующие меры предосторожности.

DB2, PostgreSQL, MySQL и SQL Server

Чтобы верно определить один из трех типов узлов и возвратить соответствующее логическое значение (1 или 0), используйте три скалярных подзапроса:

```
 1  select e.ename,
 2         (select sign(count(*)) from emp d
 3           where 0 =
 4             (select count(*) from emp f
 5               where f.mgr = e.empno)) as is_leaf,
 6         (select sign(count(*)) from emp d
 7           where d.mgr = e.empno
 8            and e.mgr is not null) as is_branch,
 9         (select sign(count(*)) from emp d
10           where d.empno = e.empno
11             and d.mgr is null) as is_root
12    from emp e
13  order by 4 desc,3 desc
```

Oracle

Решение с применением скалярных подзапросов подойдет и для Oracle и должно использоваться для версий до Oracle Database 10*g*. Следующее решение для идентификации корневых и концевых узлов опирается на предлагаемые Oracle встроенные функции (которые были введены в Oracle Database 10*g*). Это функции CONNECT_BY_ROOT и CONNECT_BY_ISLEAF соответственно:

```
 1   select ename,
 2          connect_by_isleaf is_leaf,
 3          (select count(*) from emp e
 4            where e.mgr = emp.empno
 5              and emp.mgr is not null
 6              and rownum = 1) is_branch,
 7          decode(ename,connect_by_root(ename),1,0) is_root
 8     from emp
 9    start with mgr is null
10  connect by prior empno = mgr
11  order by 4 desc, 3 desc
```

Обсуждение

DB2, PostgreSQL, MySQL и SQL Server

Для выявления концевых, корневых узлов и узлов ветвления в решении просто применяются правила, описанные в разделе «Задача». Первый шаг – дать определение тому, какой узел считается концевым узлом. Если служащий не является руководителем (никто не находится в его подчинении), он – концевой узел. Первый скалярный подзапрос, IS_LEAF, показан ниже:

```
select e.ename,
       (select sign(count(*)) from emp d
         where 0 =
           (select count(*) from emp f
             where f.mgr = e.empno)) as is_leaf
  from emp e
order by 2 desc
```

```
ENAME         IS_LEAF
---------- ----------
SMITH              1
ALLEN              1
WARD               1
ADAMS              1
TURNER             1
MARTIN             1
JAMES              1
MILLER             1
JONES              0
BLAKE              0
CLARK              0
FORD               0
SCOTT              0
KING               0
```

Поскольку IS_LEAF должен возвращать 0 или 1, к операции COUNT(*) необходимо применить функцию SIGN. В противном случае для концевых строк мы получим 14, а не 1. В качестве альтернативы для подсчета можно использовать таблицу с одной строкой, ведь нам требуется возвращать только 0 или 1. Например:

```
select e.ename,
       (select count(*) from t1 d
         where not exists
           (select null from emp f
             where f.mgr = e.empno)) as is_leaf
  from emp e
order by 2 desc
```

```
ENAME         IS_LEAF
---------- ----------
SMITH              1
```

```
ALLEN              1
WARD               1
ADAMS              1
TURNER             1
MARTIN             1
JAMES              1
MILLER             1
JONES              0
BLAKE              0
CLARK              0
FORD               0
SCOTT              0
KING               0
```

Следующий шаг – найти узлы ветвления. Если служащий является руководителем (имеет подчиненных), а также находится в подчинении у другого служащего, он является узлом ветвления. Результаты скалярного подзапроса IS_BRANCH показаны ниже:

```
select e.ename,
       (select sign(count(*)) from emp d
         where d.mgr = e.empno
          and e.mgr is not null) as is_branch
  from emp e
order by 2 desc
```

```
ENAME       IS_BRANCH
---------- ----------
JONES               1
BLAKE               1
SCOTT               1
CLARK               1
FORD                1
SMITH               0
TURNER              0
MILLER              0
JAMES               0
ADAMS               0
KING                0
ALLEN               0
MARTIN              0
WARD                0
```

Опять же необходимо применить функцию SIGN к операции COUNT(*). В противном случае для узлов ветвления будут получены (потенциально) значения больше 1. Как и в скалярном подзапросе IS_LEAF, избежать SIGN можно с помощью таблицы в одну строку. В следующем решении используется такая таблица под названием dual:[1]

[1] Таблица dual – стандартная таблица для Oracle; во всех остальных случаях придется создавать таблицу, которая будет содержать одну строку; поэтому в примере указана t1, а не dual. – *Примеч. науч. ред.*

```
select e.ename,
       (select count(*) from t1 t
         where exists (
          select null from emp f
           where f.mgr = e.empno
             and e.mgr is not null)) as is_branch
  from emp e
order by 2 desc
```

```
ENAME       IS_BRANCH
---------- ----------
JONES               1
BLAKE               1
SCOTT               1
CLARK               1
FORD                1
SMITH               0
TURNER              0
MILLER              0
JAMES               0
ADAMS               0
KING                0
ALLEN               0
MARTIN              0
WARD                0
```

Последний шаг – найти корневые узлы. Корневой узел определен как служащий, являющийся руководителем, но не находящийся ни в чьем подчинении. В таблице EMP только служащий KING удовлетворяет этому условию и является корневым узлом. Скалярный подзапрос IS_ROOT показан ниже:

```
select e.ename,
       (select sign(count(*)) from emp d
         where d.empno = e.empno
           and d.mgr is null) as is_root
  from emp e
order by 2 desc
```

```
ENAME         IS_ROOT
---------- ----------
KING                1
SMITH               0
ALLEN               0
WARD                0
JONES               0
TURNER              0
JAMES               0
MILLER              0
FORD                0
ADAMS               0
MARTIN              0
```

```
BLAKE              0
CLARK              0
SCOTT              0
```

EMP – маленькая таблица в 14 строк, поэтому легко увидеть, что служащий KING является единственным корневым узлом, и не обязательно применять функцию SIGN к операции COUNT(*). Но если корневых узлов может быть несколько, тогда в скалярном подзапросе необходимо использовать или SIGN, или таблицу в одну строку, как показано ранее для подзапросов IS_BRANCH и IS_ LEAF.

Oracle

Те, кто использует более ранние версии Oracle (до Oracle Database 10*g*), могут рассматривать решение для других СУБД, потому что оно подойдет (без всяких изменений) и для Oracle. При работе с Oracle Database 10*g* и более поздними версиями есть возможность воспользоваться преимуществами двух функций, которые сильно упрощают задачу по идентификации корневых и концевых узлов: это функции CONNECT_BY_ROOT и CONNECT_BY_ISLEAF соответственно. На момент написания данной книги, чтобы использовать CONNECT_BY_ROOT и CONNECT_BY_ISLEAF, в выражении SQL должен присутствовать оператор CONNECT BY. Первый шаг – найти концевые узлы, применяя CONNECT_BY_ISLEAF следующим образом:

```
select ename,
       connect_by_isleaf is_leaf
  from emp
 start with mgr is null
connect by prior empno = mgr
order by 2 desc
```

```
ENAME         IS_LEAF
---------- ----------
ADAMS              1
SMITH              1
ALLEN              1
TURNER             1
MARTIN             1
WARD               1
JAMES              1
MILLER             1
KING               0
JONES              0
BLAKE              0
CLARK              0
FORD               0
SCOTT              0
```

Далее с помощью скалярного подзапроса ищем узлы ветвления. Узлы ветвления – это служащие, являющиеся руководителями и подчиненными одновременно:

```
select ename,
       (select count(*) from emp e
         where e.mgr = emp.empno
           and emp.mgr is not null
           and rownum = 1) is_branch
  from emp
 start with mgr is null
connect by prior empno = mgr
order by 2 desc
```

```
ENAME      IS_BRANCH
---------- ----------
JONES               1
SCOTT               1
BLAKE               1
FORD                1
CLARK               1
KING                0
MARTIN              0
MILLER              0
JAMES               0
TURNER              0
WARD                0
ADAMS               0
ALLEN               0
SMITH               0
```

Фильтр по ROWNUM обязателен и гарантирует возвращение 1 или 0 и ничего другого.

Последний шаг – с помощью функции CONNECT_BY_ROOT найти корневые узлы. Решение находит ENAME корневого узла и сравнивает его со всеми строками, возвращенными запросом. Если соответствий не обнаружено, эта строка является корневым узлом:

```
select ename,
       decode(ename,connect_by_root(ename),1,0) is_root
  from emp
 start with mgr is null
connect by prior empno = mgr
order by 2 desc
```

```
ENAME        IS_ROOT
---------- ----------
KING                1
JONES               0
SCOTT               0
ADAMS               0
FORD                0
SMITH               0
BLAKE               0
ALLEN               0
WARD                0
```

```
MARTIN              0
TURNER              0
JAMES               0
CLARK               0
MILLER              0
```

Для Oracle 9*i* Database и более поздних версий в качестве альтернативы CONNECT_BY_ROOT можно использовать функцию SYS_CONNECT_BY_PATH. Вот версия предыдущего решения для Oracle 9*i* Database:

```
select ename,
       decode(substr(root,1,instr(root,',')-1),NULL,1,0) root
  from (
select ename,
       ltrim(sys_connect_by_path(ename,','),',') root
  from emp
start with mgr is null
connect by prior empno=mgr
       )
```

```
ENAME      ROOT
---------- ----
KING          1
JONES         0
SCOTT         0
ADAMS         0
FORD          0
SMITH         0
BLAKE         0
ALLEN         0
WARD          0
MARTIN        0
TURNER        0
JAMES         0
CLARK         0
MILLER        0
```

Функция SYS_CONNECT_BY_PATH составляет иерархию, начиная с корневого значения, как показано ниже:

```
select ename,
       ltrim(sys_connect_by_path(ename,','),',') path
  from emp
start with mgr is null
connect by prior empno=mgr
```

```
ENAME      PATH
---------- ----------------------------
KING       KING
JONES      KING,JONES
SCOTT      KING,JONES,SCOTT
ADAMS      KING,JONES,SCOTT,ADAMS
FORD       KING,JONES,FORD
```

```
SMITH      KING,JONES,FORD,SMITH
BLAKE      KING,BLAKE
ALLEN      KING,BLAKE,ALLEN
WARD       KING,BLAKE,WARD
MARTIN     KING,BLAKE,MARTIN
TURNER     KING,BLAKE,TURNER
JAMES      KING,BLAKE,JAMES
CLARK      KING,CLARK
MILLER     KING,CLARK,MILLER
```

Чтобы получить корневую строку, просто с помощью функции SUBSTR извлеките подстроку первого значения ENAME в PATH:

```
select ename,
       substr(root,1,instr(root,',')-1) root
  from (
select ename,
       ltrim(sys_connect_by_path(ename,','),',') root
  from emp
start with mgr is null
connect by prior empno=mgr
       )
```

```
ENAME      ROOT
---------- ---------
KING
JONES      KING
SCOTT      KING
ADAMS      KING
FORD       KING
SMITH      KING
BLAKE      KING
ALLEN      KING
WARD       KING
MARTIN     KING
TURNER     KING
JAMES      KING
CLARK      KING
MILLER     KING
```

Последний шаг – отметить флагом строку, в поле ROOT которой содержится значение NULL; это и есть корневая строка.

14

Всякая всячина

В данной главе собраны запросы, по тем или иным причинам не вошедшие в остальные части книги: или потому что глава, в которую они могли бы быть включены, и так слишком обширна, или потому что они представляют собой скорее просто красивые технические решения, чем решения практических задач. Это глава «для забавы», ее рецептами вы можете никогда в жизни не воспользоваться. Тем не менее мне они показались интересными, и я решил включить их в книгу.

Создание отчетов с перекрестными ссылками с помощью оператора SQL Server PIVOT

Задача

Требуется создать отчет с перекрестными ссылками, преобразовать строки результирующего множества в столбцы. Традиционные методы разворачивания нам известны, но хотелось бы попробовать что-то другое, в частности, получить следующее результирующее множество без применения выражений CASE или объединений:

```
DEPT_10    DEPT_20    DEPT_30    DEPT_40
------- ---------- ---------- ----------
      3          5          6          0
```

Решение

Чтобы создать требуемое результирующее множество, не прибегая к выражениям CASE или дополнительным объединениям, используйте оператор PIVOT:

```
1  select [10] as dept_10,
2         [20] as dept_20,
```

```
3         [30] as dept_30,
4         [40] as dept_40
5    from (select deptno, empno from emp) driver
6   pivot (
7      count(driver.empno)
8      for driver.deptno in ( [10],[20],[30],[40] )
9   ) as empPivot
```

Обсуждение

Оператор PIVOT может показаться на первый взгляд странным, но операция, которую он осуществляет, аналогична тому, что делает более привычный транспонирующий запрос, показанный ниже:

```
select sum(case deptno when 10 then 1 else 0 end) as dept_10,
       sum(case deptno when 20 then 1 else 0 end) as dept_20,
       sum(case deptno when 30 then 1 else 0 end) as dept_30,
       sum(case deptno when 40 then 1 else 0 end) as dept_40
  from emp

DEPT_10    DEPT_20    DEPT_30    DEPT_40
------- ---------- ---------- ----------
      3          5          6          0
```

Теперь, зная, что происходит по существу, разложим действия оператора PIVOT на составляющие. В строке 5 решения показан вложенный запрос DRIVER:

```
from (select deptno, empno from emp) driver
```

Псевдоним «driver» выбран потому, что строки, возвращаемые этим вложенным запросом (или табличным выражением), напрямую поступают в операцию PIVOT. Оператор PIVOT поворачивает и превращает строки в столбцы, обрабатывая элементы, перечисленные в строке 8 в списке FOR (показан ниже):

```
for driver.deptno in ( [10],[20],[30],[40] )
```

Обработка происходит примерно следующим образом:

1. Выполняем операцию агрегации COUNT(DRIVER.EMPNO) для строк, значение поля DEPTNO которых равно 10.
2. Повторяем то же самое для строк с DEPTNO, равными 20, 30 и 40.

Элементы, перечисленные в квадратных скобках в строке 8, не только определяют значения, участвующие в агрегации, они также используются как имена столбцов результирующего множества (без скобок). Ссылка на элементы списка FOR и присвоение им псевдонимов осуществляется в операторе SELECT решения. Если псевдонимы не заданы, в списке FOR используются имена столбцов без квадратных скобок.

Довольно любопытно, что, поскольку DRIVER – это просто вложенный запрос, его можно сделать более сложным. Например, рассмотрим ситуацию, когда требуется изменить результирующее множество

таким образом, чтобы в качестве имени столбца использовалось реальное название отдела. Ниже приведены строки таблицы DEPT:

```
select * from dept

DEPTNO DNAME          LOC
------ -------------- -------------
    10 ACCOUNTING     NEW YORK
    20 RESEARCH       DALLAS
    30 SALES          CHICAGO
    40 OPERATIONS     BOSTON
```

Хотелось бы с помощью оператора PIVOT получить следующее результирующее множество:

```
ACCOUNTING   RESEARCH      SALES OPERATIONS
---------- ---------- ---------- ----------
         3          5          6          0
```

Поскольку в качестве вложенного запроса DRIVER может использоваться практически любое корректное табличное выражение, можно выполнить объединение таблицы EMP с таблицей DEPT и затем с помощью PIVOT получить эти строки. В результате выполнения следующего запроса будет получено необходимое результирующее множество:

```
select [ACCOUNTING] as ACCOUNTING,
       [SALES]      as SALES,
       [RESEARCH]   as RESEARCH,
       [OPERATIONS] as OPERATIONS
  from (
          select d.dname, e.empno
            from emp e,dept d
           where e.deptno=d.deptno

        ) driver
  pivot (
   count(driver.empno)
   for driver.dname in ([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
  ) as empPivot
```

Как видите, PIVOT представляет разворачивание результирующих множеств несколько иначе. Независимо от того, предпочтете ли вы эту технику традиционным методам разворачивания, еще один инструмент в профессиональном арсенале не будет лишним.

Обратное разворачивание отчета с помощью оператора SQL Server UNPIVOT

Задача

Имеется результирующее множество, полученное с применением разворачивания (или просто массивная таблица), и требуется нормализовать (выполнить обратное разворачивание) результирующее множест-

во. Например, вместо результирующего множества, состоящего из одной строки и четырех столбцов, необходимо получить результирующее множество с двумя столбцами и четырьмя строками. Используя результирующее множество предыдущего рецепта, мы должны преобразовать такую таблицу:

```
ACCOUNTING   RESEARCH      SALES OPERATIONS
---------- ---------- ---------- ----------
         3          5          6          0
```

в следующую:

```
DNAME                CNT
-------------- ----------
ACCOUNTING             3
RESEARCH               5
SALES                  6
OPERATIONS             0
```

Решение

Ведь вы не думали, что SQL Server может предоставить возможность развернуть таблицу без возможности развернуть ее в обратном направлении? Чтобы провести обратное разворачивание результирующего множества, просто используем его как driver и позволим оператору UNPIVOT выполнить всю работу. Необходимо только задать имена столбцов:

```
 1  select DNAME, CNT
 2    from (
 3      select [ACCOUNTING] as ACCOUNTING,
 4             [SALES]      as SALES,
 5             [RESEARCH]   as RESEARCH,
 6             [OPERATIONS] as OPERATIONS
 7        from (
 8                select d.dname, e.empno
 9                  from emp e,dept d
10                 where e.deptno=d.deptno
11
12              ) driver
13        pivot (
14          count(driver.empno)
15          for driver.dname in
([ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS])
16        ) as empPivot
17  ) new_driver
18  unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
19  ) as un_pivot
```

Надеюсь, прежде чем прочитать этот рецепт, вы ознакомились с предыдущим, потому что вложенный запрос NEW_DRIVER взят оттуда без всяких изменений (если возникают какие-то вопросы, пожалуйста, рассмотрите сначала предыдущий рецепт, а потом переходите

к данному). Поскольку строки 3–16 повторяют код, который мы уже видели, единственное новшество здесь – строка 18, в которой используется оператор UNPIVOT.

Команда UNPIVOT просто просматривает результирующее множество, возвращаемое NEW_DRIVER, и обрабатывает каждый столбец и строку. Например, оператор UNPIVOT обрабатывает имена столбцов, возвращаемые NEW_DRIVER. Встречая имя ACCOUNTING, он преобразует имя столбца ACCOUNTING в значение строки (поля DNAME). Он также берет возвращаемое NEW_DRIVER значение ACCOUNTING (которое равно 3) и возвращает его тоже как часть строки ACCOUNTING (значение столбца CNT). UNPIVOT делает таким образом для каждого из элементов, определенных в списке FOR, и просто возвращает каждый из них как строку.

Новое результирующее множество компактно и состоит всего из двух столбцов, DNAME и CNT, и четырех строк:

```
select DNAME, CNT
  from (
    select [ACCOUNTING] as ACCOUNTING,
           [SALES]      as SALES,
           [RESEARCH]   as RESEARCH,
           [OPERATIONS] as OPERATIONS
      from (
              select d.dname, e.empno
                from emp e,dept d
               where e.deptno=d.deptno

            ) driver
      pivot (
        count(driver.empno)
        for driver.dname in ( [ACCOUNTING],[SALES],[RESEARCH],[OPERATIONS] )
      ) as empPivot
) new_driver
unpivot (cnt for dname in (ACCOUNTING,SALES,RESEARCH,OPERATIONS)
) as un_pivot
```

```
DNAME                CNT
-------------- ----------
ACCOUNTING              3
RESEARCH                5
SALES                   6
OPERATIONS              0
```

Транспонирование результирующего множества с помощью оператора Oracle MODEL

Задача

Как и в первом рецепте данной главы, необходимо найти альтернативу традиционным, уже известным техникам разворачивания. Хочется

поработать с Oracle-оператором MODEL. В отличие от оператора SQL Server PIVOT оператор Oracle MODEL предназначен не для разворачивания результирующих множеств. На самом деле, я не ошибусь, если скажу, что применение оператора MODEL для разворачивания является применением его не по назначению; очевидно, что он не для этого был задуман. Тем не менее оператор MODEL обеспечивает интересный подход к обычной задаче. В данном конкретном случае требуется преобразовать следующее результирующее множество:

```
select deptno, count(*) cnt
  from emp
 group by deptno
```

```
DEPTNO        CNT
------ ----------
    10          3
    20          5
    30          6
```

в такое множество:

```
       D10        D20        D30
---------- ---------- ----------
         3          5          6
```

Решение

Используйте в операторе MODEL агрегацию и выражения CASE точно так же, как делали бы это при разворачивании традиционными методами. Основное отличие в данном случае в том, что для хранения значений, полученных в результате агрегации, используются массивы, и массивы возвращаются в результирующем множестве:

```
select max(d10) d10,
       max(d20) d20,
       max(d30) d30
  from (
select d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
  dimension by(deptno d)
   measures(deptno, cnt d10, cnt d20, cnt d30)
   rules(
     d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
     d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
     d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
  )
  )
```

Обсуждение

Оператор MODEL является исключительно полезным и мощным дополнением к языку SQL для Oracle. Начав работать с MODEL, вы обна-

ружите такие удобные возможности, как итерация, доступ к значениям строк как к элементам массива, возможность применения к результирующему множеству логики «upsert»[1] и возможность создавать справочные модели. В данном рецепте не используется ни одна из всех этих предлагаемых MODEL замечательных возможностей, но ведь здорово уметь посмотреть на проблему с разных сторон и использовать разные средства неожиданным образом (если нет других причин, то хотя бы, чтобы понять, в каких случаях те или иные возможности более полезны, чем остальные).

Первый шаг к пониманию решения – проанализировать вложенный запрос конструкции FROM. Он просто подсчитывает в таблице EMP количество служащих в каждом отделе. Результаты показаны ниже:

```
select deptno, count(*) cnt
  from emp
 group by deptno

DEPTNO        CNT
------ ----------
    10          3
    20          5
    30          6
```

Результирующее множество – это то, что получает для работы MODEL. Посмотрев на конструкцию MODEL, можно выделить три подоператора: DIMENSION BY, MEASURES и RULES. Начнем с MEASURES.

Элементами списка MEASURES являются массивы, которые были объявлены для этого запроса. В запросе используется четыре массива: DEPTNO, D10, D20 и D30. Как и у столбцов в списке оператора SELECT, у массивов в списке MEASURES могут быть псевдонимы. Как видите, три из четырех массивов являются значениями поля CNT вложенного запроса.

Если в списке MEASURES находятся массивы, тогда в подоператоре DIMENSION BY – индексы массивов. Рассмотрим следующее: массив D10 – просто псевдоним CNT. Если взглянуть на результирующее множество вложенного запроса, приведенного выше, увидим, что CNT имеет три значения: 3, 5 и 6. При создании массива CNT создается массив с тремя элементами, а именно тремя целыми числами, 3, 5 и 6. Как теперь организовать доступ к каждому из этих элементов по отдельности? С помощью индекса массива. Индекс, определенный в подоператоре DIMENSION BY, имеет три значения: 10, 20 и 30 (из приведенного выше результирующего множества). Итак, к примеру, следующее выражение:

```
d10[10]
```

[1] Upsert – изменить данные, если они существуют/вставить, если их еще нет. – *Примеч. перев.*

равно 3, поскольку здесь осуществляется доступ к значению CNT в массиве D10 для DEPTNO 10 (которое равно 3).

Поскольку все три массива (D10, D20, D30) содержат значения CNT, для всех трех результаты одинаковые. Как тогда поместить элемент в соответствующий массив? Введем подоператор RULES. Из результирующего множества, показанного ранее, можно увидеть, что значениями DEPTNO являются 10, 20 и 30. Выражения с участием CASE в конструкции RULES просто определяют каждое значение массива DEPTNO:

- Если значение равно 10, сохраняем значение CNT, соответствующее DEPTNO 10, в D10[10], в противном случае сохраняем 0.
- Если значение равно 20, сохраняем значение CNT, соответствующее DEPTNO 20, в D20[20], в противном случае сохраняем 0.
- Если значение равно 30, сохраняем значение CNT, соответствующее DEPTNO 30, в D30[30], в противном случае сохраняем 0.

Если вы почувствовали себя Алисой, падающей в кроличью нору, не волнуйтесь; просто остановитесь и выполните все то, что мы обсудили. Иногда проще прочитать, взглянуть на код, реализующий то, что было прочитано, вернуться и прочитать все еще раз. Представленный далее код на самом деле довольно прост, надо лишь увидеть его в действии:

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
  dimension by(deptno d)
   measures(deptno, cnt d10, cnt d20, cnt d30)
   rules(
     d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
     d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
     d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
  )

DEPTNO        D10        D20        D30
------ ---------- ---------- ----------
    10          3          0          0
    20          0          5          0
    30          0          0          6
```

Как видите, именно подоператор RULES меняет значения массивов. Если это до сих пор непонятно, выполните запрос, закомментировав выражения в конструкции RULES:

```
select deptno, d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
  dimension by(deptno d)
   measures(deptno, cnt d10, cnt d20, cnt d30)
   rules(
    /*
     d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
```

```
     d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
     d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
    */
  )
DEPTNO        D10        D20        D30
------ ---------- ---------- ----------
    10          3          3          3
    20          5          5          5
    30          6          6          6
```

Теперь вы должны понять, что оператор MODEL возвращает такое же результирующее множество, что и вложенный запрос, только результатам операции COUNT присвоены псевдонимы D10, D20 и D30. Следующий запрос доказывает это:

```
select deptno, count(*) d10, count(*) d20, count(*) d30
  from emp
 group by deptno
DEPTNO        D10        D20        D30
------ ---------- ---------- ----------
    10          3          3          3
    20          5          5          5
    30          6          6          6
```

Итак, оператор MODEL взял значения для DEPTNO и CNT, поместил их в массивы и затем обеспечил представление каждого массива отдельным DEPTNO. На данный момент каждый массив, D10, D20 и D30, имеет одно отличное от нуля значение, представляющее CNT для данного DEPTNO. Результирующее множество уже транспонировано. Осталось только применить агрегатную функцию MAX (можно было бы использовать MIN или SUM; в данном случае нет никакий разницы), чтобы возвратить только одну строку:

```
select max(d10) d10,
       max(d20) d20,
       max(d30) d30
  from (
select d10,d20,d30
  from ( select deptno, count(*) cnt from emp group by deptno )
 model
  dimension by(deptno d)
   measures(deptno, cnt d10, cnt d20, cnt d30)
   rules(
     d10[any] = case when deptno[cv()]=10 then d10[cv()] else 0 end,
     d20[any] = case when deptno[cv()]=20 then d20[cv()] else 0 end,
     d30[any] = case when deptno[cv()]=30 then d30[cv()] else 0 end
  )
  )
      D10        D20        D30
---------- ---------- ----------
         3          5          6
```

Извлечение элементов строки, положение которых в строке неизвестно

Задача

Имеется строковое поле, содержащее сериализованные журнальные данные. Требуется провести синтаксический анализ строки и извлечь из нее необходимую информацию. К сожалению, эта информация может находиться с любом месте строки. Поэтому при решении задачи придется полагаться на тот факт, что интересующая нас информация окружена определенными символами. Например, рассмотрим следующие строки:

```
xxxxxabc[867]xxx[-]xxxx[5309]xxxxx
xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx
call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx
film:[non_marked]qq:[unit]tailpipe:[withabanana?]80sxxxxx
```

Требуется извлечь значения, заключенные в квадратные скобки, и получить следующее результирующее множество:

```
FIRST_VAL       SECOND_VAL      LAST_VAL
--------------- --------------- ---------------
867             -               5309
11271978        4               Joe
F_GET_ROWS()    ROSEWOOD...SIR  44400002
non_marked      unit            withabanana?
```

Решение

Несмотря на отсутствие информации о точном местоположении интересующих нас значений, известно, что они заключены в квадратные скобки, [], и что их всего три. С помощью встроенной функции Oracle INSTR найдите местоположения скобок и, используя встроенную функцию SUBSTR, извлеките значения из строки. Представление V будет содержать строки, подлежащие синтаксическому разбору, и определяется следующим образом (оно используется исключительно для удобства чтения):

```
create view V
as
select 'xxxxxabc[867]xxx[-]xxxx[5309]xxxxx' msg
    from dual
   union all
  select 'xxxxxtime:[11271978]favnum:[4]id:[Joe]xxxxx' msg
    from dual
   union all
  select 'call:[F_GET_ROWS()]b1:[ROSEWOOD...SIR]b2:[44400002]77.90xxxxx' msg
    from dual
   union all
  select 'film:[non_marked]qq:[unit]tailpipe:[withabanana?]80sxxxxx' msg
```

```
   from dual

 1  select substr(msg,
 2          instr(msg,'[',1,1)+1,
 3          instr(msg,']',1,1)-instr(msg,'[',1,1)-1) first_val,
 4         substr(msg,
 5          instr(msg,'[',1,2)+1,
 6          instr(msg,']',1,2)-instr(msg,'[',1,2)-1) second_val,
 7         substr(msg,
 8          instr(msg,'[',-1,1)+1,
 9          instr(msg,']',-1,1)-instr(msg,'[',-1,1)-1) last_val
10    from V
```

Обсуждение

Применение встроенной функции Oracle INSTR значительно упрощает решение этой задачи. Поскольку известно, что искомые значения заключены в [] и что в строке три набора [], первый шаг решения – с помощью INSTR найти числовые позиции [] в каждой строке. Следующий пример возвращает числовые позиции открывающих и закрывающих скобок в каждой строке:

```
select instr(msg,'[',1,1) "1st_[",
       instr(msg,']',1,1) "]_1st",
       instr(msg,'[',1,2) "2nd_[",
       instr(msg,']',1,2) "]_2nd",
       instr(msg,'[',-1,1) "3rd_[",
       instr(msg,']',-1,1) "]_3rd"
  from V

1st_[ ]_1st      2nd_[ ]_2nd      3rd_[ ]_3rd
------ ----- ---------- ----- ---------- -----
     9    13         17    19         24    29
    11    20         28    30         34    38
     6    19         23    38         42    51
     6    17         21    26         36    49
```

Вся тяжелая работа выполнена. Осталось лишь вставить полученные числовые позиции в SUBSTR и разобрать MSG. В законченном примере можно заметить, что над значениями, возвращаемыми INSTR, выполняются простые арифметические действия, в частности, +1 и -1. Это гарантирует, что в окончательном результирующем множестве не будет открывающей квадратной скобки [. Ниже представлено решение без добавления и вычитания 1 из возвращаемых INSTR значений. Обратите внимание, что в этом случае каждое значение начинается с открывающей квадратной скобки:

```
select substr(msg,
        instr(msg,'[',1,1),
        instr(msg,']',1,1)-instr(msg,'[',1,1)) first_val,
       substr(msg,
        instr(msg,'[',1,2),
```

```
        instr(msg,']',1,2)-instr(msg,'[',1,2)) second_val,
       substr(msg,
        instr(msg,'[',-1,1),
        instr(msg,']',-1,1)-instr(msg,'[',-1,1)) last_val
  from V

FIRST_VAL       SECOND_VAL      LAST_VAL
--------------- --------------- -------------
[867            [-              [5309
[11271978       [4              [Joe
[F_GET_ROWS()   [ROSEWOOD...SIR [44400002
[non_marked     [unit           [withabanana?
```

Из приведенного выше результирующего множества видим, что со значениями возвращается и открывающая скобка. Вероятно, вы думаете: «Хорошо, верну добавление 1 к INSTR и избавлюсь от начальной квадратной скобки. Зачем вычитать 1?» Причина такова: если вернуть добавление без вычитания, в результаты попадут закрывающие квадратные скобки, как можно увидеть ниже:

```
select substr(msg,
        instr(msg,'[',1,1)+1,
        instr(msg,']',1,1)-instr(msg,'[',1,1)) first_val,
       substr(msg,
        instr(msg,'[',1,2)+1,
        instr(msg,']',1,2)-instr(msg,'[',1,2)) second_val,
       substr(msg,
        instr(msg,'[',-1,1)+1,
        instr(msg,']',-1,1)-instr(msg,'[',-1,1)) last_val
  from V

FIRST_VAL       SECOND_VAL      LAST_VAL
--------------- --------------- -------------
867]            -]              5309]
11271978]       4]              Joe]
F_GET_ROWS()]   ROSEWOOD...SIR] 44400002]
non_marked]     unit]           withabanana?]
```

На данный момент должно быть понятно: чтобы гарантировать отсутствие квадратных скобок в возвращаемых результатах, необходимо добавить 1 к начальному индексу и отнять 1 из конечного индекса.

Как определить количество дней в году (альтернативное решение для Oracle)

Задача

Требуется определить количество дней в году.

Рассматриваемый рецепт представляет альтернативу решению задачи «Как определить количество дней в году» главы 9. Данное решение подойдет только для Oracle.

Решение

С помощью функции TO_CHAR представьте последнюю дату года как трехзначный порядковый номер дня года:

```
1  select 'Days in 2005: '||
2         to_char(add_months(trunc(sysdate,'y'),12)-1,'DDD')
3         as report
4    from dual
5  union all
6  select 'Days in 2004: '||
7         to_char(add_months(trunc(
8                       to_date('01-SEP-2004'),'y'),12)-1,'DDD')
9    from dual

REPORT
-----------------
Days in 2005: 365
Days in 2004: 366
```

Обсуждение

Начнем с использования функции TRUNC, чтобы возвратить первый день года соответственно заданной дате:

```
select trunc(to_date('01-SEP-2004'),'y')
  from dual

TRUNC(TO_DA
-----------
01-JAN-2004
```

Далее к полученной дате с помощью ADD_MONTHS добавляем один год (12 месяцев). После этого вычитаем один день, что возвращает нас в последний день года соответственно исходной дате:

```
select add_months(
         trunc(to_date('01-SEP-2004'),'y'),
         12) before_subtraction,
       add_months(
         trunc(to_date('01-SEP-2004'),'y'),
         12)-1 after_subtraction
  from dual

BEFORE_SUBT AFTER_SUBTR
----------- -----------
01-JAN-2005 31-DEC-2004
```

Теперь, получив последний день рассматриваемого года, просто применяем TO_CHAR и возвращаем трехзначное число, представляющее, каким по счету днем (первым, пятидесятым и т. д.) является последний день года:

```
select to_char(
         add_months(
```

```
          trunc(to_date('01-SEP-2004'),'y'),
          12)-1,'DDD') num_days_in_2004
  from dual

NUM
---
366
```

Поиск смешанных буквенно-цифровых строк

Задача

Имеется столбец со смешанными буквенно-цифровыми данными. Требуется выбрать строки, содержащие и буквенные, и числовые символы. Иначе говоря, если в строке присутствуют только числа или только буквы, она нас не интересует. Возвращаемые значения должны сочетать в себе буквы и числа. Рассмотрим следующие данные:

```
STRINGS
------------
1010 switch
333
3453430278
ClassSummary
findRow 55
threes
```

Окончательное результирующее множество должно содержать только те строки, в которых присутствуют и буквы, и числа:

```
STRINGS
------------
1010 switch
findRow 55
```

Решение

С помощью встроенной функции TRANSLATE замените все буквы и цифры экземплярами определенных символов. Затем выберите только те строки, в которых оба символа встречаются хотя бы по разу. В решении используется синтаксис Oracle, но DB2 и PostgreSQL поддерживают TRANSLATE, поэтому доработать решение, чтобы использовать его для этих платформ, не составит труда:

```
with v as (
select 'ClassSummary' strings from dual union
select '3453430278'           from dual union
select 'findRow 55'           from dual union
select '1010 switch'          from dual union
select '333'                  from dual union
select 'threes'               from dual
)
select strings
```

```
  from (
select strings,
       translate(
         strings,
         'abcdefghijklmnopqrstuvwxyz0123456789',
         rpad('#',26,'#')||rpad('*',10,'*')) translated
  from v
       ) x
 where instr(translated,'#') > 0
   and instr(translated,'*') > 0
```

В качестве альтернативы оператору WITH можно использовать вложенный запрос или просто создать представление.

Обсуждение

Функция TRANSLATE чрезвычайно упрощает решение этой задачи. Первый шаг – с помощью TRANSLATE заменить все буквы и цифры символом фунта (#) и звездочкой (*) соответственно. Вот промежуточные результаты (возвращаемые вложенным запросом X):

```
with v as (
select 'ClassSummary' strings from dual union
select '3453430278'           from dual union
select 'findRow 55'           from dual union
select '1010 switch'          from dual union
select '333'                  from dual union
select 'threes'               from dual
)
select strings,
       translate(
         strings,
         'abcdefghijklmnopqrstuvwxyz0123456789',
         rpad('#',26,'#')||rpad('*',10,'*')) translated
  from v
```

```
STRINGS      TRANSLATED
------------ ------------
1010 switch  **** ######
333          ***
3453430278   **********
ClassSummary C####S######
findRow 55   ####R## **
threes       ######
```

Теперь осталось только выбрать строки, имеющие, по крайней мере, по одному экземпляру «#» и «*». С помощью функции INSTR определите, присутствуют ли эти символы в строке. Если да, возвращаемое значение будет больше нуля. Окончательные строки, которые должны быть получены (для ясности вместе с их транслированными значениями), показаны далее:

```
with v as (
select 'ClassSummary' strings from dual union
select '3453430278'           from dual union
select 'findRow 55'           from dual union
select '1010 switch'          from dual union
select '333'                  from dual union
select 'threes'               from dual
)
select strings, translated
  from (
select strings,
       translate(
         strings,
         'abcdefghijklmnopqrstuvwxyz0123456789',
         rpad('#',26,'#')||rpad('*',10,'*')) translated
  from v
       )
 where instr(translated,'#') > 0
   and instr(translated,'*') > 0
```

```
STRINGS      TRANSLATED
------------ ------------
1010 switch  **** ######
findRow 55   ####R## **
```

Преобразование целых чисел в их двоичное представление с использованием Oracle

Задача

Требуется преобразовать целое число в его двоичное представление в системе Oracle. Например, необходимо получить все заработные платы таблицы EMP в двоичном виде как часть следующего результирующего множества:

```
ENAME        SAL SAL_BINARY
---------- ----- --------------------
SMITH        800 1100100000
ALLEN       1600 11001000000
WARD        1250 10011100010
JONES       2975 101110011111
MARTIN      1250 10011100010
BLAKE       2850 101100100010
CLARK       2450 100110010010
SCOTT       3000 101110111000
KING        5000 1001110001000
TURNER      1500 10111011100
ADAMS       1100 10001001100
JAMES        950 1110110110
FORD        3000 101110111000
MILLER      1300 10100010100
```

Решение

В этом решении используется оператор MODEL, поэтому оно подходит только для Oracle Database 10*g* и более поздних версий. MODEL обладает возможностью выполнять итерации и обеспечивать доступ к значениям строк как к элементам массива, поэтому естественно было выбрать его для этой операции (исходя из предположения, что задача должна быть решена в SQL, поскольку пользовательская функция была бы здесь более уместна). Как уже говорилось в отношении всех решений данной книги, если вы не видите практического применения этому коду, сосредоточьтесь на технике. Полезно знать, что оператор MODEL может выполнять процедурные задачи, сохраняя при этом ориентированную на работу с множествами природу и мощь, свойственные SQL. Возможно, вы говорите себе: «Я бы никогда не делал этого в SQL». Я ни в коем случае не указываю, что вы должны делать, а что не должны. Я только призываю сосредоточиться на технике, чтобы суметь применить ее, если вдруг столкнетесь с возможностью более «практического» ее применения.

Следующее решение возвращает все значения столбцов ENAME и SAL из таблицы EMP. При этом в скалярном подзапросе вызывается оператор MODEL (здесь он выполняет роль обычной функции, работающей с таблицей EMP, которая просто получает входные данные, обрабатывает их и возвращает значения, во многом как это делала бы любая другая функция):

```
 1  select ename,
 2         sal,
 3         (
 4         select bin
 5           from dual
 6          model
 7          dimension by ( 0 attr )
 8          measures ( sal num,
 9                     cast(null as varchar2(30)) bin,
10                     '0123456789ABCDEF' hex
11                   )
12          rules iterate (10000) until (num[0] <= 0) (
13            bin[0] = substr(hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
14            num[0] = trunc(num[cv()]/2)
15          )
16         ) sal_binary
17    from emp
```

Обсуждение

В разделе «Решение» я говорил, что данную задачу лучше решать с помощью пользовательской функции. В самом деле, идея этого рецепта возникла из функции. Фактически он является адаптацией функции TO_BASE, написанной Томом Кайтом, сотрудником корпорации Oracle. Как и остальные рецепты данной книги, этот рецепт, даже если вы

решите его не использовать, полезен тем, что демонстрирует некоторые возможности оператора MODEL, такие как итерации и доступ к строкам как к элементам массива.

Чтобы упростить объяснение, я собираюсь показать небольшие вариации подзапроса, содержащего оператор MODEL. Приведенный далее код является кодом подзапроса из решения за исключением того, что здесь жестко запрограммировано возвратить двоичное представление значения 2:

```
select bin
  from dual
 model
 dimension by ( 0 attr )
 measures ( 2 num,
            cast(null as varchar2(30)) bin,
            '0123456789ABCDEF' hex
          )
 rules iterate (10000) until (num[0] <= 0) (
   bin[0] = substr (hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
   num[0] = trunc(num[cv()]/2)
 )

BIN
----------
10
```

Следующий запрос выводит значения, возвращенные в результате одной итерации конструкции RULES из запроса выше:

```
select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
  from dual

START_VAL HEX              BIN        NUM
--------- ---------------- ---------- ---
        2 0123456789ABCDEF 0            1
```

Поле START_VAL содержит число, для которого должно быть получено двоичное представление, в данном случае это 2. Значение столбца BIN – результат операции SUBSTR над строкой ‘0123456789ABCDEF’ (в оригинальном решении HEX). Значение NUM – это признак выхода из цикла.

Как видно из предыдущего результирующего множества, после первой итерации цикла BIN равен 0, и NUM равен 1. Поскольку значение NUM не меньше или не равно 0, выполняется следующая итерация, результаты которой представляет приведенное ниже выражение SQL:

```
select num start_val,
       substr('0123456789ABCDEF',mod(1,2)+1,1) || bin bin,
```

```
       trunc(1/2) num
  from (
select 2 start_val,
       '0123456789ABCDEF' hex,
       substr('0123456789ABCDEF',mod(2,2)+1,1) ||
       cast(null as varchar2(30)) bin,
       trunc(2/2) num
  from dual
       )

START_VAL BIN        NUM
--------- ---------- ---
        1 10           0
```

После следущей итерации операция SUBSTR над HEX возвращает 1, к которой присоединяется предыдущее значение BIN, 0. Проверочное значение, NUM, теперь равно 0; таким образом, это последняя итерация, а возвращенное значение «10» – двоичное представление числа 2. Разобравшись с тем, что происходит, можно убрать итерацию из оператора MODEL и строка за строкой проследить, как применяются правила для получения окончательного результирующего множества, что показано ниже:

```
select 2 orig_val, num, bin
  from dual
 model
 dimension by ( 0 attr )
 measures ( 2 num,
            cast(null as varchar2(30)) bin,
            '0123456789ABCDEF' hex
          )
 rules (
   bin[0] = substr (hex[cv()],mod(num[cv()],2)+1,1)||bin[cv()],
   num[0] = trunc(num[cv()]/2),
   bin[1] = substr (hex[0],mod(num[0],2)+1,1)||bin[0],
   num[1] = trunc(num[0]/2)
 )

ORIG_VAL NUM BIN
-------- --- ---------
       2   1 0
       2   0 10
```

Разворачивание ранжированного результирующего множества

Задача

Требуется ранжировать значения таблицы, затем развернуть результирующее множество и получить три столбца. Идея в том, чтобы показать в разных столбцах три наибольших значения, три следующих значения и все остальные. Например, необходимо ранжировать слу-

жащих в таблице EMP по значениям SAL и затем развернуть результаты в три столбца. Должно быть получено следующее результирующее множество:

```
TOP_3           NEXT_3          REST
--------------- --------------- --------------
KING   (5000)   BLAKE  (2850)   TURNER (1500)
FORD   (3000)   CLARK  (2450)   MILLER (1300)
SCOTT  (3000)   ALLEN  (1600)   MARTIN (1250)
JONES  (2975)                   WARD   (1250)
                                ADAMS  (1100)
                                JAMES  (950)
                                SMITH  (800)
```

Решение

Прежде всего, в данном решении необходимо применить ранжирующую функцию DENSE_RANK OVER, которая выполнит ранжирование служащих по SAL, допуская при этом одинаковые значения ранга для дублирующихся значений. В результате этого можно сразу увидеть три наивысшие, следующие три и все остальные заработные платы. Далее с помощью ранжирующей функции ROW_NUMBER OVER сортируем служащих в группах (группа «три наибольшие», «следующие три» или группа «остальные»). Чтобы облагородить результаты, выполняем после этого классическое транспонирование, применяя при этом встроенные функции для работы со строками, доступные на используемой платформе. В следующем решении используется синтаксис Oracle. Поскольку DB2 и SQL Server 2005 поддерживают оконные функции, трансформировать решение под эти платформы не составит труда:

```
 1  select max(case grp when 1 then rpad(ename,6) ||
 2                      ' ('|| sal ||')' end) top_3,
 3         max(case grp when 2 then rpad(ename,6) ||
 4                      ' ('|| sal ||')' end) next_3,
 5         max(case grp when 3 then rpad(ename,6) ||
 6                      ' ('|| sal ||')' end) rest
 7    from (
 8  select ename,
 9         sal,
10         rnk,
11         case when rnk <= 3 then 1
12              when rnk <= 6 then 2
13              else               3
14         end grp,
15         row_number()over (
16           partition by case when rnk <= 3 then 1
17                             when rnk <= 6 then 2
18                             else               3
19                        end
20               order by sal desc, ename
```

```
21         ) grp_rnk
22    from (
23  select ename,
24         sal,
25         dense_rank()over(order by sal desc) rnk
26    from emp
27         ) x
28         ) y
29   group by grp_rnk
```

Обсуждение

Данный рецепт является замечательным примером того, как много можно сделать с небольшой помощью оконных функций. Решение может показаться сложным, но как только вы разложите его на составляющие, будете удивлены его простоте. Начнем с выполнения вложенного запроса X:

```
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
  from emp
```

```
ENAME        SAL        RNK
---------- ----- ----------
KING        5000          1
SCOTT       3000          2
FORD        3000          2
JONES       2975          3
BLAKE       2850          4
CLARK       2450          5
ALLEN       1600          6
TURNER      1500          7
MILLER      1300          8
WARD        1250          9
MARTIN      1250          9
ADAMS       1100         10
JAMES        950         11
SMITH        800         12
```

Как видно из приведенного выше результирующего множества, вложенный запрос X просто ранжирует записи служащих по SAL, допуская при этом наличие одинаковых значений рангов для дублирующихся значений (поскольку используется функция DENSE_RANK, а не RANK, получаем непрерывную последовательность значений рангов). Следующий шаг – сгруппировать строки, возвращенные вложенным запросом X, применяя выражения CASE для обработки результатов, возвращаемых DENSE_RANK. Кроме того, используем ранжирующую функцию ROW_NUMBER OVER и ранжируем служащих по SAL в их группах (в рамках групп, созданных выражением CASE). Все это происходит во вложенном запросе Y и показано ниже:

```
select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
            when rnk <= 6 then 2
            else               3
       end grp,
       row_number()over (
         partition by case when rnk <= 3 then 1
                           when rnk <= 6 then 2
                           else               3
                      end
            order by sal desc, ename
       ) grp_rnk
  from (
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
  from emp
       ) x
```

```
ENAME        SAL  RNK  GRP GRP_RNK
---------- ----- ---- ---- -------
KING        5000    1    1       1
FORD        3000    2    1       2
SCOTT       3000    2    1       3
JONES       2975    3    1       4
BLAKE       2850    4    2       1
CLARK       2450    5    2       2
ALLEN       1600    6    2       3
TURNER      1500    7    3       1
MILLER      1300    8    3       2
MARTIN      1250    9    3       3
WARD        1250    9    3       4
ADAMS       1100   10    3       5
JAMES        950   11    3       6
SMITH        800   12    3       7
```

Теперь запрос начинает вырисовываться. Если проследить его с самого начала (от вложенного запроса X), можно увидеть, что он не так уж и сложен. На данный момент запрос возвращает всех служащих и для каждого из них заработную плату (SAL), ранг (RNK), представляющий уровень заработной платы служащего по сравнению со всеми остальными сотрудниками, номер группы (GRP), обозначающий, к какой группе принадлежит служащий (на основании SAL), и, наконец, ранг в группе (GRP_RANK), определяемый на основании значения SAL в рамках отдельной GRP.

Теперь выполним традиционное разворачивание по ENAME, присоединяя при этом значения SAL оператором конкатенации Oracle «||». Функция RPAD обеспечивает выравнивание числовых значений в круглых скобках. Наконец, применяем GROUP BY по GRP_RNK, чтобы в ре-

зультирующем множестве гарантированно присутствовали все служащие. Окончательное результирующее множество показано ниже:

```
select max(case grp when 1 then rpad(ename,6) ||
                    ' ('|| sal ||')' end) top_3,
       max(case grp when 2 then rpad(ename,6) ||
                    ' ('|| sal ||')' end) next_3,
       max(case grp when 3 then rpad(ename,6) ||
                    ' ('|| sal ||')' end) rest
  from (
select ename,
       sal,
       rnk,
       case when rnk <= 3 then 1
            when rnk <= 6 then 2
            else               3
       end grp,
       row_number()over (
         partition by case when rnk <= 3 then 1
                           when rnk <= 6 then 2
                           else               3
                      end
            order by sal desc, ename
       ) grp_rnk
  from (
select ename,
       sal,
       dense_rank()over(order by sal desc) rnk
  from emp
       ) x
       ) y
 group by grp_rnk

TOP_3           NEXT_3          REST
--------------- --------------- -------------
KING   (5000)   BLAKE  (2850)   TURNER (1500)
FORD   (3000)   CLARK  (2450)   MILLER (1300)
SCOTT  (3000)   ALLEN  (1600)   MARTIN (1250)
JONES  (2975)                   WARD   (1250)
                                ADAMS  (1100)
                                JAMES  (950)
                                SMITH  (800)
```

Если проанализировать запросы на всех этапах, можно заметить, что обращение к таблице EMP происходит только один раз. Одно из выдающихся свойств оконных функций – то, как много всего можно сделать всего за одно обращение к данным. Не нужны рефлексивные объединения или временные таблицы. Лишь достаем необходимые строки и позволяем оконным функциям сделать всю работу. Доступ к таблице EMP осуществляется только во вложенном запросе X. Остальное – просто манипулирование данными с целью придать им желаемый вид.

Представьте, что все это значит для производительности, когда можно создать такой отчет лишь за одно обращение к таблице. Просто замечательно.

Как добавить заголовок столбца в дважды развернутое результирующее множество

Задача

Необходимо создать два результирующих множества и затем разворачиванием скомпоновать их в два столбца. Кроме того, каждому столбцу строк должен быть присвоен «заголовок». Например, имеется две таблицы, содержащие даные о служащих, работающих в различных подразделениях компании (скажем, в исследовательском и эксплуатационном)

```
select * from it_research

DEPTNO ENAME
------ --------------------
   100 HOPKINS
   100 JONES
   100 TONEY
   200 MORALES
   200 P.WHITAKER
   200 MARCIANO
   200 ROBINSON
   300 LACY
   300 WRIGHT
   300 J.TAYLOR

select * from it_apps

DEPTNO ENAME
------ -----------------
   400 CORRALES
   400 MAYWEATHER
   400 CASTILLO
   400 MARQUEZ
   400 MOSLEY
   500 GATTI
   500 CALZAGHE
   600 LAMOTTA
   600 HAGLER
   600 HEARNS
   600 FRAZIER
   700 GUINN
   700 JUDAH
   700 MARGARITO
```

Хотелось бы создать отчет, в котором все служащие из этих таблиц были бы перечислены в двух столбцах. Необходимо представить все отделы

(DEPTNO) и всех служащих (ENAME) для каждого из них. В итоге должно быть получено следующее результирующее множество:

```
RESEARCH             APPS
-------------------- ---------------
100                  400
  JONES                MAYWEATHER
  TONEY                CASTILLO
  HOPKINS              MARQUEZ
200                    MOSLEY
  P.WHITAKER           CORRALES
  MARCIANO           500
  ROBINSON             CALZAGHE
  MORALES              GATTI
300                  600
  WRIGHT               HAGLER
  J.TAYLOR             HEARNS
  LACY                 FRAZIER
                       LAMOTTA
                     700
                       JUDAH
                       MARGARITO
                       GUINN
```

Решение

В целом для данного решения не требуется ничего, кроме простого составления данных и разворачивания (объединения, затем разворачивания) с дополнительным изменением порядка: DEPTNO должно предшествовать ENAME. В приводимой здесь технике формирование дополнительных строк для каждого DEPTNO осуществляется с помощью декартова произведения. В результате мы получаем строки, необходимые для представления всех служащих, плюс место для вывода DEPTNO. Решение использует синтаксис Oracle, но поскольку DB2 поддерживает оконные функции, которые могут работать со скользящими окнами данных (оператор кадрирования), это решение нетрудно изменить для работы с DB2. Таблицы IT_RESEARCH и IT_APPS используются только в данном рецепте, поэтому выражения создания этих таблиц показаны вместе с решением:

```
create table IT_research (deptno number, ename varchar2(20))

insert into IT_research values (100,'HOPKINS')
insert into IT_research values (100,'JONES')
insert into IT_research values (100,'TONEY')
insert into IT_research values (200,'MORALES')
insert into IT_research values (200,'P.WHITAKER')
insert into IT_research values (200,'MARCIANO')
insert into IT_research values (200,'ROBINSON')
insert into IT_research values (300,'LACY')
insert into IT_research values (300,'WRIGHT')
```

```
insert into IT_research values (300,'J.TAYLOR')

create table IT_apps (deptno number, ename varchar2(20))

insert into IT_apps values (400,'CORRALES')
insert into IT_apps values (400,'MAYWEATHER')
insert into IT_apps values (400,'CASTILLO')
insert into IT_apps values (400,'MARQUEZ')
insert into IT_apps values (400,'MOSLEY')
insert into IT_apps values (500,'GATTI')
insert into IT_apps values (500,'CALZAGHE')
insert into IT_apps values (600,'LAMOTTA')
insert into IT_apps values (600,'HAGLER')
insert into IT_apps values (600,'HEARNS')
insert into IT_apps values (600,'FRAZIER')
insert into IT_apps values (700,'GUINN')
insert into IT_apps values (700,'JUDAH')
insert into IT_apps values (700,'MARGARITO')

 1  select max(decode(flag2,0,it_dept)) research,
 2         max(decode(flag2,1,it_dept)) apps
 3    from (
 4  select sum(flag1)over(partition by flag2
 5                            order by flag1,rownum) flag,
 6         it_dept, flag2
 7    from (
 8  select 1 flag1, 0 flag2,
 9         decode(rn,1,to_char(deptno),'  '||ename) it_dept
10    from (
11  select x.*, y.id,
12         row_number()over(partition by x.deptno order by y.id) rn
13    from (
14  select deptno,
15         ename,
16         count(*)over(partition by deptno) cnt
17    from it_research
18         ) x,
19         (select level id from dual connect by level <= 2) y
20         )
21   where rn <= cnt+1
22  union all
23  select 1 flag1, 1 flag2,
24         decode(rn,1,to_char(deptno),'  '||ename) it_dept
25    from (
26  select x.*, y.id,
27         row_number()over(partition by x.deptno order by y.id) rn
28    from (
29  select deptno,
30         ename,
31         count(*)over(partition by deptno) cnt
32    from it_apps
33         ) x,
```

```
34         (select level id from dual connect by level <= 2) y
35         )
36   where rn <= cnt+1
37         ) tmp1
38         ) tmp2
39   group by flag
```

Обсуждение

Как и многие другие запросы для построения хранилищ данных/составления отчетов, представленное решение кажется довольно запутанным. Но если разложить его на составляющие, станет понятным, что в нем нет ничего, кроме объединения, разворачивания и декартова произведения. Сначала рассмотрим все части оператора UNION ALL по отдельности, затем сведем их вместе для разворачивания. Начнем с нижней части UNION ALL:

```
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
       ) z
 where rn <= cnt+1
```

```
FLAG1      FLAG2 IT_DEPT
----- ---------- --------------------------
    1          1 400
    1          1   MAYWEATHER
    1          1   CASTILLO
    1          1   MARQUEZ
    1          1   MOSLEY
    1          1   CORRALES
    1          1 500
    1          1   CALZAGHE
    1          1   GATTI
    1          1 600
    1          1   HAGLER
    1          1   HEARNS
    1          1   FRAZIER
    1          1   LAMOTTA
    1          1 700
    1          1   JUDAH
    1          1   MARGARITO
    1          1   GUINN
```

Рассмотрим, как именно образуется результирующее множество. Разложив представленный выше запрос на его простейшие компоненты, получаем вложенный запрос X, который просто возвращает из таблицы IT_APPS все значения ENAME и DEPTNO и количество служащих в каждом отделе. Результаты таковы:

```
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
```

```
DEPTNO ENAME                     CNT
------ ------------------- ----------
   400 CORRALES                     5
   400 MAYWEATHER                   5
   400 CASTILLO                     5
   400 MARQUEZ                      5
   400 MOSLEY                       5
   500 GATTI                        2
   500 CALZAGHE                     2
   600 LAMOTTA                      4
   600 HAGLER                       4
   600 HEARNS                       4
   600 FRAZIER                      4
   700 GUINN                        3
   700 JUDAH                        3
   700 MARGARITO                    3
```

Следующий шаг – создание декартова произведения строк, возвращенных вложенным запросом X, и двух строк, сформированных с использованием CONNECT BY из DUAL. Результаты этой операции приведены ниже:

```
select *
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
 order by 2
```

```
DEPTNO ENAME      CNT  ID
------ ---------- --- ---
   500 CALZAGHE     2   1
   500 CALZAGHE     2   2
   400 CASTILLO     5   1
   400 CASTILLO     5   2
   400 CORRALES     5   1
   400 CORRALES     5   2
   600 FRAZIER      4   1
   600 FRAZIER      4   2
```

```
   500 GATTI        2   1
   500 GATTI        2   2
   700 GUINN        3   1
   700 GUINN        3   2
   600 HAGLER       4   1
   600 HAGLER       4   2
   600 HEARNS       4   1
   600 HEARNS       4   2
   700 JUDAH        3   1
   700 JUDAH        3   2
   600 LAMOTTA      4   1
   600 LAMOTTA      4   2
   700 MARGARITO    3   1
   700 MARGARITO    3   2
   400 MARQUEZ      5   1
   400 MARQUEZ      5   2
   400 MAYWEATHER   5   1
   400 MAYWEATHER   5   2
   400 MOSLEY       5   1
   400 MOSLEY       5   2
```

Как видим, теперь в результате декартова произведения с вложенным запросом Y каждая строка вложенного запроса X возвращена дважды. Зачем необходимо декартово произведение, вскоре станет ясно. Следующий шаг – ранжировать всех служащих полученного результирующего множества в рамках каждого DEPTNO по ID (ID в результате декартова произведения имеет значения 1 или 2). Результат этого ранжирования показан в выводе следующего запроса:

```
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
```

```
DEPTNO ENAME      CNT  ID         RN
------ ---------- --- --- ----------
   400 CORRALES     5   1          1
   400 MAYWEATHER   5   1          2
   400 CASTILLO     5   1          3
   400 MARQUEZ      5   1          4
   400 MOSLEY       5   1          5
   400 CORRALES     5   2          6
   400 MOSLEY       5   2          7
   400 MAYWEATHER   5   2          8
   400 CASTILLO     5   2          9
   400 MARQUEZ      5   2         10
   500 GATTI        2   1          1
```

```
  500 CALZAGHE     2   1          2
  500 GATTI        2   2          3
  500 CALZAGHE     2   2          4
  600 LAMOTTA      4   1          1
  600 HAGLER       4   1          2
  600 HEARNS       4   1          3
  600 FRAZIER      4   1          4
  600 LAMOTTA      4   2          5
  600 HAGLER       4   2          6
  600 FRAZIER      4   2          7
  600 HEARNS       4   2          8
  700 GUINN        3   1          1
  700 JUDAH        3   1          2
  700 MARGARITO    3   1          3
  700 GUINN        3   2          4
  700 JUDAH        3   2          5
  700 MARGARITO    3   2          6
```

Ранжируются все служащие, затем их дубликаты. Результирующее множество содержит дубликаты всех служащих из таблицы IT_APP, а также их ранги и значения DEPTNO. Все эти лишние строки необходимы, потому что в результирующем множестве мы должны иметь место для вставки значений DEPTNO в столбец ENAME. В результате декартова произведения IT_APPS и таблицы в одну строку лишних строк не появится (потому что кардинальность любой таблицы × 1= кардинальности данной таблицы).

Следующий шаг – полученное результирующее множество развернуть таким образом, чтобы все значения ENAME располагались в одном столбце, но были разделены по отделам соответствующими значениями DEPTNO. Из следующего запроса видно, как это происходит:

```
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
       ) z
 where rn <= cnt+1
```

```
FLAG1      FLAG2 IT_DEPT
----- ---------- -------------------------
    1          1 400
    1          1   MAYWEATHER
    1          1   CASTILLO
    1          1   MARQUEZ
```

```
     1          1   MOSLEY
     1          1   CORRALES
     1          1 500
     1          1   CALZAGHE
     1          1   GATTI
     1          1 600
     1          1   HAGLER
     1          1   HEARNS
     1          1   FRAZIER
     1          1   LAMOTTA
     1          1 700
     1          1   JUDAH
     1          1   MARGARITO
     1          1   GUINN
```

FLAG1 и FLAG2 начнут действовать позже, поэтому пока обратим внимание на столбец IT_DEPT. Для каждого значения DEPTNO возвращено CNT*2 строк, но необходимо было получить CNT+1 строк, как обозначено предикатом WHERE. Поле RN – это ранг служащего. Были выбраны строки, ранг которых меньше или равен CNT + 1; т. е. все служащие каждого отдела плюс один (этот дополнительный служащий – служащий с первым рангом в своем DEPTNO). В эту дополнительную строку и будет помещено значение DEPTNO. С помощью функции DECODE (старая функция Oracle, являющаяся в той или иной мере эквивалентом выражению CASE) мы обрабатываем значение RN и помещаем значение DEPTNO в результирующее множество. Служащий, находившийся в первой позиции (на основании значения RN), по-прежнему присутствует в результирующем множестве, но теперь его имя указано последним в списке (поскольку порядок не имеет значения, это не представляет проблемы). Вот довольно подробно, как работает нижняя часть UNION ALL.

В верхней части UNION ALL происходит то же самое, что и в нижней, поэтому останавливаться на ней нет необходимости. Давайте лучше рассмотрим результирующее множество, возвращаемое при совмещении запросов:

```
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
       ) x,
       (select level id from dual connect by level <= 2) y
       )
 where rn <= cnt+1
```

```
union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
       )
 where rn <= cnt+1
```

```
FLAG1      FLAG2 IT_DEPT
----- ---------- ----------------------
    1          0 100
    1          0   JONES
    1          0   TONEY
    1          0   HOPKINS
    1          0 200
    1          0   P.WHITAKER
    1          0   MARCIANO
    1          0   ROBINSON
    1          0   MORALES
    1          0 300
    1          0   WRIGHT
    1          0   J.TAYLOR
    1          0   LACY
    1          1 400
    1          1   MAYWEATHER
    1          1   CASTILLO
    1          1   MARQUEZ
    1          1   MOSLEY
    1          1   CORRALES
    1          1 500
    1          1   CALZAGHE
    1          1   GATTI
    1          1 600
    1          1   HAGLER
    1          1   HEARNS
    1          1   FRAZIER
    1          1   LAMOTTA
    1          1 700
    1          1   JUDAH
    1          1   MARGARITO
    1          1   GUINN
```

На данный момент назначение FLAG1 непонятно, но можно заметить, что FLAG2 показывает, в результате выполнения какой части UNION ALL получена строка (0 – верхней части, 1 – нижней).

Следующий шаг – поместить состоящее из отдельных групп результирующее множество во вложенный запрос и вычислить текущую сумму по FLAG1 (наконец его предназначение раскрыто!), который будет играть роль ранга для каждой строки в каждой группе. Результаты ранжирования (текущего суммирования) показаны ниже:

```
select sum(flag1)over(partition by flag2
                          order by flag1,rownum) flag,
       it_dept, flag2
  from (
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
       ) x,
       (select level id from dual connect by level <= 2) y
       )
 where rn <= cnt+1
union all
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
       )
 where rn <= cnt+1
       ) tmp1
```

```
FLAG IT_DEPT              FLAG2
---- --------------- ----------
   1 100                      0
   2   JONES                  0
   3   TONEY                  0
   4   HOPKINS                0
   5 200                      0
   6   P.WHITAKER             0
   7   MARCIANO               0
   8   ROBINSON               0
   9   MORALES                0
```

```
 10 300                    0
 11   WRIGHT               0
 12   J.TAYLOR             0
 13   LACY                 0
  1 400                    1
  2   MAYWEATHER           1
  3   CASTILLO             1
  4   MARQUEZ              1
  5   MOSLEY               1
  6   CORRALES             1
  7 500                    1
  8   CALZAGHE             1
  9   GATTI                1
 10 600                    1
 11   HAGLER               1
 12   HEARNS               1
 13   FRAZIER              1
 14   LAMOTTA              1
 15 700                    1
 16   JUDAH                1
 17   MARGARITO            1
 18   GUINN                1
```

И последний шаг (наконец!) – развернуть значения, возвращаемые TMP1, по FLAG2, группируя при этом по FLAG (текущей сумме, сгенерированной в TMP1). Результаты TMP1 помещаются во вложенный запрос и разворачиваются (все это оформлено как окончательный вложенный запрос TMP2). Окончательное решение и результирующее множество показаны ниже:

```
select max(decode(flag2,0,it_dept)) research,
       max(decode(flag2,1,it_dept)) apps
  from (
select sum(flag1)over(partition by flag2
                          order by flag1,rownum) flag,
       it_dept, flag2
  from (
select 1 flag1, 0 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_research
       ) x,
       (select level id from dual connect by level <= 2) y
       )
 where rn <= cnt+1
union all
```

```
select 1 flag1, 1 flag2,
       decode(rn,1,to_char(deptno),'  '||ename) it_dept
  from (
select x.*, y.id,
       row_number()over(partition by x.deptno order by y.id) rn
  from (
select deptno deptno,
       ename,
       count(*)over(partition by deptno) cnt
  from it_apps
       ) x,
       (select level id from dual connect by level <= 2) y
       )
 where rn <= cnt+1
       ) tmp1
       ) tmp2
 group by flag
```

```
RESEARCH             APPS
-------------------- ---------------
100                  400
  JONES                MAYWEATHER
  TONEY                CASTILLO
  HOPKINS              MARQUEZ
200                    MOSLEY
  P.WHITAKER           CORRALES
  MARCIANO           500
  ROBINSON             CALZAGHE
  MORALES              GATTI
300                  600
  WRIGHT               HAGLER
  J.TAYLOR             HEARNS
  LACY                 FRAZIER
                       LAMOTTA
                     700
                       JUDAH
                       MARGARITO
                       GUINN
```

Преобразование скалярного подзапроса в составной подзапрос (Oracle)

Задача

Хотелось бы обойти ограничение о возвращении скалярным подзапросом всего одного значения. Например, при попытке выполнить следующий запрос:

```
select e.deptno,
       e.ename,
       e.sal,
```

```
       (select d.dname,d.loc,sysdate today
          from dept d
         where e.deptno=d.deptno)
  from emp e
```

получаем ошибку, потому что подзапросы в списке оператора SELECT могут возвращать только одно значение.

Решение

Надо сказать, эта задача довольно нереалистична, потому что простое объединение таблиц EMP и DEPT позволило бы извлекать из DEPT сколько угодно значений. Тем не менее наша задача – сосредоточиться на технике и понять, как применять ее в сценарии, который может оказаться полезным. Обойти требование о возвращении одного значения при размещении оператора SELECT в SELECT (скалярный подзапрос) можно, воспользовавшись преимуществами объектных типов Oracle. Просто определяем объект с несколькими атрибутами и затем работаем с ним как с единой сущностью или ссылаемся на каждый его элемент в отдельности. Фактически правило не нарушено. Получено одно значение, объект, который содержит множество атрибутов.

В этом решении используется следующий объектный тип:

```
create type generic_obj
    as object (
    val1 varchar2(10),
    val2 varchar2(10),
    val3 date
 );
```

Имея в своем распоряжении этот тип, можно выполнить следующий запрос:

```
 1  select x.deptno,
 2         x.ename,
 3         x.multival.val1 dname,
 4         x.multival.val2 loc,
 5         x.multival.val3 today
 6    from (
 7  select e.deptno,
 8         e.ename,
 9         e.sal,
10         (select generic_obj(d.dname,d.loc,sysdate+1)
11            from dept d
12           where e.deptno=d.deptno) multival
13    from emp e
14         ) x

DEPTNO ENAME      DNAME      LOC        TODAY
------ ---------- ---------- ---------- -----------
    20 SMITH      RESEARCH   DALLAS     12-SEP-2005
    30 ALLEN      SALES      CHICAGO    12-SEP-2005
```

```
    30 WARD       SALES      CHICAGO    12-SEP-2005
    20 JONES      RESEARCH   DALLAS     12-SEP-2005
    30 MARTIN     SALES      CHICAGO    12-SEP-2005
    30 BLAKE      SALES      CHICAGO    12-SEP-2005
    10 CLARK      ACCOUNTING NEW YORK   12-SEP-2005
    20 SCOTT      RESEARCH   DALLAS     12-SEP-2005
    10 KING       ACCOUNTING NEW YORK   12-SEP-2005
    30 TURNER     SALES      CHICAGO    12-SEP-2005
    20 ADAMS      RESEARCH   DALLAS     12-SEP-2005
    30 JAMES      SALES      CHICAGO    12-SEP-2005
    20 FORD       RESEARCH   DALLAS     12-SEP-2005
    10 MILLER     ACCOUNTING NEW YORK   12-SEP-2005
```

Обсуждение

Ключ к решению – использовать функцию-конструктор объекта (по умолчанию имя функции-конструктора совпадает с именем объекта). Поскольку объект является единичным скалярным значением, правило скалярных подзапросов не нарушается, как видно из следующего примера:

```
select e.deptno,
       e.ename,
       e.sal,
       (select generic_obj(d.dname,d.loc,sysdate-1)
          from dept d
         where e.deptno=d.deptno) multival
  from emp e
```

```
DEPTNO ENAME    SAL MULTIVAL(VAL1, VAL2, VAL3)
------ ------ ----- ------------------------------------------------------
    20 SMITH    800 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
    30 ALLEN   1600 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
    30 WARD    1250 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
    20 JONES   2975 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
    30 MARTIN  1250 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
    30 BLAKE   2850 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
    10 CLARK   2450 GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2005')
    20 SCOTT   3000 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
    10 KING    5000 GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2005')
    30 TURNER  1500 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
    20 ADAMS   1100 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
    30 JAMES    950 GENERIC_OBJ('SALES', 'CHICAGO', '12-SEP-2005')
    20 FORD    3000 GENERIC_OBJ('RESEARCH', 'DALLAS', '12-SEP-2005')
    10 MILLER  1300 GENERIC_OBJ('ACCOUNTING', 'NEW YORK', '12-SEP-2005')
```

Следующий шаг – поместить запрос во вложенный запрос и извлечь атрибуты.

Одно важное замечание: в Oracle, в отличие от других СУБД, не обязательно присваивать имена вложенным запросам. Однако в данном конкретном случае дать имя вложенному запросу необходимо, иначе мы не сможем ссылаться на атрибуты объекта.

Синтаксический разбор сериализованных данных в строки таблицы

Задача

Имеются сериализованные данные (хранящиеся в виде строк), которые необходимо разобрать и возвратить в виде строк таблицы. Например, хранятся следующие данные:

```
STRINGS
-----------------------------------
entry:stewiegriffin:lois:brian:
entry:moe::sizlack:
entry:petergriffin:meg:chris:
entry:willie:
entry:quagmire:mayorwest:cleveland:
entry:::flanders:
entry:robo:tchi:ken:
```

Требуется преобразовать эти сериализованные строки в следующее результирующее множество:

```
VAL1            VAL2            VAL3
--------------- --------------- ---------------
moe                             sizlack
petergriffin    meg             chris
quagmire        mayorwest       cleveland
robo            tchi            ken
stewiegriffin   lois            brian
willie
                                flanders
```

Решение

В каждой сериализованной строке этого примера может храниться до трех значений, разделенных двоеточиями. Но в строке может быть и меньше трех записей. В этом случае необходимо быть внимательным, чтобы разместить доступные записи в соответствующих столбцах результирующего множества. Например, рассмотрим следующую строку:

```
entry:::flanders:
```

Эта строка представляет запись, в которой пропущены первые два значения и присутствует только третье. Поэтому, посмотрев на результирующее множество, приведенное в разделе «Задача», можно увидеть, что в строке для «flanders» в столбцах VAL1 и VAL2 располагаются значения NULL.

Ключ к решению – обход строки и сопутствующий ему синтаксический разбор с последующим простым разворачиванием. В данном решении будем работать со строками из представления V, описание которого представлено ниже. В примере используется синтаксис Oracle, но поскольку, кроме функций для синтаксического разбора строк,

здесь ничего больше не требуется, очень просто трансформировать решение под другие платформы:

```
create view V
    as
select 'entry:stewiegriffin:lois:brian:' strings
  from dual
 union all
select 'entry:moe::sizlack:'
  from dual
 union all
select 'entry:petergriffin:meg:chris:'
  from dual
 union all
select 'entry:willie:'
  from dual
 union all
select 'entry:quagmire:mayorwest:cleveland:'
  from dual
 union all
select 'entry:::flanders:'
  from dual
 union all
select 'entry:robo:tchi:ken:'
  from dual
```

При использовании представления V как источника данных, подвергающихся впоследствии синтаксическому разбору, решение выглядит следующим образом:

```
 1  with cartesian as (
 2  select level id
 3    from dual
 4   connect by level <= 100
 5  )
 6  select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
 7         max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
 8         max(decode(id,3,substr(strings,p1+1,p2-1))) val3
 9    from (
10  select v.strings,
11         c.id,
12         instr(v.strings,':',1,c.id) p1,
13         instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
14    from v, cartesian c
15   where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
16         )
17   group by strings
18   order by 1
```

Обсуждение

Первый шаг – обход сериализованных строк:

```
with cartesian as (
select level id
  from dual
 connect by level <= 100
)
select v.strings,
       c.id
  from v,cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
```

```
STRINGS                            ID
---------------------------------- ---
entry:::flanders:                   1
entry:::flanders:                   2
entry:::flanders:                   3
entry:moe::sizlack:                 1
entry:moe::sizlack:                 2
entry:moe::sizlack:                 3
entry:petergriffin:meg:chris:       1
entry:petergriffin:meg:chris:       3
entry:petergriffin:meg:chris:       2
entry:quagmire:mayorwest:cleveland: 1
entry:quagmire:mayorwest:cleveland: 3
entry:quagmire:mayorwest:cleveland: 2
entry:robo:tchi:ken:                1
entry:robo:tchi:ken:                2
entry:robo:tchi:ken:                3
entry:stewiegriffin:lois:brian:     1
entry:stewiegriffin:lois:brian:     3
entry:stewiegriffin:lois:brian:     2
entry:willie:                       1
```

Следующий шаг – используя функцию INSTR, находим числовую позицию каждого двоеточия в каждой строке. Поскольку все значения, которые должны быть извлечены, заключены в два двоеточия, числовым значениям присвоены псевдонимы P1 и P2, что означает «position 1» и «position 2» соответственно:

```
with cartesian as (
select level id
  from dual
 connect by level <= 100
)
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
  from v,cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
 order by 1
```

```
STRINGS                            ID         P1         P2
---------------------------------- --- ---------- ----------
```

```
entry:::flanders:                    1          6          1
entry:::flanders:                    2          7          1
entry:::flanders:                    3          8          9
entry:moe::sizlack:                  1          6          4
entry:moe::sizlack:                  2         10          1
entry:moe::sizlack:                  3         11          8
entry:petergriffin:meg:chris:        1          6         13
entry:petergriffin:meg:chris:        3         23          6
entry:petergriffin:meg:chris:        2         19          4
entry:quagmire:mayorwest:cleveland:  1          6          9
entry:quagmire:mayorwest:cleveland:  3         25         10
entry:quagmire:mayorwest:cleveland:  2         15         10
entry:robo:tchi:ken:                 1          6          5
entry:robo:tchi:ken:                 2         11          5
entry:robo:tchi:ken:                 3         16          4
entry:stewiegriffin:lois:brian:      1          6         14
entry:stewiegriffin:lois:brian:      3         25          6
entry:stewiegriffin:lois:brian:      2         20          5
entry:willie:                        1          6          7
```

Теперь, когда нам известны числовые позиции каждой пары двоеточий в каждой строке, просто передаем эту информацию в функцию SUBSTR для извлечения значений. Поскольку должно быть создано результирующее множество с тремя столбцами, используем функцию DECODE, чтобы вычислить ID из декартова произведения:

```
with cartesian as (
select level id
  from dual
 connect by level <= 100
)
select decode(id,1,substr(strings,p1+1,p2-1)) val1,
       decode(id,2,substr(strings,p1+1,p2-1)) val2,
       decode(id,3,substr(strings,p1+1,p2-1)) val3
  from (
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
  from v,cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
       )
 order by 1
```

```
VAL1            VAL2            VAL3
--------------- --------------- --------------
moe
petergriffin
quagmire
robo
stewiegriffin
willie
```

```
lois

meg
mayorwest

tchi
                brian
                sizlack
                chris
                cleveland
                flanders
                ken
```

И, наконец, чтобы создать пригодное для восприятия человеком результирующее множество, к возвращенным SUBSTR значениям применяем агрегатную функцию, проводя при этом группировку по ID:

```
with cartesian as (
select level id
  from dual
 connect by level <= 100
)
select max(decode(id,1,substr(strings,p1+1,p2-1))) val1,
       max(decode(id,2,substr(strings,p1+1,p2-1))) val2,
       max(decode(id,3,substr(strings,p1+1,p2-1))) val3
  from (
select v.strings,
       c.id,
       instr(v.strings,':',1,c.id) p1,
       instr(v.strings,':',1,c.id+1)-instr(v.strings,':',1,c.id) p2
  from v,cartesian c
 where c.id <= (length(v.strings)-length(replace(v.strings,':')))-1
       )
 group by strings
 order by 1
```

```
VAL1            VAL2            VAL3
--------------- --------------- -----------
moe                             sizlack
petergriffin    meg             chris
quagmire        mayorwest       cleveland
robo            tchi            ken
stewiegriffin   lois            brian
willie
                                flanders
```

Определение доли от целого в процентном выражении

Задача

Требуется вывести множество числовых значений, представив каждое из них как долю от целого в процентном выражении. Например, стоит

задача получить в системе Oracle результирующее множество, отражающее распределение заработных плат по должностям, чтобы можно было увидеть, какая из позиций JOB обходится компании дороже всего. Кроме того, чтобы данные были истолкованы правильно, должно быть приведено количество служащих, занимающих каждую из должностей. Ожидается получить следующее результирующее множество:

```
JOB         NUM_EMPS PCT_OF_ALL_SALARIES
--------- ---------- ------------------
CLERK              4                 14
ANALYST            2                 20
MANAGER            3                 28
SALESMAN           4                 19
PRESIDENT          1                 17
```

Как видите, если бы в отчет не было включено количество служащих, могло бы показаться, что президент получает самую маленькую заработную плату. И только информация о том, что ее получает один человек, обеспечивает представление о том, что означают эти 17% от общей суммы заработных плат на самом деле.

Решение

Только Oracle обеспечивает достойное решение этой задачи, в котором используется встроенная функция RATIO_TO_REPORT. Чтобы подсчитать процентные соотношения в других базах данных, можно использовать деление, как показано в разделе «Вычисление доли от целого в процентном выражении» в главе 7:

```
1  select job,num_emps,sum(round(pct)) pct_of_all_salaries
2    from (
3  select job,
4         count(*)over(partition by job) num_emps,
5         ratio_to_report(sal)over()*100 pct
6    from emp
7         )
8   group by job,num_emps
```

Обсуждение

Первый шаг – с помощью оконной функции COUNT OVER получить количество служащих для каждого значения JOB. Затем, используя RATIO_TO_REPORT, возвратить долю суммарной заработной платы для каждой должности от общей заработной платы (в виде десятичной дроби):

```
select job,
       count(*)over(partition by job) num_emps,
       ratio_to_report(sal)over()*100 pct
  from emp

JOB         NUM_EMPS        PCT
--------- ---------- ----------
```

```
ANALYST            2 10.3359173
ANALYST            2 10.3359173
CLERK              4 2.75624462
CLERK              4 3.78983635
CLERK              4  4.4788975
CLERK              4 3.27304048
MANAGER            3 10.2497847
MANAGER            3 8.44099914
MANAGER            3 9.81912145
PRESIDENT          1 17.2265289
SALESMAN           4 5.51248923
SALESMAN           4 4.30663221
SALESMAN           4 5.16795866
SALESMAN           4 4.30663221
```

Последний шаг – посредством агрегатной функции SUM суммировать возвращенные RATIO_TO_REPORT значения. Не забудьте группировать по JOB и NUM_EMPS. Умножаем на 100, чтобы выражение доли в процентах было представлено целым числом (например, чтобы 25% были представлены как 25, а не 0,25):

```
select job,num_emps,sum(round(pct)) pct_of_all_salaries
  from (
select job,
       count(*)over(partition by job) num_emps,
       ratio_to_report(sal)over()*100 pct
  from emp
       )
 group by job,num_emps
```

```
JOB         NUM_EMPS PCT_OF_ALL_SALARIES
--------- ---------- -------------------
CLERK              4                  14
ANALYST            2                  20
MANAGER            3                  28
SALESMAN           4                  19
PRESIDENT          1                  17
```

Создание списка разделенных запятыми значений в Oracle

Задача

Требуется из строк таблицы создать список с разделителями (возможно, используя в качестве разделителей запятые). Например, из таблицы EMP хотелось бы получить следующее результирующее множество:

```
DEPTNO LIST
------ -------------------------------------
    10 MILLER,KING,CLARK
    20 FORD,ADAMS,SCOTT,JONES,SMITH
    30 JAMES,TURNER,BLAKE,MARTIN,WARD,ALLEN
```

Работаем с системой Oracle (Oracle Database 10*g* или более поздней версией) и хотим использовать оператор MODEL.

Решение

В данном решении используются итеративные возможности оператора Oracle MODEL. Методика заключается в ранжировании служащих каждого отдела (по EMPNO, выбор произволен) с помощью ранжирующей функции ROW_NUMBER OVER. Поскольку MODEL обеспечивает доступ к элементам таблицы как к элементам массива, доступ к предыдущим элементам массива можно организовывать путем вычитания рангов. Итак, для каждой строки создаем список, включающий имя каждого служащего плюс имя служащего, согласно рангу являющегося предыдущим по отношению к рассматриваемому:

```
 1  select deptno,
 2         list
 3    from (
 4  select *
 5    from (
 6  select deptno,empno,ename,
 7         lag(deptno)over(partition by deptno
 8                              order by empno) prior_deptno
 9    from emp
10         )
11   model
12     dimension by
13     (
14       deptno,
15       row_number()over(partition by deptno order by empno) rn
16     )
17     measures
18     (
19       ename,
20       prior_deptno,cast(null as varchar2(60)) list,
21       count(*)over(partition by deptno) cnt,
22       row_number()over(partition by deptno order by empno) rnk
23     )
24     rules
25     (
26       list[any,any]
27       order by deptno,rn = case when prior_deptno[cv(),cv()] is null
28                                 then ename[cv(),cv()]
29                                 else ename[cv(),cv()]||','||
30                                      list[cv(),rnk[cv(),cv()]-1]
31                            end
32     )
33     )
34   where cnt = rn
```

Обсуждение

Первый шаг – используя функцию LAG OVER, получить значение DEPTNO предыдущего служащего (сортировка по EMPNO). Результаты группируются по DEPTNO, поэтому для первого служащего в отделе (соответственно EMPNO) будет возвращено значение NULL, для всех остальных – значение DEPTNO. Результаты следующие:

```
select deptno,empno,ename,
       lag(deptno)over(partition by deptno
                           order by empno) prior_deptno
  from emp

DEPTNO      EMPNO ENAME  PRIOR_DEPTNO
------ ---------- ------ ------------
    10       7782 CLARK
    10       7839 KING             10
    10       7934 MILLER           10
    20       7369 SMITH
    20       7566 JONES            20
    20       7788 SCOTT            20
    20       7876 ADAMS            20
    20       7902 FORD             20
    30       7499 ALLEN
    30       7521 WARD             30
    30       7654 MARTIN           30
    30       7698 BLAKE            30
    30       7844 TURNER           30
    30       7900 JAMES            30
```

Следующий шаг – проанализировать подоператор MEASURES оператора MODEL. Элементы списка MEASURES являются массивами:

ENAME

Массив всех значений ENAME таблицы EMP.

PRIOR_DEPTNO

Массив значений, возвращенных оконной функцией LAG OVER.

CNT

Массив количества служащих по отделам.

RNK

Массив рангов (по EMPNO) служащих в каждом DEPTNO.

Индексами массивов являются DEPTNO и RN (значение, возвращенное ранжирующей функцией ROW_NUMBER OVER в подоператоре DIMENSION BY). Чтобы увидеть содержимое всех этих массивов, закомментируем код подоператора RULES оператора MODEL и выполним запрос в следующем виде:

```
select *
  from (
```

```
select deptno,empno,ename,
       lag(deptno)over(partition by deptno
                           order by empno) prior_deptno
  from emp
       )
 model
   dimension by
   (
     deptno,
     row_number()over(partition by deptno order by empno) rn
   )
   measures
   (
     ename,
     prior_deptno,cast(null as varchar2(60)) list,
     count(*)over(partition by deptno) cnt,
     row_number()over(partition by deptno order by empno) rnk
   )
   rules
   (
/*
     list[any,any]
     order by deptno,rn = case when prior_deptno[cv(),cv()] is null
                               then ename[cv(),cv()]
                               else ename[cv(),cv()]||','||
                                    list[cv(),rnk[cv(),cv()]-1]
                          end
*/
   )
 order by 1
```

```
DEPTNO  RN ENAME  PRIOR_DEPTNO LIST       CNT  RNK
------ --- ------ ------------ ---------- --- ----
    10   1 CLARK                           3    1
    10   2 KING             10             3    2
    10   3 MILLER           10             3    3
    20   1 SMITH                           5    1
    20   2 JONES            20             5    2
    20   4 ADAMS            20             5    4
    20   5 FORD             20             5    5
    20   3 SCOTT            20             5    3
    30   1 ALLEN                           6    1
    30   6 JAMES            30             6    6
    30   4 BLAKE            30             6    4
    30   3 MARTIN           30             6    3
    30   5 TURNER           30             6    5
    30   2 WARD             30             6    2
```

Теперь, когда точно известно назначение каждого элемента оператора MODEL, перейдем к подоператору RULES. Посмотрим на выражение CASE. Как видим, в нем происходит вычисление текущего значения PRIOR_DEPTNO. Значение NULL говорит о том, что в текущий мас-

сив LIST должен быть возвращен первый служащий соответственно рангам по DEPTNO и ENAME. Если значение PRIOR_DEPTNO не NULL, присоединяем значение LIST предыдущего служащего к имени текущего служащего (массив ENAME) и затем возвращаем этот результат как LIST текущего служащего. Выполнение этой операции выражения CASE для каждой строки с заданным значением DEPTNO обеспечивает итеративное создание списка разделенных запятыми значений (comma-separated values, CSV). Промежуточные результаты можно увидеть в следующем примере:

```
select deptno,
       list
  from (
select *
  from (
select deptno,empno,ename,
       lag(deptno)over(partition by deptno
                         order by empno) prior_deptno
  from emp
       )
 model
   dimension by
   (
     deptno,
     row_number()over(partition by deptno order by empno) rn
   )
   measures
   (
     ename,
     prior_deptno,cast(null as varchar2(60)) list,
     count(*)over(partition by deptno) cnt,
     row_number()over(partition by deptno order by empno) rnk
   )
   rules
   (
     list[any,any]
     order by deptno,rn = case when prior_deptno[cv(),cv()] is null
                               then ename[cv(),cv()]
                               else ename[cv(),cv()]||','||
                                    list[cv(),rnk[cv(),cv()]-1]
                          end
   )
   )
```

```
DEPTNO LIST
------ ---------------------------------------
    10 CLARK
    10 KING,CLARK
    10 MILLER,KING,CLARK
    20 SMITH
    20 JONES,SMITH
```

```
20 SCOTT,JONES,SMITH
20 ADAMS,SCOTT,JONES,SMITH
20 FORD,ADAMS,SCOTT,JONES,SMITH
30 ALLEN
30 WARD,ALLEN
30 MARTIN,WARD,ALLEN
30 BLAKE,MARTIN,WARD,ALLEN
30 TURNER,BLAKE,MARTIN,WARD,ALLEN
30 JAMES,TURNER,BLAKE,MARTIN,WARD,ALLEN
```

Последний шаг – выбрать в каждом отделе только последнего служащего, что гарантирует получение полного списка служащих отдела, в котором они перечислены через запятую. Чтобы возвратить только полные списки служащих каждого отдела, используем значения массивов CNT и RN. Поскольку RN представляет ранги служащих каждого отдела по EMPNO, последним служащим в отделе будет служащий, для которого CNT = RN, как показывает следующий пример:

```
select deptno,
       list
  from (
select *
  from (
select deptno,empno,ename,
       lag(deptno)over(partition by deptno
                           order by empno) prior_deptno
  from emp
       )
 model
   dimension by
   (
     deptno,
     row_number()over(partition by deptno order by empno) rn
   )
   measures
   (
     ename,
     prior_deptno,cast(null as varchar2(60)) list,
     count(*)over(partition by deptno) cnt,
     row_number()over(partition by deptno order by empno) rnk
   )
   rules
   (
     list[any,any]
     order by deptno,rn = case when prior_deptno[cv(),cv()] is null
                               then ename[cv(),cv()]
                               else ename[cv(),cv()]||','||
                                    list[cv(),rnk[cv(),cv()]-1]
                          end
   )
   )
```

```
 where cnt = rn

DEPTNO LIST
------ ---------------------------------------
    10 MILLER, KING, CLARK
    20 FORD, ADAMS, SCOTT, JONES, SMITH
    30 JAMES, TURNER, BLAKE, MARTIN, WARD, ALLEN
```

Выбор текста, не соответствующего шаблону (Oracle)

Задача

Имеется текстовое поле, содержащее некоторый структурированный текст (например, телефонные номера), и требуется найти неправильно структурированные элементы. Например, есть следующие данные:

```
select emp_id, text
  from employee_comment

EMP_ID     TEXT
---------- ------------------------------------------------------------
7369       126 Varnum, Edmore MI 48829, 989 313-5351
7499       1105 McConnell Court
           Cedar Lake MI 48812
           Home: 989-387-4321
           Cell: (237) 438-3333
```

и необходимо составить список строк с неверно форматированными телефонными номерами. Например, следующая строка должна войти в этот список, потому что в ней в телефонном номере используется два разных символа-разделителя:

```
7369       126 Varnum, Edmore MI 48829, 989 313-5351
```

Действительными считаются телефонные номера, в которых в обоих случаях используется один и тот же разделитель.

Решение

Решение этой задачи состоит из нескольких частей:

1. Описываем множество истинных телефонных номеров.
2. Исключаем из рассмотрения все правильно форматированные телефонные номера.
3. Проверяем, остались ли еще какие-либо телефонные номера. Если остались, значит, именно они форматированы неправильно.

Следующее решение выгодно использует регулярные выражения, введенные в Oracle Database 10*g*:

```
select emp_id, text
from employee_comment
where regexp_like(text, '[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}')
```

```
  and regexp_like(
         regexp_replace(text,
            '[0-9]{3}([-. ])[0-9]{3}\1[0-9]{4}','***'),
         '[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}')

    EMP_ID TEXT
---------- ------------------------------------------------------------
      7369 126 Varnum, Edmore MI 48829, 989 313-5351
      7844 989-387.5359
      9999 906-387-1698, 313-535.8886
```

Каждая из этих строк содержит, по крайней мере, один телефонный номер, отформатированный неправильно.

Обсуждение

Ключ к этому решению в определении «истинного телефонного номера». Поскольку телефонные номера хранятся в поле комментариев, любой текст в поле может быть истолкован как недействительный телефонный номер. Необходимо сузить рассматриваемое множество до более разумного набора значений. Например, не хотелось бы видеть в выводе следующую строку:

```
    EMP_ID TEXT
---------- ------------------------------------------------------------
      7900 Cares for 100-year-old aunt during the day. Schedule only
           for evening and night shifts.
```

Понятно, что в этой строке вообще нет никакого телефонного номера, тем более неверно форматированного. Для нас это очевидно. Вопрос в том, как заставить СУБД «увидеть» это. Думаю, ответ вам понравится. Пожалуйста, дочитайте до конца.

Этот рецепт взят из статьи Джонатана Генника (по его разрешению) «Regular Expression Anti-Patterns», которую можно найти по адресу: *http://gennick.com/antiregex.htm*.

Для определения множества «истинных» телефонных номеров в решении используется Pattern A (шаблон A):

```
Pattern A: [0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}
```

Pattern A проверяет наличие двух групп, состоящих из трех цифр, за которыми следует группа из четырех цифр. В качестве разделителей между группами могут выступать тире (–), точка (.) или пробел. Можно было бы создать более сложный шаблон, например принять во внимание и семизначные телефонные номера. Но не будем уходить в сторону. Наша задача состоит в том, что мы должны каким-то образом определить множество возможных телефонных номеров. В данном случае это множество определено Pattern A. Можно использовать другой Pattern A, но решение в целом будет по-прежнему работоспособным.

В данном решении Pattern A размещается в предикате WHERE, что гарантирует рассмотрение только тех строк, в которых потенциально присутствуют телефонные номера (согласно шаблону!):

```
select emp_id, text
  from employee_comment
 where regexp_like(text, '[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}')
```

Далее необходимо определить, как выглядит «действительный» телефонный номер. В решении для этого применяется Pattern B:

```
Pattern B: [0-9]{3}([-. ])[0-9]{3}\1[0-9]{4}
```

На этот раз в шаблоне используется \1, что указывает на первое подвыражение. Если символ соответствует ([-.]), он также должен соответствовать \1. Pattern B описывает действительные телефонные номера, которые должны быть исключены из рассмотрения (поскольку они не являются ошибочными). В решении исключение правильно форматированных телефонных номеров осуществляется через вызов функции REGEXP_REPLACE:

```
        regexp_replace(text,
           '[0-9]{3}([-. ])[0-9]{3}\1[0-9]{4}','***'),
```

Вызов REGEXP_REPLACE выполняется в предикате WHERE. Все правильно форматированные телефонные номера замещаются строкой из трех звездочек. Опять же Pattern B может быть любым, главное, чтобы он описывал допустимый формат номеров.

После того как все правильно форматированные телефонные номера замещены строками из трех звездочек (***), все оставшиеся «истинные» телефонные номера должны быть по определению неправильно отформатированными. Применяя функцию REGEXP_LIKE к выводу функции REGEXP_REPLACE, проверяем, остались ли неправильно форматированные номера:

```
and regexp_like(
       regexp_replace(text,
          '[0-9]{3}([-. ])[0-9]{3}\1[0-9]{4}','***'),
       '[0-9]{3}[-. ][0-9]{3}[-. ][0-9]{4}')
```

Этот рецепт было бы тяжело реализовать, если бы не возможности сопоставления шаблонов, возникшие относительно недавно в Oracle с появлением регулярных выражений. В частности, этот рецепт полагается на REGEXP_REPLACE. Другие базы данных (например, PostgreSQL) реализуют поддержку регулярных выражений. Но, насколько мне известно, только Oracle поддерживает функциональность поиска и замены на основании регулярных выражений, на которых построено данное решение.

Преобразование данных с помощью вложенного запроса

Задача

Имеется таблица, в одном из столбцов которой могут храниться как числовые, так и символьные данные. В этой же таблице есть другой столбец, показывающий, какие именно данные хранятся в первом столбце. Требуется с помощью подзапроса выбрать только числовые данные:

```
select *
  from ( select flag, to_number(num) num
       from subtest
       where flag in ('A', 'C') )
       where num > 0
```

К сожалению, выполнение этого запроса часто (но не всегда!) приводит к следующему сообщению об ошибке:

```
ERROR:
ORA-01722: invalid number
```

Решение

Одно из решений – заставить вложенный запрос полностью завершиться до выполнения внешнего выражения SELECT. Это можно сделать, по крайней мере, в Oracle, включая псевдостолбец номера строки в список внутреннего SELECT:

```
select *
  from ( select rownum, flag, to_number(num) num
       from subtest
       where flag in ('A', 'C') )
       where num > 0
```

В разделе «Обсуждение» объясняется, почему это решение работает.

Обсуждение

Причина возникновения ошибки в запросе, рассматриваемом в разделе «Задача», в том, что некоторые оптимизаторы объединяют внутренний и внешний запросы. Хотя все выглядит так, будто сначала выполняется внутренний запрос, удаляющий все нечисловые значения NUM, на самом деле происходит следующее:

```
select flag, to_number(num) num
from subtest
where to_number(num) > 0 and flag in ('A', 'C');
```

Вот теперь причина ошибки очевидна: строки с нечисловыми значениями в поле NUM *не* отсеиваются перед применением функции TO_NUMBER.

Объединит ли база данных основной и вложенный запросы? Ответ зависит от того, с какой точки зрения подойти: реляционной теории, стандарта SQL или конкретной реализации SQL отдельным производителем. Больше информации можно найти по адресу *http://gennick.com/madness.html*.

В данном случае проблема решена, по крайней мере, в Oracle, благодаря введению ROWNUM в список оператора SELECT внутреннего запроса. ROWNUM – это функция, возвращающая последовательно возрастающее число для каждой строки, *возвращаемой запросом*. Здесь важны последние слова. Последовательно возрастающее число, получившее название *номер строки*, не может вычисляться вне контекста возвращения строки запросом. Таким образом, Oracle вынужден материализовать результат подзапроса, что означает, что Oracle вынужден выполнить сначала подзапрос, чтобы возвратить строки из этого подзапроса для правильного присвоения номеров строк. Итак, использование в запросе ROWNUM – один из механизмов заставить Oracle полностью выполнить подзапрос перед выполнением основного запроса (т. е. слияние запросов не допускается). Если требуется установить принудительный порядок выполнения подзапроса в какой-то другой СУБД, не Oracle, проверьте, не поддерживает ли эта база данных функции, аналогичной ROWNUM Oracle.

Проверка существования значения в группе

Задача

Требуется создать для строки логический флаг, указывающий на наличие в любой строке ее группы определенного значения. Возьмем для примера студента, который сдает определенное количество экзаменов за определенный промежуток времени. Студент будет сдавать три экзамена за три месяца. Если он сдает один из этих экзаменов, требование считается удовлетворенным, и для выражения этого факта должен быть возвращен флаг. Если студент не сдает ни одного из трех экзаменов за трехмесячный период, должен быть возвращен дополнительный флаг, указывающий и на этот факт. Рассмотрим следующий пример (для создания строк используется синтаксис Oracle; для DB2 и SQL Server понадобятся минимальные изменения, поскольку обе СУБД поддерживают оконные функции):

```
create view V
as
select 1 student_id,
       1 test_id,
       2 grade_id,
       1 period_id,
       to_date('02/01/2005','MM/DD/YYYY') test_date,
       0 pass_fail
```

```
  from dual union all
select 1, 2, 2, 1, to_date('03/01/2005','MM/DD/YYYY'),  1 from dual union all
select 1, 3, 2, 1, to_date('04/01/2005','MM/DD/YYYY'),  0 from dual union all
select 1, 4, 2, 2, to_date('05/01/2005','MM/DD/YYYY'),  0 from dual union all
select 1, 5, 2, 2, to_date('06/01/2005','MM/DD/YYYY'),  0 from dual union all
select 1, 6, 2, 2, to_date('07/01/2005','MM/DD/YYYY'),  0 from dual

select *
  from V

STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE   PASS_FAIL
---------- ------- -------- --------- ----------- ---------
         1       1        2         1 01-FEB-2005         0
         1       2        2         1 01-MAR-2005         1
         1       3        2         1 01-APR-2005         0
         1       4        2         2 01-MAY-2005         0
         1       5        2         2 01-JUN-2005         0
         1       6        2         2 01-JUL-2005         0
```

Рассмотрев приведенное выше результирующее множество, видим, что студент должен сдать шесть экзаменов за два трехмесячных периода. Студент сдал один экзамен (1 означает «сдал», 0 – «не сдал»), таким образом, требование удовлетворено для всего первого периода. Поскольку студент не сдал ни одного экзамена в течение второго периода (следующие три месяца), PASS_FAIL равен 0 для всех трех экзаменов. Хотелось бы получить результирующее множество, из которого видно, сдал ли студент хотя бы один экзамен за данный период. В итоге должно быть представлено следующее результирующее множество:

```
STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE   METREQ IN_PROGRESS
---------- ------- -------- --------- ----------- ------ -----------
         1       1        2         1 01-FEB-2005      +           0
         1       2        2         1 01-MAR-2005      +           0
         1       3        2         1 01-APR-2005      +           0
         1       4        2         2 01-MAY-2005      -           0
         1       5        2         2 01-JUN-2005      -           0
         1       6        2         2 01-JUL-2005      -           1
```

В столбце METREQ («met requirement»[1]) представлены значения «+» и «–», обозначающие, выполнил или нет студент требование о сдаче, по крайней мере, одного экзамена за период (три месяца) соответственно. Значение поля IN_PROGRESS должно быть равным 0, если студент уже сдал какой-то экзамен за данный период. Если студент не сдал ни одного экзамента, в поле IN_PROGRESS строки с датой последнего экзамена этого студента будет располагаться значение 1.

Решение

Немного усложняет эту задачу тот факт, что строки необходимо рассматривать не индивидуально, а в группе. Посмотрим на значения

[1] Требование выполнено. – *Примеч. перев.*

PASS_FAIL в разделе «Задача». Если бы записи обрабатывались последовательно, строка за строкой, значение поля METREQ во всех строках, кроме TEST_ID 2, было бы «–», хотя это неправильно. Мы должны обеспечить групповую обработку строк. Оконная функция MAX OVER позволяет без труда определить, сдал ли студент хотя бы один экзамен в течение конкретного периода. После получения этой информации расстановка логических значений – просто вопрос применения выражений CASE:

```
 1  select student_id,
 2         test_id,
 3         grade_id,
 4         period_id,
 5         test_date,
 6         decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
 7         decode( grp_p_f,1,0,
 8                 decode( test_date,last_test,1,0 ) ) in_progress
 9    from (
10  select V.*,
11         max(pass_fail)over(partition by
12                       student_id,grade_id,period_id) grp_p_f,
13         max(test_date)over(partition by
14                       student_id,grade_id,period_id) last_test
15    from V
16         ) x
```

Обсуждение

Ключ к решению – применение оконной функции MAX OVER и возвращение с ее помощью наибольшего значения PASS_FAIL для каждой группы. Поскольку значениями PASS_FAIL могут быть только 1 или 0, в случае, если студент сдал хотя бы один экзамен, MAX OVER возвращает 1 для всей группы. Как это получается, показано ниже:

```
select V.*,
       max(pass_fail)over(partition by
                       student_id,grade_id,period_id) grp_pass_fail
  from V

STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE   PASS_FAIL GRP_PASS_FAIL
---------- ------- -------- --------- ----------- --------- -------------
         1       1        2         1 01-FEB-2005         0             1
         1       2        2         1 01-MAR-2005         1             1
         1       3        2         1 01-APR-2005         0             1
         1       4        2         2 01-MAY-2005         0             0
         1       5        2         2 01-JUN-2005         0             0
         1       6        2         2 01-JUL-2005         0             0
```

Из приведенного выше результирующего множества видно, что студент сдал, по крайней мере, один экзамен в течение первого периода. Таким образом, вся группа получает значение 1 или «pass» (сдал). Следующее требование – если студент не сдал ни одного экзамена за период,

в поле IN_PROGRESS строки с датой последнего экзамента в группе возвратить значение 1. Для этого также может использоваться оконная функция MAX OVER:

```
select V.*,
       max(pass_fail)over(partition by
                     student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
                     student_id,grade_id,period_id) last_test
  from V
```

```
STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE   PASS_FAIL GRP_P_F LAST_TEST
---------- ------- -------- --------- ----------- --------- ------- -----------
         1       1        2         1 01-FEB-2005         0       1 01-APR-2005
         1       2        2         1 01-MAR-2005         1       1 01-APR-2005
         1       3        2         1 01-APR-2005         0       1 01-APR-2005
         1       4        2         2 01-MAY-2005         0       0 01-JUL-2005
         1       5        2         2 01-JUN-2005         0       0 01-JUL-2005
         1       6        2         2 01-JUL-2005         0       0 01-JUL-2005
```

Теперь, когда определено, в какой период студент сдал экзамен и какова дата последнего экзамена, осталось просто поколдовать с форматированием и приукрасить результирующее множество. В окончательном решении для создания столбцов METREQ и IN_PROGRESS используется функция Oracle DECODE (сторонники CASE, кусайте локти). Функция LPAD поможет выровнять значения METREQ по правому краю:

```
select student_id,
       test_id,
       grade_id,
       period_id,
       test_date,
       decode( grp_p_f,1,lpad('+',6),lpad('-',6) ) metreq,
       decode( grp_p_f,1,0,
               decode( test_date,last_test,1,0 ) ) in_progress
  from (
select V.*,
       max(pass_fail)over(partition by
                     student_id,grade_id,period_id) grp_p_f,
       max(test_date)over(partition by
                     student_id,grade_id,period_id) last_test
  from V
       ) x
```

```
STUDENT_ID TEST_ID GRADE_ID PERIOD_ID TEST_DATE   METREQ IN_PROGRESS
---------- ------- -------- --------- ----------- ------ -----------
         1       1        2         1 01-FEB-2005      +           0
         1       2        2         1 01-MAR-2005      +           0
         1       3        2         1 01-APR-2005      +           0
         1       4        2         2 01-MAY-2005      -           0
         1       5        2         2 01-JUN-2005      -           0
         1       6        2         2 01-JUL-2005      -           1
```

A

Оконные функции, краткий обзор

В рецептах данной книги широко используются оконные функции, введенные в стандарт ISO SQL в 2003 году, и собственные оконные функции отдельных производителей. Данное приложение является кратким обзором оконных функций. Оконные функции существенно упрощают решение многих обычно вызывающих сложности (при решении с использованием стандартных механизмов SQL) задач. Полный список доступных оконных функций, их синтаксис и подробное описание их работы вы найдете в документации своей базы данных.

Группировка

Прежде чем перейти к оконным функциям, важно понять механизм группировки в SQL. По моему опыту, идея группировки результатов в SQL является камнем преткновения для многих. Проблемы возникают из-за недопонимания принципов работы оператора GROUP BY и его влияния на результаты запросов.

Если давать простое определение, группировка – это способ организации подобных строк. При использовании GROUP BY в запросе каждая строка результирующего множества является группой и представляет одну или более строк с одинаковыми значениями в одном или более заданных полях. Вот суть группировки.

Если группа – это просто уникальный экземпляр строки, представляющий одну или более строк с таким же значением в определенном столбце (или столбцах), тогда примерами групп в таблице EMP являются *все служащие 10-го отдела* (общее значение, позволяющее объединить этих служащих в одну группу, – DEPTNO=10) или *все клерки* (общее значение, позволяющее объединить этих служащих в одну группу, – JOB='CLERK'). Рассмотрим следующие запросы. Первый показывает всех служащих 10-го отдела; второй группирует служащих

10-го отдела и возвращает о них следующую информацию: количество строк (членов) в группе, наибольшая заработная плата и наименьшая заработная плата:

```
select deptno,ename
  from emp
 where deptno=10

DEPTNO ENAME
------ ----------
    10 CLARK
    10 KING
    10 MILLER

select deptno,
       count(*) as cnt,
       max(sal) as hi_sal,
       min(sal) as lo_sal
  from emp
 where deptno=10
 group by deptno

DEPTNO        CNT     HI_SAL     LO_SAL
------ ---------- ---------- ----------
    10          3       5000       1300
```

Если бы не было возможности сгруппировать служащих 10-го отдела, для получения информации, возвращаемой вторым из приведенных выше запросов, пришлось бы самостоятельно проверять записи, соответствующие этому отделу (это просто, если таких строк всего три, а если этих строк три миллиона?). Итак, почему возникает необходимость в группировке? Причины весьма разнообразны. Может потребоваться представить, сколько существует разных групп или сколько членов (строк) в каждой группе. Как видно из простого примера, показанного выше, группировка позволяет получать информацию о многих строках таблицы, не проверяя их одну за другой.

Определение группы в SQL

В математике основным определением группы является тройка (G, •, e), где G – множество, • – бинарная операция над G, и e – член G. Мы будем использовать это определение как базовое для описания группы в SQL. SQL-группа определяется как пара (G, e), где G – результирующее множество самостоятельного или самодостаточного запроса, в котором используется оператор GROUP BY, e – член G, и выполняются следующие аксиомы:

- Для каждого e в G e является уникальным и представляет один или более экземпляров e.
- Для каждого e в G агрегатная функция COUNT возвращает значение > 0.

В описание SQL-группы включено результирующее множество для подкрепления того факта, что определение группы дается только для работы с запросами. Таким образом, было бы правильным заменить в каждом постулате «е» на «строка», потому что фактически строки в результирующем множестве – это группы.

Поскольку данные свойства являются основополагающими для групп, важно подтвердить их истинность (это мы сделаем на примере некоторых SQL-запросов).

Группы не могут быть пустыми

По определению группа должна включать в себя хотя бы один элемент (или строку). Если это так, можно сказать, что группа не может быть создана из пустой таблицы. Чтобы доказать истинность этого утверждения, пойдем от противного и попытаемся доказать, что оно ложно. В следующем примере создается пустая таблица, и затем выполняются три разных запроса к ней с попыткой создать группы:

```
create table fruits (name varchar(10))

select name
  from fruits
 group by name

(не выбрана ни одна строка)

select count(*) as cnt
  from fruits
 group by name

(не выбрана ни одна строка)

select name, count(*) as cnt
  from fruits
 group by name

(не выбрана ни одна строка)
```

Как видно из этих запросов, из пустой таблицы невозможно создать то, что в SQL считается группой.

Группы уникальны

Теперь давайте подтвердим, что группы, созданные посредством оператора GROUP BY, являются уникальными. В следующем примере в таблицу FRUITS (фрукты) вставляется пять строк, из которых впоследствии создаются группы:

```
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
insert into fruits values ('Oranges')
```

```
insert into fruits values ('Apple')
insert into fruits values ('Peach')

select *
  from fruits

NAME
----------
Oranges
Oranges
Oranges
Apple
Peach

select name
  from fruits
 group by name

NAME
----------
Apple
Oranges
Peach

select name, count(*) as cnt
  from fruits
 group by name

NAME              CNT
---------- ----------
Apple               1
Oranges             3
Peach               1
```

Первый запрос показывает, что запись «Oranges» (апельсины) встречается в таблице FRUITS трижды. Однако второй и третий запросы (с использованием GROUP BY) возвращают только один экземпляр «Oranges». Эти запросы доказывают, что строки в результирующем множестве (*e* в *G*, согласно определению) являются уникальными, и каждое значение NAME представляет один или более экземпляров самого себя в таблице FRUITS.

Вы должны знать, что группы уникальны, и не вводить ключевое слово DISTINCT в список оператора SELECT при использовании GROUP BY в запросах.

Я никоим образом не говорю, что GROUP BY и DISTINCT – это одно и то же. Они представляют две совершенно разные концепции. Я только утверждаю, что элементы, перечисленные в операторе GROUP BY, будут уникальными в результирующем множестве, и что использовать DISTINCT одновременно с GROUP BY будет излишним.

Аксиома Фреге и парадокс Расселла

Аксиома *абстракции* Фреге, основанная на канторовском определении принадлежности к бесконечному или неисчислимому множеству, утверждает, что если есть идентифицирующее свойство, то существует множество, членами+дают этим свойством. Источником неприятностей, как формулирует Роберт Столл, «является неограниченное использование принципа абстракции». Бертран Рассел предложил Готлобу Фреге рассмотреть множество, членами которого являются множества, определяющим свойством которых является то, что они не являются членами самих себя.

Как обозначил Рассел, аксиома абстракции обеспечивает слишком большую свободу, потому что задаются просто условие или свойство, определяющие членство в множестве, а это дает пространство для противоречий. Чтобы лучше описать, как можно выявить противоречия, он придумал «парадокс цирюльника», который звучит так:

> В некотором городишке есть мужской парикмахер, который бреет всех тех и только тех мужчин, которые не бреются самостоятельно. Если это так, кто тогда бреет парикмахера?

Возьмем более конкретный пример и рассмотрим множество, которое может быть описано как:

> *Все члены x в y, которые удовлетворяют определенному условию (P)*

Вот математическая запись этого описания:

```
{x ε y | P(x)}
```

Поскольку обсуждаемое множество рассматривает *только те x в y, которые удовлетворяют условию (P)*, понятнее было бы описать его так: *x является членом y в том и только в том случае, если x удовлетворяет условию (P)*.

А сейчас давайте определим это условие *P*(*x*) как *x не является членом x*:

```
( x ε x )
```

Теперь данное множество определено как *x является членом y в том и только в том случае, если x не член x*:

```
{x ε y | ( x ε x )}
```

Возможно, вы еще не осмыслили четко парадокс Рассела, но задайте себе вопрос: может ли обсуждаемое множество быть членом самого себя? Предположим, что $x = y$, и еще раз посмотрим на приведенное выше множество. Теперь множество можно определить как *y является членом y в том и только в том случае, если y не член y*:

```
{y ε y | ( y ε y )}
```

Попросту говоря, в парадоксе Рассела рассуждается о множестве, одновременно являющемся и не являющемся членом самого себя, что есть противоречие. Интуитивно можно прийти к выводу, что это вообще не является проблемой. В самом деле, как может множество быть членом самого себя? Множество книг ведь не является книгой. Так почему этот парадокс существует, и какую проблему он может представлять? Проблема возникает, если рассматривать более абстрактные приложения теории множеств. Например, парадокс Рассела можно продемонстрировать «на практике» на примере множества всех множеств. Если допустить существование такого понятия, тогда, по его собственному определению, оно должно быть членом самого себя (в конце концов, оно же является множеством всех множеств). Тогда что будет, если применить условие $P(x)$ к множеству всех множеств? Парадокс Рассела утверждает, что множество всех множеств является членом самого себя тогда и только тогда, когда оно не является членом самого себя, – очевидное противоречие.

Если вы заинтересовались, Эрнст Цермело (Ernst Zermelo) разработал аксиому «Схема выделения» (ее иногда называют просто аксиомой выделения или аксиомой подмножеств), чтобы элегантно обойти парадокс Рассела в аксиоматической теории множеств.

Значение COUNT никогда не равно нулю

Запросы и полученные результаты предыдущих разделов подтверждают и последнюю аксиому о том, что агрегатная функция COUNT никогда не возвратит нуль при использовании в запросе с оператором GROUP BY к непустой таблице. В том, что для группы не может быть возвращен нулевой счетчик, не должно быть ничего удивительного. Мы уже доказали, что группа не может быть создана из пустой таблицы, таким образом, в группе должна быть, по крайней мере, одна строка. Если существует хотя бы одна строка, счетчик всегда будет, как минимум, равен 1.

Помните, мы говорим только о совместном использовании функции COUNT с GROUP BY. Запрос к пустой таблице с COUNT без GROUP BY, конечно же, возвратит нуль.

Парадоксы

> «Для автора научного труда не может быть ничего хуже, чем крушение одной из основ его теории после того, когда работа уже завершена... В такое положение меня поставило письмо мр. Бертрана Рассела, которое я получил, когда печать этой книги уже близилась к завершению».

Это слова Готтлоба Фреге в ответ на открытие Бертраном Расселом противоречия аксиоме абстракции Фреге в теории множеств.

Парадоксы часто порождают сценарии, которые, казалось бы, противоречат признанным теориям или идеям. Во многих случаях эти противоречия можно выявить и «обойти», или они распространяются на такое небольшое число контрольных примеров, что могут быть благополучно проигнорированы.

Наверное, вы уже догадались, что смысл всей этой дискуссии о парадоксах в том, что в нашем определении SQL-группы также есть парадокс, и этот парадокс нельзя оставить без внимания. Хотя в данный момент мы рассматриваем группы, в конечном счете, обсуждение касается запросов SQL. В операторе GROUP BY запроса могут располагаться разнообразнейшие значения: константы, выражения или, чаще всего, столбцы таблицы. Но мы расплачиваемся за эту гибкость, поскольку NULL является корректным «значением» в SQL. Значения NULL создают проблемы, так как игнорируются агрегатными функциями. Исходя из этого, что возвратит агрегатная функция COUNT, используемая в запросе с GROUP BY, если таблица состоит из одной строки со значением NULL? Согласно нашему определению при использовании GROUP BY и агрегатной функции COUNT должно быть получено значение >= 1. Но тогда что происходит, если присутствуют значения, игнорируемые такими функциями, как COUNT, и что это означает для нашего определения ГРУППЫ? Рассмотрим следующий пример, раскрывающий парадокс группы NULL (функция COALESCE используется для удобства чтения):

```
select *
  from fruits

NAME
----------
Oranges
Oranges
Oranges
Apple
Peach
```

```
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)
insert into fruits values (null)

select coalesce(name,'NULL') as name
  from fruits

NAME
----------
Oranges
Oranges
Oranges
Apple
Peach
NULL
NULL
NULL
NULL
NULL

select coalesce(name,'NULL') as name,
       count(name) as cnt
  from fruits
 group by name

NAME              CNT
---------- ----------
Apple               1
NULL                0
Oranges             3
Peach               1
```

Казалось бы, присутствие значений NULL в нашей таблице приводит к противоречию с данным нами определением SQL-группы. К счастью, это противоречие не является поводом для беспокойства, потому что парадокс касается скорее вопроса реализации агрегатных функций, а не нашего определения. Рассмотрим последний запрос из приведенных выше; поставленную задачу для него можно сформулировать следующим образом:

> *Подсчитать, сколько раз встречается каждое имя в таблице FRUITS или сколько членов в каждой группе.*

Проанализировав приведенное выше выражение INSERT, можно увидеть, что строк со значениями NULL пять, т. е. существует группа NULL из пяти членов.

Хотя, несомненно, значение NULL обладает свойствами, которые отличают его от других значений, тем не менее это значение, и оно может быть группой.

Тогда как создать запрос, правильно подсчитывающий количество членов в группе значений NULL, таким образом, предоставляющий нам искомую информацию и при этом соответствующий нашему определению группы? Пример ниже показывает, как справиться с парадоксом группы значений NULL:

```
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
```

```
NAME              CNT
---------- ----------
Apple               1
Oranges             3
Peach               1
NULL                5
```

Чтобы обойти парадокс группы значений NULL, используем функцию COUNT(*), а не COUNT(NAME). Агрегатные функции проигнорируют значения NULL, если таковые встретятся в переданных столбцах. Таким образом, чтобы не получить нуль при использовании COUNT, передаем не имена столбцов, а звездочку (*). Это заставит функцию COUNT подсчитывать строки, а не значения столбцов, поэтому есть ли там значения NULL или нет, не важно.

Еще один парадокс связан с аксиомой о том, что каждая группа результирующего множества (для каждого e в G) уникальна. Из-за природы результирующих множеств SQL и таблиц, которые, если быть более точным, являются не множествами, а мультимножествами или «множествами с повторяющимися элементами» (поскольку допускается существование дублирующихся строк), может быть получено результирующее множество с дублирующимися группами. Рассмотрим следующие запросы:

```
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
 union all
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
```

```
NAME              CNT
---------- ----------
Apple               1
Oranges             3
Peach               1
NULL                5
Apple               1
```

```
Oranges             3
Peach               1
NULL                5

select x.*
  from (
select coalesce(name,'NULL') as name,
       count(*) as cnt
  from fruits
 group by name
       ) x,
       (select deptno from dept) y

NAME              CNT
---------- ----------
Apple               1
Apple               1
Apple               1
Apple               1
Oranges             3
Oranges             3
Oranges             3
Oranges             3
Peach               1
Peach               1
Peach               1
Peach               1
NULL                5
NULL                5
NULL                5
NULL                5
```

Как видим, в окончательных результатах группы повторяются. К счастью, не стоит сильно переживать, поскольку это можно назвать парадоксом лишь частично. Первое свойство группы говорит, что для (G,e) G является результирующим множеством самостоятельного или самодостаточного запроса, использующего оператор GROUP BY. Попросту говоря, результирующее множество любого запроса с GROUP BY соответствует нашему определению группы. Дублирование групп может возникнуть лишь при создании мультимножества в результате сочетания результатов двух запросов с GROUP BY. В первом запросе предыдущего примера используется оператор UNION ALL, что является операцией не над множествами, а над мультимножествами, и дважды вызывается GROUP BY, таким образом, по сути, выполняется два запроса.

Если использовать оператор UNION, что является операцией над множествами, повторяющихся групп не появится.

Второй из представленных запросов использует декартово произведение, но для его выполнения сначала надо материализовать группу. Та-

ким образом, самодостаточный запрос с GROUP BY удовлетворяет нашему определению. Ни в одном из двух примеров нет никаких противоречий определению SQL-группы. Они приведены здесь для полноты картины и как доказательство того, что в SQL возможно практически все.

Отношения между SELECT и GROUP BY

Дав определение понятию группа и подтвердив его, можем перейти к более приземленным аспектам запросов, использующих оператор GROUP BY. При создании групп в SQL важно понимать взаимоотношения между операторами SELECT и GROUP BY. При использовании агрегатных функций, таких как COUNT, необходимо помнить, что любой элемент списка оператора SELECT, не используемый как аргумент агрегатной функции, должен быть частью группы. Например, если записать оператор SELECT следующим образом:

```
select deptno, count(*) as cnt
  from emp
```

DEPTNO должен быть обязательно указан в списке оператора GROUP BY:

```
select deptno, count(*) as cnt
  from emp
 group by deptno
```

```
DEPTNO    CNT
------ ------
    10      3
    20      5
    30      6
```

Исключениями из этого правила являются константы, скалярные значения, возвращенные пользовательскими функциями, оконными функциями и несвязанными скалярными подзапросами. Поскольку оператор SELECT обрабатывается после оператора GROUP BY, эти конструкции могут присутствовать в списке оператора SELECT и не требуют (а в некоторых случаях это невозможно) задания в GROUP BY. Например:

```
select 'hello' as msg,
       1 as num,
       deptno,
       (select count(*) from emp) as total,
       count(*) as cnt
  from emp
 group by deptno
```

```
MSG   NUM DEPTNO TOTAL CNT
----- --- ------ ----- ---
hello   1     10    14   3
hello   1     20    14   5
hello   1     30    14   6
```

Не дайте этому запросу сбить вас с толку. Элементы списка SELECT, не перечисленные в операторе GROUP BY, не оказывают влияния ни на значения CNT для каждого DEPTNO, ни на значения DEPTNO. На основании результатов предыдущего запроса можно более точно сформулировать правило о соответствии элементов списка SELECT и оператора GROUP BY при использовании агрегатов:

> Элементы списка оператора SELECT, которые потенциально могут изменить группу или значение, возвращаемое агрегатной функцией, должны быть включены в оператор GROUP BY.

В предыдущем запросе дополнительные элементы списка оператора SELECT не меняют ни значения CNT любой из групп (для каждого DEPTNO), ни сами группы.

Теперь пришло время задать вопрос: а какие именно элементы списка SELECT могут менять группировку или значение, возвращаемое агрегатной функцией? Ответ прост: другие столбцы запрашиваемых(ой) таблиц(ы). Добавим в рассматриваемый запрос столбец JOB:

```
select deptno, job, count(*) as cnt
  from emp
 group by deptno, job

DEPTNO JOB       CNT
------ --------- ---
    10 CLERK       1
    10 MANAGER     1
    10 PRESIDENT   1
    20 CLERK       2
    20 ANALYST     2
    20 MANAGER     1
    30 CLERK       1
    30 MANAGER     1
    30 SALESMAN    4
```

Указывая еще один столбец таблицы EMP, JOB, мы меняем группу и результирующее множество. Таким образом, теперь JOB необходимо включить в оператор GROUP BY вместе с DEPTNO, в противном случае запрос даст сбой. Включение JOB в операторы SELECT/GROUP BY меняет запрос и превращает его из «Сколько служащих в каждом отделе?» в «Сколько разных типов служащих в каждом отделе?». Еще раз обратите внимание на то, что группы уникальны. Значения DEPTNO и JOB *по отдельности* не уникальны, а вот их сочетания (которые мы видим в списке GROUP BY и SELECT и, таким образом, которые являются группами) уникальны (например, пара 10 и CLERK появляется только один раз).

Если в списке оператора SELECT присутствуют только агрегатные функции, в операторе GROUP BY могут быть перечислены любые действительные столбцы. Рассмотрим следующие два запроса, подтверждающие этот факт:

```
select count(*)
  from emp
 group by deptno

  COUNT(*)
----------
         3
         5
         6

select count(*)
  from emp
 group by deptno,job

  COUNT(*)
----------
         1
         1
         1
         2
         2
         1
         1
         1
         4
```

Кроме агрегатных функций в списке оператора SELECT могут быть и другие элементы. Включать их в список не обязательно, но часто это делает результаты запроса более понятными и удобными для использования.

Как правило, при использовании GROUP BY и агрегатных функций любые элементы списка оператора SELECT (из таблиц(ы), указанных(ой) в операторе FROM), не используемые как аргумент агрегатной функции, должны быть включены в оператор GROUP BY. Однако в MySQL есть «возможность» отойти от этого правила и размещать в списке SELECT элементы (т. е. столбцы таблиц(ы), из которых(ой) осуществляется выборка), не используемые как аргументы агрегатной функции и не представленные в операторе GROUP BY. Я очень неточно употребляю здесь термин «возможность», поскольку это «бомба замедленного действия», ошибка, ждущая своего часа, и я советую избегать ее применения. Собственно говоря, если вы работаете с MySQL и крайне беспокоитесь о правильности своих запросов, не используйте эту, гм, «возможность».

Сегментирование

Если ясна концепция группировки и использования агрегатов в SQL, разобраться с *оконными функциями* просто. Оконные функции, как и агрегатные функции, выполняют агрегацию заданного набора (группы) строк, но вместо того чтобы возвращать по одному значению на

группу, оконные функции могут возвращать несколько значений для каждой группы. Группа строк, подвергающаяся агрегации, – это *окно* (отсюда название «оконные функции»). В DB2 такие функции называют *функциями оперативного анализа данных* (*online analytic processing, OLAP*), а в Oracle – *аналитическими функциями*, но стандарт ISO SQL называет их оконными функциями, поэтому в книге я использую именно этот термин.

Простой пример

Скажем, требуется подсчитать общее число служащих во всех отделах. Традиционно для этого используется запрос с функцией COUNT(*) ко всей таблице EMP:

```
select count(*) as cnt
  from emp

   CNT
------
    14
```

Это довольно просто, но часто возникает необходимость получить такие агрегатные данные из строк, которые не входят в данную группировку. Оконные функции упрощают решение подобных задач. Например, следующий запрос показывает, как с помощью оконной функции можно получить агрегатные данные (общее число служащих) из заданных строк (по одной на каждого служащего):

```
select ename,
       deptno,
       count(*) over() as cnt
  from emp
 order by 2

ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10     14
KING           10     14
MILLER         10     14
SMITH          20     14
ADAMS          20     14
FORD           20     14
SCOTT          20     14
JONES          20     14
ALLEN          30     14
BLAKE          30     14
MARTIN         30     14
JAMES          30     14
TURNER         30     14
WARD           30     14
```

В этом примере COUNT(*) OVER() – это вызов оконной функции. Присутствие ключевого слова OVER показывает, что COUNT будет рассматриваться не как агрегатная, а как оконная функция. В общем, стандарт SQL допускает использование всех агрегатных функций в качестве оконных, а ключевое слово OVER является отличительным признаком оконных функций.

Итак, что именно делает оконная функция COUNT(*) OVER ()? Для каждой возвращаемой запросом строки она возвращает количество *всех строк* в таблице. Как предполагают пустые круглые скобки, ключевое слово OVER принимает дополнительные операторы, которые определяют диапазон строк, рассматриваемый оконной функцией. Если таких операторов нет, оконная функция обрабатывает все строки результирующего множества, поэтому в данном примере значение 14 повторяется в каждой строке вывода.

Надеюсь, вы начинаете видеть громадный потенциал оконных функций, состоящий в обеспечении возможности работать со многими уровнями агрегации в одну строку. Далее в этом приложении будет показано, насколько невероятно полезной может быть эта возможность.

Порядок выполнения

Прежде чем углубиться в оператор OVER, важно отметить, что оконные функции выполняются как последний шаг в обработке SQL перед оператором ORDER BY. В качестве примера порядка выполнения оконных функций возьмем запрос из предыдущего раздела и применим предикат WHERE, чтобы отфильтровать служащих 20 и 30-го отделов (DEPTNO 20 и 30):

```
select ename,
       deptno,
       count(*) over() as cnt
  from emp
 where deptno = 10
 order by 2

ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10      3
KING           10      3
MILLER         10      3
```

Значение поля CNT во всех строках теперь не 14, а 3. В этом примере тремя строками результирующее множество ограничивает предикат WHERE, поэтому оконная функция обрабатывает только три строки (на момент обработки части SELECT запроса оконная функция имеет доступ всего к трем строкам). На этом примере мы видим, что оконные функции выполняются после обработки таких операторов, как WHERE и GROUP BY.

Сегменты

Для определения *сегмента* или группы строк, подвергающихся агрегации, используется оператор PARTITION BY. Как мы видели ранее, при использовании пустых круглых скобок сегментом, агрегат которого будет вычислять оконная функция, является все результирующее множество. Оператор PARTITION BY можно рассматривать как «скользящий GROUP BY», потому что в отличие от обычного GROUP BY группы, создаваемые PARTITION BY, в результирующем множестве не являются уникальными. PARTITION BY может использоваться для вычисления агрегата заданной группы строк (отсчет начинается заново для каждой новой группы), и тогда будут представлены все экземпляры этого значения в таблице (все члены каждой группы), а не одна группа. Рассмотрим следующий запрос:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as cnt
  from emp
 order by 2
```

```
ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10      3
KING           10      3
MILLER         10      3
SMITH          20      5
ADAMS          20      5
FORD           20      5
SCOTT          20      5
JONES          20      5
ALLEN          30      6
BLAKE          30      6
MARTIN         30      6
JAMES          30      6
TURNER         30      6
WARD           30      6
```

Этот запрос по-прежнему возвращает 14 строк, но теперь в результате применения PARTITION BY DEPTNO функция COUNT выполняется для каждого отдела. Значение поля CNT для всех служащих одного отдела (одного сегмента) будет одинаковым, потому что агрегация выполняется по отделам (отсчет начинается заново для нового отдела). Заметьте также, что, кроме членов каждой группы, мы получаем информацию о каждой группе. Предыдущий запрос можно рассматривать как более рациональную версию следующего запроса:

```
select e.ename,
       e.deptno,
       (select count(*) from emp d
         where e.deptno=d.deptno) as cnt
```

```
  from emp e
 order by 2
```

```
ENAME      DEPTNO    CNT
---------- ------ ------
CLARK          10      3
KING           10      3
MILLER         10      3
SMITH          20      5
ADAMS          20      5
FORD           20      5
SCOTT          20      5
JONES          20      5
ALLEN          30      6
BLAKE          30      6
MARTIN         30      6
JAMES          30      6
TURNER         30      6
WARD           30      6
```

Оператор PARTITION BY замечателен также тем, что выполняет вычисления независимо от других оконных функций, осуществляя сегментирование по другим столбцам в том же выражении SELECT. Рассмотрим следующий запрос, в результате которого для каждого служащего возвращается такая информация: его имя, отдел, количество служащих в этом отделе, его должность и количество служащих, занимающих эту должность:

```
select ename,
       deptno,
       count(*) over(partition by deptno) as dept_cnt,
       job,
       count(*) over(partition by job)    as job_cnt
  from emp
 order by 2
```

```
ENAME      DEPTNO DEPT_CNT JOB       JOB_CNT
---------- ------ -------- --------- -------
MILLER         10        3 CLERK           4
CLARK          10        3 MANAGER         3
KING           10        3 PRESIDENT       1
SCOTT          20        5 ANALYST         2
FORD           20        5 ANALYST         2
SMITH          20        5 CLERK           4
JONES          20        5 MANAGER         3
ADAMS          20        5 CLERK           4
JAMES          30        6 CLERK           4
MARTIN         30        6 SALESMAN        4
TURNER         30        6 SALESMAN        4
WARD           30        6 SALESMAN        4
ALLEN          30        6 SALESMAN        4
BLAKE          30        6 MANAGER         3
```

В результирующем множестве можно увидеть, что служащие одного отдела имеют одно значение поля DEPT_CNT и что значение поля JOB_CNT одинаковое для служащих, занимающих одну и ту же должность.

На данный момент должно быть ясно, что оператор PARTITION BY работает так же, как и оператор GROUP BY, но на него не оказывают влияния другие элементы оператора SELECT, и он не требует присутствия оператора GROUP BY.

Что происходит в случае присутствия NULL-значений

Как и оператор GROUP BY, PARTITION BY сводит все NULL-значения в одну группу или сегмент. Таким образом, результат присутствия NULL-значений при использовании PARTITION BY аналогичен тому, который имеем при работе с GROUP BY. В следующем запросе оконная функция применяется для подсчета количества служащих, получающих ту или иную сумму комиссионных (для удобства чтения вместо NULL возвращается -1):

```
select coalesce(comm,-1) as comm,
       count(*)over(partition by comm) as cnt
  from emp
```

```
  COMM        CNT
------ ----------
     0          1
   300          1
   500          1
  1400          1
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
    -1         10
```

Поскольку используется COUNT(*), подсчитывается количество строк. Можно заметить, что 10 служащих не получают комиссионные (NULL). Примените COMM вместо * и получите совершенно другие результаты:

```
select coalesce(comm,-1) as comm,
       count(comm)over(partition by comm) as cnt
  from emp
```

```
COMM        CNT
---- ----------
   0          1
```

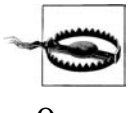
```
 300          1
 500          1
1400          1
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
  -1          0
```

В этом запросе используется COUNT(COMM). Это означает, что пересчитываются только не-NULL значения столбца COMM. Выявлен один служащий с размером комиссионного вознаграждения 0, один служащий с размером комиссионного вознаграждения 300 и т. д. Но обратите внимание на значение счетчика для служащих с комиссионными NULL! Оно равно 0. Почему? Потому что агрегатные функции игнорируют NULL-значения или, если выразиться более точно, агрегатные функции учитывают только не-NULL значения.

При работе с функцией COUNT необходимо продумать, как должны обрабатываться NULL-значения. Чтобы не учитывать NULL-значения, используется COUNT(column). Если NULL-значения требуется включить в рассмотрение, применяется COUNT(*) (при этом подсчитываются уже не значения столбца, а строки).

Когда порядок имеет значение

Иногда порядок рассмотрения строк оконной функцией оказывает влияние на результаты запроса. Поэтому синтаксис оконной функции включает подоператор ORDER BY, который можно разместить в операторе OVER. Оператор ORDER BY определяет порядок расположения строк в сегменте (помните, «сегмент» в отсутствие оператора PARTITION BY – это все результирующее множество).



Некоторые оконные функции *требуют* упорядочения обрабатываемых сегментов строк. Таким образом, для таких оконных функций оператор ORDER BY является обязательным.

Оператор ORDER BY, используемый в операторе OVER оконной функции, определяет две вещи:

1. Как упорядочены строки в сегменте.
2. Какие строки участвуют в вычислениях.

Рассмотрим следующий запрос, который суммирует и вычисляет текущую сумму заработных плат служащих 10-го отдела (DEPTNO 10):

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate) as running_total
  from emp
 where deptno=10
```

```
DEPTNO ENAME  HIREDATE      SAL TOTAL1 TOTAL2 RUNNING_TOTAL
------ ------ ----------- ----- ------ ------ -------------
    10 CLARK  09-JUN-1981  2450   8750   8750          2450
    10 KING   17-NOV-1981  5000   8750   8750          7450
    10 MILLER 23-JAN-1982  1300   8750   8750          8750
```

Чтобы держать вас в тонусе, я включил суммирование с пустыми круглыми скобками. Обратите внимание, что значения TOTAL1 и TOTAL2 одинаковые. Почему? Опять же из-за порядка обработки оконных функций. Предикат WHERE фильтрует результирующее множество таким образом, что при суммировании учитываются только заработные платы 10-го отдела (DEPTNO 10). В данном случае имеется всего один сегмент – все результирующее множество, состоящее из заработных плат только 10-го отдела. Следовательно, TOTAL1 и TOTAL2 равны.

Взглянув на значения столбца SAL, можно сразу понять, как получаются значения RUNNING_TOTAL (промежуточные результаты найти несложно, просуммировав значения SAL). Но более важно, почему включение ORDER BY в конструкцию OVER обеспечило вычисление текущей суммы? Дело в том, что при использовании ORDER BY в конструкции OVER в сегменте задается «скользящее» или «подвижное» окно по умолчанию, даже несмотря на то что мы его не видим. Последним элементом строки является сумма по HIREDATE, заданная выражением ORDER BY HIREDATE.

Следующий запрос аналогичен предыдущему, но в нем поведение по умолчанию, являющееся результатом применения ORDER BY HIREDATE, явно задается оператором RANGE BETWEEN (о котором более подробно рассказывается далее):

```
select deptno,
       ename,
       hiredate,
       sal,
       sum(sal)over(partition by deptno) as total1,
       sum(sal)over() as total2,
       sum(sal)over(order by hiredate
                    range between unbounded preceding
                      and current row) as running_total
  from emp
 where deptno=10
```

```
DEPTNO ENAME  HIREDATE      SAL TOTAL1 TOTAL2 RUNNING_TOTAL
------ ------ ----------- ----- ------ ------ -------------
    10 CLARK  09-JUN-1981  2450   8750   8750          2450
    10 KING   17-NOV-1981  5000   8750   8750          7450
    10 MILLER 23-JAN-1982  1300   8750   8750          8750
```

Оператор RANGE BETWEEN, представленный в этом запросе, ANSI называет *оператором кадрирования*. Далее я буду придерживаться этого термина. Теперь нетрудно увидеть, почему ORDER BY в конструкции OVER обусловливает вычисление текущей суммы; мы (по умолчанию) указали запросу суммировать все строки, начиная с текущей строки и включая все предыдущие («предыдущие» согласно ORDER BY; в данном случае строки упорядочиваются по HIREDATE).

Оператор кадрирования

Применим к результирующему множеству оператор кадрирования из предыдущего запроса, начиная со служащего CLARK, который был принят на работу первым:

1. Вычисляем сумму, начиная с заработной платы служащего CLARK, 2450, и включая заработные платы всех служащих, принятых на работу до CLARK. Поскольку CLARK был первым служащим 10-го отдела, сумма просто равна его заработной плате, 2450. Это первое значение, возвращенное RUNNING_TOTAL.
2. Перейдем к следующему согласно HIREDATE служащему, KING, и еще раз применим оператор кадрирования. Вычисляем сумму по столбцу SAL, начиная с текущей строки, 5000 (заработная плата KING), и включаем все предыдущие строки (всех служащих, которые были приняты на работу до KING). CLARK – единственный служащий, принятый на работу раньше KING, поэтому сумма равна 5000 + 2450, что составляет 7450 – второе значение, возвращенное RUNNING_TOTAL.
3. Переходим к MILLER, последнему служащему сегмента по HIREDATE, и еще раз применяем оператор кадрирования. Вычисляем сумму по столбцу SAL, начиная с текущей строки, 1300 (заработная плата MILLER), и включаем все предыдущие строки (всех служащих, которые были приняты на работу до MILLER). CLARK и KING поступили на работу раньше MILLER, таким образом, их заработные платы включаются в текущую сумму для MILLER: 2450+5000+1300= =8750 – значение, возвращаемое RUNNING_TOTAL для MILLER.

Как видите, именно оператор кадрирования формирует текущую сумму. ORDER BY определяет порядок вычисления и также подразумевает кадрирование по умолчанию.

В общем, оператор кадрирования позволяет определять разные «подокна» данных, участвующих в вычислениях. Существует множество способов определения таких подокон. Рассмотрим следующий запрос:

```
select deptno,
       ename,
       sal,
       sum(sal)over(order by hiredate
                    range between unbounded preceding
                      and current row) as run_total1,
       sum(sal)over(order by hiredate
                     rows between 1 preceding
                      and current row) as run_total2,
       sum(sal)over(order by hiredate
                    range between current row
                      and unbounded following) as run_total3,
       sum(sal)over(order by hiredate
                     rows between current row
                      and 1 following) as run_total4
  from emp
 where deptno=10

DEPTNO ENAME    SAL RUN_TOTAL1 RUN_TOTAL2 RUN_TOTAL3 RUN_TOTAL4
------ ------ ----- ---------- ---------- ---------- ----------
    10 CLARK   2450       2450       2450       8750       7450
    10 KING    5000       7450       7450       6300       6300
    10 MILLER  1300       8750       6300       1300       1300
```

Не пугайтесь, этот запрос не так ужасен, как выглядит. Мы уже видели RUN_TOTAL1 и результаты применения оператора кадрирования «UNBOUNDED PRECEDING AND CURRENT ROW». Вот краткое описание происходящего в других примерах:

RUN_TOTAL2

Вместо ключевого слова RANGE в данном операторе кадрирования используется ROWS; это означает, что *кадр*, или окно, будет создано из некоторого количества строк. 1 PRECEDING говорит о том, что кадр будет начинаться со строки, стоящей непосредственно перед текущей строкой. Диапазон распространяется до CURRENT ROW. Таким образом, RUN_TOTAL2 – это сумма заработных плат текущего и предыдущего, на основании HIREDATE, сотрудников.

Так случилось, что RUN_TOTAL1 и RUN_TOTAL2 для CLARK и KING равны. Почему? Подумайте, какие значения суммировались для каждого из этих служащих, в каждой из двух оконных функций. Подумайте хорошенько и найдете ответ.

RUN_TOTAL3

Оконная функция для вычисления RUN_TOTAL3 выполняет обратное тому, что делалось для RUN_TOTAL1. Суммирование начинается с текущей строки и включает не все предыдущие строки, а все последующие строки.

RUN_TOTAL4

Инверсия RUN_TOTAL2. Суммирование начинается с текущей строки и включает не одну предыдущую, а одну следующую строку.

Если вы поняли все, о чем шла речь до сих пор, у вас не возникнет проблем ни с одним рецептом данной книги. Однако если что-то остается неясным, попытайтесь попрактиковаться с собственными примерами и данными. Лично мне проще учиться через написание кода с использованием новых возможностей, а не просто читать о них.

Заключение к вопросу о кадрировании

Ниже приведен последний пример того, какой эффект на результат запроса оказывает применение оператора кадрирования:

```
select ename,
       sal,
       min(sal)over(order by sal) min1,
       max(sal)over(order by sal) max1,
       min(sal)over(order by sal
                    range between unbounded preceding
                      and unbounded following) min2,
       max(sal)over(order by sal
                    range between unbounded preceding
                      and unbounded following) max2,
       min(sal)over(order by sal
                    range between current row
                      and current row) min3,
       max(sal)over(order by sal
                    range between current row
                      and current row) max3,
       max(sal)over(order by sal
                     rows between 3 preceding
                      and 3 following) max4
  from emp
```

```
ENAME    SAL   MIN1   MAX1   MIN2   MAX2   MIN3   MAX3   MAX4
------ ----- ------ ------ ------ ------ ------ ------ ------
SMITH    800    800    800    800   5000    800    800   1250
JAMES    950    800    950    800   5000    950    950   1250
ADAMS   1100    800   1100    800   5000   1100   1100   1300
WARD    1250    800   1250    800   5000   1250   1250   1500
MARTIN  1250    800   1250    800   5000   1250   1250   1600
MILLER  1300    800   1300    800   5000   1300   1300   2450
TURNER  1500    800   1500    800   5000   1500   1500   2850
ALLEN   1600    800   1600    800   5000   1600   1600   2975
CLARK   2450    800   2450    800   5000   2450   2450   3000
BLAKE   2850    800   2850    800   5000   2850   2850   3000
JONES   2975    800   2975    800   5000   2975   2975   5000
SCOTT   3000    800   3000    800   5000   3000   3000   5000
```

```
FORD    3000    800    3000    800    5000    3000    3000    5000
KING    5000    800    5000    800    5000    5000    5000    5000
```

Хорошо, разложим этот запрос на составляющие:

MIN1

В оконной функции, формирующей этот столбец, не определен оператор кадрирования, поэтому в дело вступает кадрирование по умолчанию, UNBOUNDED PRECEDING AND CURRENT ROW. Почему значение MIN1 равно 800 для всех строк? Потому что первой идет наименьшая заработная плата (ORDER BY SAL), и она так всегда и остается наименьшей, или минимальной.

MAX1

Значения MAX1 сильно отличаются от значений MIN1. Почему? Ответ (опять же) – из-за оператора кадрирования по умолчанию, UNBOUNDED PRECEDING AND CURRENT ROW. В сочетании с ORDER BY SAL этот оператор кадрирования обеспечивает соответствие максимальной заработной платы значению заработной платы в текущей строке.

Рассмотрим первую строку, служащего SMITH. При обработке заработной платы SMITH и всех предыдущих заработных плат значение MAX1 для строки SMITH получается равным его заработной плате, потому что предыдущих заработных плат нет. Переходим к следующей строке, служащему JAMES. При сравнении заработной платы JAMES со всеми предыдущими заработными платами, в данном случае с заработной платой служащего SMITH, получаем, что заработная плата JAMES больше, чем у SMITH, и, таким образом, является максимальной из двух. Если применить эту логику ко всем строкам, мы увидим, что значение MAX1 для каждой строки соответствует заработной плате текущего служащего.

MIN2 и MAX2

В данном случае задан оператор кадрирования UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING, что аналогично заданию пустых круглых скобок. Таким образом, при вычислении MIN и MAX участвуют все строки результирующего множества. Как и следовало ожидать, значения MIN и MAX для всего результирующего множества являются постоянными, и, следовательно, значения этих столбцов также неизменны.

MIN3 и MAX3

В данном случае задан оператор кадрирования CURRENT ROW AND CURRENT ROW. Это означает, что при поиске MIN и MAX заработных плат используется только заработная плата текущего служащего. Таким образом, во всех строках и MIN3, и MAX3 равны SAL. Это просто, не так ли?

MAX4

Для вычисления MAX4 задан оператор кадрирования 3 PRECEDING AND 3 FOLLOWING, что означает, что для каждой строки рассматриваем три предыдущие и три последующие строки, а также саму текущую строку. В данном случае в результате вызова функции MAX(SAL) будет получено наибольшее значение заработной платы для этих строк.

Если взглянуть на значение MAX4 для служащего MARTIN, можно заметить, как действует оператор кадрирования. Заработная плата MARTIN – 1250. Заработные платы трех служащих до MARTIN: 1250 (WARD), 1100 (ADAMS) и 950 (JAMES). Заработные платы трех служащих после MARTIN: 1300 (MILLER), 1500 (TURNER) и 1600 (ALLEN). Из всех этих заработных плат, включая заработную плату MARTIN, наибольшей является заработная плата ALLEN. Таким образом, значение MAX4 для MARTIN равно 1600.

Понятность + производительность = мощь

Как видите, оконные функции обладают исключительной мощью, поскольку позволяют создавать запросы, содержащие как детальную, так и обобщенную информацию. Запросы, в которых применяются оконные функции, короче, но при этом эффективнее, чем запросы, использующие несколько рефлексивных объединений и/или скалярных подзапросов. Рассмотрим следующий запрос, который без труда отвечает на все поставленные вопросы: «Сколько служащих в каждом отделе? Сколько служащих, занимающих ту или иную должность, в каждом отделе (например, сколько клерков в 10-м отделе)? Сколько всего служащих в таблице EMP?»

```
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over () as total
  from emp
```

```
DEPTNO JOB          EMP_CNT    JOB_CNT      TOTAL
------ --------- ---------- ---------- ----------
    10 CLERK              3          1         14
    10 MANAGER            3          1         14
    10 PRESIDENT          3          1         14
    20 ANALYST            5          2         14
    20 ANALYST            5          2         14
    20 CLERK              5          2         14
    20 CLERK              5          2         14
    20 MANAGER            5          1         14
    30 CLERK              6          1         14
    30 MANAGER            6          1         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
```

```
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
```

Чтобы получить такое же результирующее множество без применения оконных функций, придется поработать немного больше:

```
select a.deptno, a.job,
       (select count(*) from emp b
         where b.deptno = a.deptno) as emp_cnt,
       (select count(*) from emp b
         where b.deptno = a.deptno and b.job = a.job) as job_cnt,
       (select count(*) from emp) as total
  from emp a
 order by 1,2
```

```
DEPTNO JOB          EMP_CNT    JOB_CNT      TOTAL
------ --------- ---------- ---------- ----------
    10 CLERK              3          1         14
    10 MANAGER            3          1         14
    10 PRESIDENT          3          1         14
    20 ANALYST            5          2         14
    20 ANALYST            5          2         14
    20 CLERK              5          2         14
    20 CLERK              5          2         14
    20 MANAGER            5          1         14
    30 CLERK              6          1         14
    30 MANAGER            6          1         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
    30 SALESMAN           6          4         14
```

Решение без использования оконных функций написать нетрудно, хотя, безусловно, оно не такое ясное или эффективное (разница в производительности для таблицы в 14 строк незаметна, но примените эти запросы к таблице, скажем, включающей 1000 или 10000 строк, и преимущество оконных функций над многократным рефлексивным объединением и скалярными подзапросами станет очевидным).

Формирование основы

Кроме повышения удобства чтения и производительности, оконные функции создают «основу» для более сложных запросов в «стиле отчета». Например, рассмотрим следующий запрос в «стиле отчета», во вложенном запросе которого используются оконные функции, а затем во внешнем запросе выполняется агрегация результатов. Оконные функции позволяют получать как детальные, так и обобщенные данные, что пригодится для отчетов. Показанный ниже запрос использует оконные функции для подсчета значений разных сегментов. Поскольку выполняется агрегация многих строк, вложенный запрос возвращает все строки таблицы EMP, которые с помощью внешних выражений CASE могут быть транспонированы для создания форматированного отчета:

```
select deptno,
       emp_cnt as dept_total,
       total,
       max(case when job = 'CLERK'
                then job_cnt else 0 end)  as clerks,
       max(case when job = 'MANAGER'
                then job_cnt else 0  end) as mgrs,
       max(case when job = 'PRESIDENT'
                then job_cnt else 0  end) as prez,
       max(case when job = 'ANALYST'
                then job_cnt else 0  end) as anals,
       max(case when job = 'SALESMAN'
                then job_cnt else 0  end) as smen
  from (
select deptno,
       job,
       count(*) over (partition by deptno) as emp_cnt,
       count(job) over (partition by deptno,job) as job_cnt,
       count(*) over () as total
  from emp
       ) x
 group by deptno, emp_cnt, total
```

```
DEPTNO DEPT_TOTAL TOTAL CLERKS MGRS PREZ ANALS SMEN
------ ---------- ----- ------ ---- ---- ----- ----
    10          3    14      1    1    1     0    0
    20          5    14      2    1    0     2    0
    30          6    14      1    1    0     0    4
```

Приведенный выше запрос возвращает все отделы, общее число служащих в каждом из отделов, общее число служащих в таблице EMP и распределение служащих по должностям для каждого отдела. Все это делает один запрос, без дополнительных объединений или временных таблиц!

В качестве заключительного примера того, как просто можно ответить на множество вопросов, используя оконные функции, рассмотрим следующий запрос:

```
select ename as name,
       sal,
       max(sal)over(partition by deptno) as hiDpt,
       min(sal)over(partition by deptno) as loDpt,
       max(sal)over(partition by job) as hiJob,
       min(sal)over(partition by job) as loJob,
       max(sal)over() as hi,
       min(sal)over() as lo,
       sum(sal)over(partition by deptno
                    order by sal,empno) as dptRT,
       sum(sal)over(partition by deptno) as dptSum,
       sum(sal)over() as ttl
  from emp
```

```
order by deptno,dptRT
```

```
NAME      SAL HIDPT LODPT HIJOB LOJOB    HI   LO  DPTRT DPTSUM    TTL
------ ----- ----- ----- ----- ----- ----- ---- ------ ------ ------
MILLER  1300  5000  1300  1300   800  5000  800   1300   8750  29025
CLARK   2450  5000  1300  2975  2450  5000  800   3750   8750  29025
KING    5000  5000  1300  5000  5000  5000  800   8750   8750  29025
SMITH    800  3000   800  1300   800  5000  800    800  10875  29025
ADAMS   1100  3000   800  1300   800  5000  800   1900  10875  29025
JONES   2975  3000   800  2975  2450  5000  800   4875  10875  29025
SCOTT   3000  3000   800  3000  3000  5000  800   7875  10875  29025
FORD    3000  3000   800  3000  3000  5000  800  10875  10875  29025
JAMES    950  2850   950  1300   800  5000  800    950   9400  29025
WARD    1250  2850   950  1600  1250  5000  800   2200   9400  29025
MARTIN  1250  2850   950  1600  1250  5000  800   3450   9400  29025
TURNER  1500  2850   950  1600  1250  5000  800   4950   9400  29025
ALLEN   1600  2850   950  1600  1250  5000  800   6550   9400  29025
BLAKE   2850  2850   950  2975  2450  5000  800   9400   9400  29025
```

Этот запрос отвечает на перечисленные ниже вопросы легко, эффективно и понятно (и без дополнительных объединений с EMP!). Чтобы определить:

1. Кто из всех служащих получает наибольшую заработную плату (HI).
2. Кто из всех служащих получает наименьшую заработную плату (LO).
3. Кто получает наибольшую заработную плату в своем отделе (HIDPT).
4. Кто получает наименьшую заработную плату в своем отделе (LODPT).
5. Кто получает наибольшую заработную плату на данной должности (HIJOB).
6. Кто получает наименьшую заработную плату на данной должности (LOJOB).
7. Сумму всех заработных плат (TTL).
8. Сумму заработных плат по отделам (DPTSUM).
9. Текущую сумму по всем заработным платам по отделам (DPTRT).

просто сравниваем каждого служащего и его заработную плату с другими строками результирующего множества.

B

Вспоминаем Розенштейна

Данное приложение – дань уважения Дэвиду Розенштейну. Как я говорил во введении, я считаю его книгу «The Essence of SQL» лучшей книгой (даже сегодня) по SQL всех времен и народов. В ней всего 119 страниц, но она охватывает, на мой взгляд, все ключевые для любого программиста на SQL вопросы. В частности, Дэвид показывает, как анализировать задачу и находить решение. Его решения ориентированы исключительно на множества. Даже если размер ваших таблиц не позволяет использовать эти решения, его методы изумительны, поскольку заставляют перестать искать процедурное решение задачи и начать думать категориями множеств.

Книга «The Essence of SQL» была опубликована задолго до появления оконных функций и операторов MODEL. В этом приложении для некоторых задач из книги Розенштейна я предлагаю альтернативные решения с использованием современных функций SQL. (Будут ли эти решения лучше, чем решения Розенштейна, зависит от обстоятельств.) В конце каждого обсуждения представлено решение, основанное на оригинальном решении из книги Дэвида. Если пример является разновидностью задачи Розенштейна, представленное для него решение тоже является вариантом решения (решение, которого может не быть в его книге, но в котором используется аналогичная техника).

Таблицы Розенштейна

Следующие таблицы основаны на таблицах из книги Розенштейна и будут использоваться в этой главе:

```
/* таблица студентов */
create table student
( sno    integer,
  sname  varchar(10),
  age    integer
```

```
)

/* таблица предметов */
create table courses
( cno      varchar(5),
  title    varchar(10),
  credits  integer
)

/* таблица преподавателей */
create table professor
( lname   varchar(10),
  dept    varchar(10),
  salary  integer,
  age     integer
)

/* таблица студентов и изучаемых ими предметов */
create table take
( sno    integer,
  cno    varchar(5)
)

/* таблица преподавателей и читаемых ими предметов */
create table teach
( lname   varchar(10),
  cno     varchar(5)
)

insert into student values (1,'AARON',20)
insert into student values (2,'CHUCK',21)
insert into student values (3,'DOUG',20)
insert into student values (4,'MAGGIE',19)
insert into student values (5,'STEVE',22)
insert into student values (6,'JING',18)
insert into student values (7,'BRIAN',21)
insert into student values (8,'KAY',20)
insert into student values (9,'GILLIAN',20)
insert into student values (10,'CHAD',21)

insert into courses values ('CS112','PHYSICS',4)
insert into courses values ('CS113','CALCULUS',4)
insert into courses values ('CS114','HISTORY',4)

insert into professor values ('CHOI','SCIENCE',400,45)
insert into professor values ('GUNN','HISTORY',300,60)
insert into professor values ('MAYER','MATH',400,55)
insert into professor values ('POMEL','SCIENCE',500,65)
insert into professor values ('FEUER','MATH',400,40)

insert into take values (1,'CS112')
insert into take values (1,'CS113')
insert into take values (1,'CS114')
insert into take values (2,'CS112')
insert into take values (3,'CS112')
```

```
insert into take values (3,'CS114')
insert into take values (4,'CS112')
insert into take values (4,'CS113')
insert into take values (5,'CS113')
insert into take values (6,'CS113')
insert into take values (6,'CS114')

insert into teach values ('CHOI','CS112')
insert into teach values ('CHOI','CS113')
insert into teach values ('CHOI','CS114')
insert into teach values ('POMEL','CS113')
insert into teach values ('MAYER','CS112')
insert into teach values ('MAYER','CS114')
```

Запросы с отрицанием

В своей книге Розенштейн подходит к обучению SQL через рассмотрение различных типов фундаментальных задач, которые часто приходится решать, в том или ином виде. Отрицание – один из таких типов. Часто требуется найти строки, для которых не выполняется некоторое условие. С простыми условиями все понятно, но, как показывают следующие запросы, для решения некоторых задач отрицания требуется применить творческий подход и здравый смысл.

Запрос 1

Требуется найти студентов, которые не изучают курс CS112. Следующий запрос возвращает неверные результаты:

```
select *
  from student
 where sno in ( select sno
                  from take
                 where cno != 'CS112' )
```

Поскольку студент может изучать несколько предметов, этот запрос может (он так и делает) возвратить студентов, изучающих CS112. Запрос неверен, потому что не отвечает на вопрос «Кто не изучает CS112?». Он дает ответ на вопрос «Кто изучает не CS112?». Требуется получить результирующее множество, включающее студентов, которые не изучают ни один предмет, а также тех, среди изучаемых предметов которых нет CS112. В итоге должно быть получено следующее результирующее множество:

```
       SNO SNAME             AGE
---------- ---------- ----------
         5 STEVE              22
         6 JING               18
         7 BRIAN              21
         8 KAY                20
         9 GILLIAN            20
        10 CHAD               21
```

MySQL и PostgreSQL

С помощью выражения CASE и агрегатной функции MAX установите флаг, если CS112 присутствует в записи студента:

```
1  select s.sno,s.sname,s.age
2    from student s left join take t
3      on (s.sno = t.sno)
4   group by s.sno,s.sname,s.age
5  having max(case when t.cno = 'CS112'
6                  then 1 else 0 end) = 0
```

DB2 и SQL Server

С помощью выражения CASE и агрегатной функции MAX OVER установите флаг, если CS112 присутствует в записи студента:

```
 1  select distinct sno,sname,age
 2    from (
 3  select s.sno,s.sname,s.age,
 4         max(case when t.cno = 'CS112'
 5                  then 1 else 0 end)
 7         over(partition by s.sno,s.sname,s.age) as takes_CS112
 9    from student s left join take t
10      on (s.sno = t.sno)
11         ) x
12   where takes_CS112 = 0
```

Oracle

Для Oracle 9*i* Database и более поздних версий может использоваться приведенное выше решение для DB2. В качестве альтернативы можно применять собственный синтаксис Oracle для внешнего объединения; для пользователей Oracle 8*i* Database и более ранних версий – это единственно возможное решение:

```
/* решение с применением group by */
1  select s.sno,s.sname,s.age
2    from student s, take t
3   where s.sno = t.sno (+)
4   group by s.sno,s.sname,s.age
5  having max(case when t.cno = 'CS112'
6                  then 1 else 0 end) = 0

/* решение с применением оконной функции */

 1  select distinct sno,sname,age
 2    from (
 3  select s.sno,s.sname,s.age,
 4         max(case when t.cno = 'CS112'
 5                  then 1 else 0 end)
 7         over(partition by s.sno,s.sname,s.age) as takes_CS112
 9    from student s, take t
10   where s.sno = t.sno (+)
```

```
11         ) x
12   where takes_CS112 = 0
```

Обсуждение

Несмотря на различия синтаксисов, методика везде одна. Идея в том, чтобы создать в результирующем множестве столбец логического типа, значения которого являются признаком того, изучает студент CS112 или нет. Если студент изучает CS112, в этом столбце возвращается значение 1; в противном случае возвращается 0. В следующем запросе выражение CASE перенесено в список оператора SELECT. Ниже показаны промежуточные результаты:

```
select s.sno,s.sname,s.age,
       case when t.cno = 'CS112'
            then 1
            else 0
       end as takes_CS112
  from student s left join take t
    on (s.sno=t.sno)
```

```
SNO SNAME             AGE TAKES_CS112
--- ---------- ---------- -----------
  1 AARON              20           1
  1 AARON              20           0
  1 AARON              20           0
  2 CHUCK              21           1
  3 DOUG               20           1
  3 DOUG               20           0
  4 MAGGIE             19           1
  4 MAGGIE             19           0
  5 STEVE              22           0
  6 JING               18           0
  6 JING               18           0
  8 KAY                20           0
 10 CHAD               21           0
  7 BRIAN              21           0
  9 GILLIAN            20           0
```

Внешнее объединение с таблицей TAKE гарантирует, что в результирующее множество войдут даже студенты, не изучающие ни одного предмета. Следующий шаг – использовать функцию MAX и выбрать для каждого студента наибольшее значение, возвращенное выражением CASE. Если студент слушает курс CS112, наибольшим значением будет 1, потому что для всех остальных курсов CASE возвращает значения 0. Заключительный шаг для решения с использованием оператора GROUP BY – с помощью оператора HAVING выбрать студентов, для которых выражение MAX/CASE возвращает значение 0. В решении с использованием оконной функции необходимо поместить запрос во вложенный запрос и дать ему псевдоним TAKES_CS112, потому что в предикате WHERE нельзя напрямую ссылаться на оконные функции.

Из-за принципа работы оконных функций также необходимо удалить дубликаты, возникающие в результате умножения значений.

Оригинальное решение

Оригинальное решение этой задачи очень разумно. Вот оно:

```
select *
  from student
 where sno not in (select sno
                     from take
                    where cno = 'CS112')
```

Это можно описать так: «В таблице TAKE найти студентов, которые изучают CS112, и затем выбрать из таблицы STUDENT всех остальных студентов». Данная техника основана на совете о том, как следует обрабатывать отрицание, который Розенштейн дает в конце своей книги:

> Помните, настоящее отрицание требует двух проходов: чтобы найти, «кто не», сначала надо найти, «кто да», и затем избавиться от них.

Запрос 2

Требуется найти студентов, которые изучают CS112 или CS114, но не оба предмета одновременно. Следующий запрос, казалось бы, должен обеспечивать решение, но возвращает неверное результирующее множество:

```
select *
  from student
 where sno in ( select sno
                  from take
                 where cno != 'CS112'
                   and cno != 'CS114' )
```

Из всех студентов, изучающих какие-либо предметы, только студенты DOUG и AARON слушают оба курса, CS112 и CS114. Эти двое должны быть исключены. Студент STEVE изучает CS113, но не CS112 или CS114, и тоже не должен войти в результат.

Поскольку студент может изучать несколько предметов, будем использовать здесь такой подход: возвратим для каждого студента строку с информацией о том, изучает ли он CS112, или CS114, или оба предмета. При таком подходе можно без труда оценить, изучает ли студент оба предмета, без многократных просмотров данных. Результирующее множество должно быть таким:

```
SNO SNAME            AGE
--- ---------- ----------
  2 CHUCK              21
  4 MAGGIE             19
  6 JING               18
```

MySQL и PostgreSQL

С помощью выражения CASE и агрегатной функции SUM найдите студентов, изучающих или CS112, или CS114, но не оба предмета одновременно:

```
1  select s.sno,s.sname,s.age
2    from student s, take t
3   where s.sno = t.sno
4   group by s.sno,s.sname,s.age
5  having sum(case when t.cno in ('CS112','CS114')
6                  then 1 else 0 end) = 1
```

DB2, Oracle и SQL Server

С помощью выражения CASE и агрегатной функции SUM OVER найдите студентов, изучающих или CS112, или CS114, но не оба предмета одновременно:

```
1  select distinct sno,sname,age
2    from (
3  select s.sno,s.sname,s.age,
4         sum(case when t.cno in ('CS112','CS114') then 1 else 0 end)
5         over (partition by s.sno,s.sname,s.age) as takes_either_or
6    from student s, take t
7   where s.sno = t.sno
8         ) x
9   where takes_either_or = 1
```

Обсуждение

Первый шаг в решении этой задачи – использовать внутреннее объединение таблицы STUDENT с таблицей TAKE и таким образом исключить из рассмотрения всех студентов, не изучающих ни одного предмета. Следующий шаг – с помощью выражения CASE обозначить, изучает ли студент рассматриваемые предметы. В следующем запросе выражения CASE перенесены в список оператора SELECT. Получаем такие промежуточные результаты:

```
select s.sno,s.sname,s.age,
       case when t.cno in ('CS112','CS114')
            then 1 else 0 end as takes_either_or
  from student s, take t
 where s.sno = t.sno
```

```
SNO SNAME      AGE TAKES_EITHER_OR
--- ---------- --- ---------------
  1 AARON       20               1
  1 AARON       20               0
  1 AARON       20               1
  2 CHUCK       21               1
  3 DOUG        20               1
  3 DOUG        20               1
```

```
  4 MAGGIE      19               1
  4 MAGGIE      19               0
  5 STEVE       22               0
  6 JING        18               0
  6 JING        18               1
```

Значение 1 в поле TAKES_EITHER_OR указывает на то, что студент изучает CS112 или CS114. Студент может изучать несколько предметов, поэтому следующий шаг – свести всю информацию о студенте в одну строку. Для этого используем GROUP BY с агрегатной функцией SUM. SUM просуммирует значения TAKES_EITHER_OR для каждого студента:

```
select s.sno,s.sname,s.age,
       sum(case when t.cno in ('CS112','CS114')
                then 1 else 0 end) as takes_either_or
  from student s, take t
 where s.sno = t.sno
 group by s.sno,s.sname,s.age

SNO SNAME      AGE TAKES_EITHER_OR
--- ---------- --- ---------------
  1 AARON       20               2
  2 CHUCK       21               1
  3 DOUG        20               2
  4 MAGGIE      19               1
  5 STEVE       22               0
  6 JING        18               1
```

Для студентов, не изучающих CS112 или CS114, значение поля TAKES_EITHER_OR равно 0. Для студентов, изучающих и CS112, и CS114, значение поля TAKES_EITHER_OR равно 2. Таким образом, нас интересуют только студенты со значением 1 в столбце TAKES_EITHER_OR. В окончательном решении для выбора таких студентов (для которых SUM по TAKES_EITHER_OR возвращает 1) используется оператор HAVING.

В решении с оконными функциями применяется такой же подход. Запрос придется поместить во вложенный запрос и затем ссылаться на столбец TAKES_EITHER_OR, поскольку в предикате WHERE нельзя обращаться к оконным функциям напрямую (при обработке SQL они рассматриваются последними, после них идет только оператор ORDER BY). Из-за принципа работы оконных функций также необходимо удалить дубликаты, возникающие в результате умножения значений.

Оригинальное решение

Следующий запрос является оригинальным решением (внесены лишь небольшие изменения). Запрос довольно хорошо продуман и использует тот же подход, что и оригинальное решение в Запросе 1. Поиск студентов, изучающих оба предмета, CS112 и CS114, осуществляется посредством рефлексивного объединения. Затем с помощью подзапроса

эти студенты отфильтровываются из множества студентов, изучающих один из этих предметов:

```
select *
  from student s, take t
 where s.sno = t.sno
   and t.cno in ( 'CS112', 'CS114' )
   and s.sno not in ( select a.sno
                        from take a, take b
                       where a.sno = b.sno
                         and a.cno = 'CS112'
                         and b.cno = 'CS114'
```

Запрос 3

Требуется найти студентов, изучающих только CS112 и никакие другие предметы, но следующий запрос возвращает неверные результаты:

```
select s.*
  from student s, take t
 where s.sno = t.sno
   and t.cno = 'CS112'
```

CHUCK – единственный студент, который изучает только CS112, и только он должен быть возвращен в результате запроса.

Эту задачу можно сформулировать по-другому: «Найти студентов, которые изучают только CS112». Приведенный выше запрос выбирает студентов, изучающих CS112, но также студентов, которые изучают и другие предметы. Запрос должен отвечать на вопрос: «Кто изучает только один предмет, и этим предметом является CS112?»

MySQL и PostgreSQL

Чтобы запрос гарантированно возвращал студентов, изучающих только один предмет, используйте агрегатную функцию COUNT:

```
 1  select s.*
 2    from student s,
 3         take t1,
 4         (
 5  select sno
 6    from take
 7   group by sno
 8  having count(*) = 1
 9         ) t2
10   where s.sno  = t1.sno
11     and t1.sno = t2.sno
12     and t1.cno = 'CS112'
```

DB2, Oracle и SQL Server

Чтобы гарантированно получить студентов, изучающих только один предмет, используйте оконную функцию COUNT OVER:

```
 1  select sno,sname,age
 2    from (
 3  select s.sno,s.sname,s.age,t.cno,
 4         count(t.cno) over (
 5           partition by s.sno,s.sname,s.age
 6         ) as cnt
 7    from student s, take t
 8   where s.sno = t.sno
 9         ) x
10   where cnt = 1
11     and cno = 'CS112'
```

Обсуждение

Ключ к решению – написать запрос, отвечающий на оба вопроса: «Кто из студентов изучает только один предмет?» и «Кто из студентов изучает CS112?». Сначала с помощью вложенного запроса T2 находим студентов, изучающих только один предмет. Следующий шаг – объединить вложенный запрос T2 с таблицей TAKE и выбрать только тех студентов, которые изучают CS112 (таким образом, получаем студентов, изучающих всего один предмет, и этот предмет – CS112). Запрос ниже обеспечивает это:

```
select t1.*
  from take t1,
       (
select sno
  from take
 group by sno
having count(*) = 1
       ) t2
 where t1.sno = t2.sno
   and t1.cno = 'CS112'
```

```
SNO CNO
--- -----
  2 CS112
```

Заключительный шаг – объединение с таблицей STUDENT и выбор студентов соответственно результатам объединения вложенного запроса T2 и таблицы TAKE. В решении с оконными функциями используется аналогичный подход, но все делается несколько иначе (более эффективно). Вложенный запрос X возвращает студентов, изучаемые ими предметы и количество изучаемых студентами предметов (внутреннее объединение таблиц TAKE и STUDENT гарантирует исключение студентов, не изучающих ни одного предмета). Результаты показаны ниже:

```
select s.sno,s.sname,s.age,t.cno,
       count(t.cno) over (
         partition by s.sno,s.sname,s.age
       ) as cnt
```

```
  from student s, take t
 where s.sno = t.sno
```

```
SNO SNAME              AGE CNO          CNT
--- ---------- ---------- ----- ----------
  1 AARON              20 CS112          3
  1 AARON              20 CS113          3
  1 AARON              20 CS114          3
  2 CHUCK              21 CS112          1
  3 DOUG               20 CS112          2
  3 DOUG               20 CS114          2
  4 MAGGIE             19 CS112          2
  4 MAGGIE             19 CS113          2
  5 STEVE              22 CS113          1
  6 JING               18 CS113          2
  6 JING               18 CS114          2
```

Когда имеются наименования предметов и их количество, остается только выбрать строки, в которых значение CNT равно 1 и CNO равно CS112.

Оригинальное решение

Оригинальное решение использует подзапрос и двойное отрицание:

```
select s.*
  from student s, take t
 where s.sno = t.sno
   and s.sno not in ( select sno
                      from take
                      where cno != 'CS112' )
```

Это исключительно разумное решение: в запросе нет ни проверки количества изучаемых студентом предметов, ни фильтра, гарантирующего, что выбранные студенты действительно изучают CS112! Но тогда, как оно работает? Подзапрос возвращает всех студентов, изучающих не CS112. Результаты показаны ниже:

```
select sno
  from take
 where cno != 'CS112'
```

```
SNO
----
   1
   1
   3
   4
   5
   6
   6
```

В результате внешнего запроса выбираются студенты, которые изучают предмет (любой) и которых нет среди студентов, возвращенных

подзапросом. Если на мгновение опустить часть NOT IN внешнего запроса, результаты будут следующими (выбираются все студенты, изучающие что-либо):

```
select s.*
  from student s, take t
 where s.sno = t.sno

SNO SNAME             AGE
--- ---------- ----------
  1 AARON              20
  1 AARON              20
  1 AARON              20
  2 CHUCK              21
  3 DOUG               20
  3 DOUG               20
  4 MAGGIE             19
  4 MAGGIE             19
  5 STEVE              22
  6 JING               18
  6 JING               18
```

Если сравним два результирующих множества, мы увидим, что введение NOT IN во внешний запрос эффективно обеспечивает вычитание из множества значений SNO, возвращаемых внешним запросом, множества значений SNO, возвращенных подзапросом, в результате чего будет возвращен студент, значение SNO которого равно 2. Итак, подзапрос находит всех студентов, изучающих не CS112. Внешний запрос возвращает студентов, которых нет среди изучающих не CS112 (здесь остаются студенты, которые изучают только CS112 или вообще не изучают ни одного предмета). В результате объединения таблиц STUDENT и TAKE студенты, не изучающие ни одного предмета, отсеиваются, и остаются только те, которые изучают CS112 и только этот предмет. Решение задачи на основе множеств в лучшем виде!

Запросы с условием «не более»

Вопросы с условием «не более» представляют другой тип задач, с которыми вы сталкиваетесь время от времени. Довольно просто найти строки, для которых условие выполняется. А если требуется ограничить количество таких строк? Этому посвящены следующие два запроса.

Запрос 4

Требуется найти студентов, изучающих не более двух предметов. Студенты, не изучающие ни одного предмета, не должны рассматриваться. Из всех студентов, изучающих какие-либо предметы, только AARON изучает их более двух и должен быть исключен из результирующего множества. В конечном счете, хотелось бы получить следующее результирующее множество:

```
SNO SNAME            AGE
--- ---------- ----------
  2 CHUCK              21
  3 DOUG               20
  4 MAGGIE             19
  5 STEVE              22
  6 JING               18
```

MySQL и PostgreSQL

Чтобы определить, кто из студентов изучает не более двух предметов, используйте агрегатную функцию COUNT:

```
1 select s.sno,s.sname,s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno,s.sname,s.age
5 having count(*) <= 2
```

DB2, Oracle и SQL Server

Чтобы определить, кто из студентов изучает не более двух предметов, используйте оконную функцию COUNT OVER:

```
 1  select distinct sno,sname,age
 2    from (
 3  select s.sno,s.sname,s.age,
 4         count(*) over (
 5           partition by s.sno,s.sname,s.age
 6         ) as cnt
 7    from student s, take t
 8   where s.sno = t.sno
 9         ) x
10   where cnt <= 2
```

Обсуждение

В обоих решениях просто выполняется подсчет экземпляров значений SNO в таблице TAKE. Внутреннее объединение с таблицей TAKE гарантирует, что студенты, не изучающие ни одного предмета, будут исключены из окончательного результирующего множества.

Оригинальное решение

В своей книге Розенштейн использовал решение с агрегацией, показанное здесь для MySQL и PostgreSQL, а также предложил альтернативное решение с множеством рефлексивных объединений, которое можно увидеть ниже:

```
select distinct s.*
  from student s, take t
 where s.sno = t.sno
   and s.sno not in ( select t1.sno
                        from take t1, take t2, take t3
```

```
where t1.sno = t2.sno
  and t2.sno = t3.sno
  and t1.cno < t2.cno
  and t2.cno < t3.cno )
```

Решение с множеством рефлексивных объединений любопытно, потому что решает задачу без использования агрегации. Чтобы понять принцип его работы, обратим внимание на предикат WHERE подзапроса. Внутренние объединения по SNO гарантируют, что во всех столбцах каждой отдельно взятой строки, которая потенциально может быть возвращена подзапросом, содержится информация одного студента. Сравнения «меньше-чем» выясняют, изучает ли студент более двух предметов. Предикат WHERE подзапроса можно сформулировать так: «Для конкретного студента возвращаем строки, в которых первое значение CNO меньше второго значения CNO, и второе CNO меньше ТРЕТЬЕГО CNO». Если студент изучает меньше трех предметов, это выражение никогда не будет истинным, поскольку третьего CNO в этом случае нет. Задача подзапроса – найти студентов, изучающих три или более предмета. Тогда внешний запрос возвращает студентов, которые изучают, по крайней мере, один предмет и которых нет среди тех, кто был возвращен подзапросом.

Запрос 5

Требуется найти студентов, которые старше не более двух студентов. Иначе говоря, ищем только тех студентов, которые старше, чем нуль, один или два других студента. Окончательное результирующее множество должно быть таким:

```
SNO SNAME      AGE
---- ---------- ---
   6 JING        18
   4 MAGGIE      19
   1 AARON       20
   9 GILLIAN     20
   8 KAY         20
   3 DOUG        20
```

MySQL и PostgreSQL

Чтобы найти студентов, которые старше, чем нуль, один или два других студента, используйте агрегатную функцию COUNT и связанный подзапрос:

```
1  select s1.*
2    from student s1
3   where 2 >= ( select count(*)
4                  from student s2
5                 where s2.age < s1.age )
```

DB2, Oracle и SQL Server

Чтобы найти студентов, которые старше, чем нуль, один или два других студента, используйте ранжирующую функцию DENSE_RANK:

```
1  select sno,sname,age
2    from (
3  select sno,sname,age,
4         dense_rank()over(order by age) as dr
5    from student
6         ) x
7   where dr <= 3
```

Обсуждение

В решении с агрегатной функцией скалярный подзапрос используется, чтобы найти всех студентов, которые старше не более двух других студентов. Чтобы увидеть, как это происходит, перепишем решение с использованием скалярного подзапроса. В следующем примере столбец CNT представляет количество студентов, которые младше рассматриваемого студента:

```
select s1.*,
       (select count(*) from student s2
         where s2.age < s1.age) as cnt
  from student s1
  order by 4
```

```
SNO SNAME             AGE        CNT
--- ---------- ---------- ----------
  6 JING               18          0
  4 MAGGIE             19          1
  1 AARON              20          2
  3 DOUG               20          2
  8 KAY                20          2
  9 GILLIAN            20          2
  2 CHUCK              21          6
  7 BRIAN              21          6
 10 CHAD               21          6
  5 STEVE              22          9
```

Переписав решение таким образом, легко увидеть, что в окончательное результирующее множество вошли студенты, для которых значение поля CNT меньше или равно 2.

Решение с использованием ранжирующей функции DENSE_RANK подобно примеру со скалярным подзапросом в том, что все строки ранжируются на основании количества студентов, которые моложе текущего (допускаются одинаковые значения рангов и пропусков нет). Следующий запрос демонстрирует результат, возвращаемый функцией DENSE_RANK:

```
select sno,sname,age,
       dense_rank()over(order by age) as dr
```

```
  from student

SNO SNAME             AGE         DR
--- ---------- ---------- ----------
  6 JING               18          1
  4 MAGGIE             19          2
  1 AARON              20          3
  3 DOUG               20          3
  8 KAY                20          3
  9 GILLIAN            20          3
  2 CHUCK              21          4
  7 BRIAN              21          4
 10 CHAD               21          4
  5 STEVE              22          5
```

Заключительный шаг – поместить запрос во вложенный запрос и выбрать только те строки, где значение DR меньше или равно 3.

Оригинальное решение

Розенштейн применил интересный подход к решению данной задачи, перефразировав ее. Вместо «поиска студентов, которые старше не более двух других студентов» его метод заключается в «поиске студентов, которые не старше трех или более (по крайней мере, трех) студентов». Этот подход бесценен для тех, кто хочет решить задачу, оперируя множествами, потому что вынуждает искать решение в два этапа:

1. Найти множество студентов, которые старше трех или более студентов.
2. Просто возвратить всех студентов, которых нет среди выбранных на этапе 1.

Решение показано ниже:

```
select *
  from student
 where sno not in (
select s1.sno
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
   and s2.age > s3.age
   and s3.age > s4.age
)

SNO SNAME      AGE
--- ---------- ---
  6 JING        18
  4 MAGGIE      19
  1 AARON       20
  9 GILLIAN     20
```

```
  8 KAY          20
  3 DOUG         20
```

Если просмотрим решение снизу вверх, мы увидим, что шаг 1, «поиск всех студентов, которые старше трех или более студентов», выполняется первым. Он показан ниже (DISTINCT используется, чтобы уменьшить размер результирующего множества для удобства чтения):

```
select distinct s1.*
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
   and s2.age > s3.age
   and s3.age > s4.age

SNO SNAME      AGE
--- ---------- ---
  2 CHUCK       21
  5 STEVE       22
  7 BRIAN       21
 10 CHAD        21
```

Если все эти рефлексивные объединения сбивают с толку, просто сосредоточьтесь на предикате WHERE. S1.AGE больше, чем S2.AGE, таким образом, мы знаем, что здесь рассматриваются студенты, которые старше, по крайней мере, одного студента. Далее, S2.AGE больше, чем S3.AGE. Здесь рассматриваются студенты, которые старше двух других студентов. Если на данном этапе возникают какие-то трудности, вспомните, что сравнения «больше-чем» транзитивны, т. е. если S1.AGE больше S2.AGE и S2.AGE больше S3.AGE, тогда S1.AGE больше S3.AGE. Чтобы лучше понять, что происходит, разложим запрос до рефлексивного объединения и затем будем постепенно достраивать его по мере понимания. Например, найдем всех студентов, которые старше, по крайней мере, одного студента (должны быть возвращены все студенты, кроме самого младшего, JING):

```
select distinct s1.*
  from student s1,
       student s2
 where s1.age > s2.age

SNO SNAME      AGE
--- ---------- ---
  5 STEVE       22
  7 BRIAN       21
 10 CHAD        21
  2 CHUCK       21
  1 AARON       20
  3 DOUG        20
  9 GILLIAN     20
```

```
  8 KAY         20
  4 MAGGIE      19
```

Далее находим всех студентов, которые старше двух или более студентов (теперь и JING, и MAGGIE должны быть исключены из результирующего множества):

```
select distinct s1.*
  from student s1,
       student s2,
       student s3
 where s1.age > s2.age
   and s2.age > s3.age
```

```
SNO SNAME      AGE
--- ---------- ---
  1 AARON       20
  2 CHUCK       21
  3 DOUG        20
  5 STEVE       22
  7 BRIAN       21
  8 KAY         20
  9 GILLIAN     20
 10 CHAD        21
```

Наконец, находим всех студентов, которые старше трех или более студентов (в результирующем множестве только CHUCK, STEVE, BRIAN и CHAD):

```
select distinct s1.*
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
   and s2.age > s3.age
   and s3.age > s4.age
```

```
SNO SNAME      AGE
--- ---------- ---
  2 CHUCK       21
  5 STEVE       22
  7 BRIAN       21
 10 CHAD        21
```

Теперь, когда известно, кто из студентов старше трех или более студентов, просто возвращаем тех студентов, которые не вошли в число этих четырех, используя оператор NOT IN с подзапросом.

Запросы с условием «не менее»

Оборотная сторона условия «не более» – условие «не менее». Часто задачи с «не менее» можно решать, применяя техники, описанные для

условия «не более». При решении задач с условием «не менее» обычно полезно перефразировать их как «имеющий не менее, чем». Как правило, если из поставленного требования можно выделить пороговую величину, задача наполовину решена. А уже зная пороговое значение, можно делать конкретный выбор метода решения: в один этап (агрегация или оконные функции, обычно используется функция COUNT) или в два этапа (отрицание с подзапросом).

Запрос 6

Требуется найти студентов, которые изучают, по крайней мере, два предмета.

Может быть полезным перефразировать задачу и сформулировать ее так: «Найти студентов, которые изучают два или более предметов» или «найти студентов, которые изучают не менее двух предметов». Здесь подойдет техника, описанная в «Запросе 4»: используем агрегатную функцию COUNT или оконную функцию COUNT OVER. Должно быть получено следующее результирующее множество:

```
SNO SNAME             AGE
--- ---------- ----------
  1 AARON              20
  3 DOUG               20
  4 MAGGIE             19
  6 JING               18
```

MySQL и PostgreSQL

Чтобы найти студентов, изучающих, по крайней мере, два предмета, используйте агрегатную функцию COUNT:

```
1 select s.sno,s.sname,s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno,s.sname,s.age
5 having count(*) >= 2
```

DB2, Oracle и SQL Server

Чтобы найти студентов, изучающих, по крайней мере, два предмета, используйте оконную функцию COUNT OVER:

```
 1  select distinct sno,sname,age
 2    from (
 3  select s.sno,s.sname,s.age,
 4         count(*) over (
 5           partition by s.sno,s.sname,s.age
 6         ) as cnt
 7    from student s, take t
 8   where s.sno = t.sno
 9         ) x
10   where cnt >= 2
```

Обсуждение

Обсуждение решений, представленных в данном разделе, можно найти в «Запросе 4»; методика аналогична. В решении с использованием агрегации таблица STUDENT объединяется с таблицей TAKE, и с помощью функции COUNT в конструкции HAVING выбираются только те студенты, которые изучают два или более предметов. В решении с использованием оконной функции таблица STUDENT объединяется с таблицей TAKE, и выполняется пересчет по сегменту, заданному всеми столбцами таблицы STUDENT. После этого просто выбираются только те строки, в которых значение CNT равно или больше двух.

Оригинальное решение

В представленном ниже решении поиск студентов, изучающих два и более предмета, осуществляется посредством рефлексивного объединения таблицы TAKE. Эквиобъединение по столбцу SNO в подзапросе гарантирует, что для каждого студента рассматриваются только изучаемые им/ею предметы. Условие «больше-чем» для значений CNO может выполниться, только если студент изучает более одного предмета, в противном случае CNO равнялось бы CNO (поскольку предмет всего один). Последний шаг – возвратить всех студентов, которые были возвращены подзапросом, что и показано ниже:

```
select *
  from student
 where sno in (
select t1.sno
 from take t1,
      take t2
 where t1.sno = t2.sno
   and t1.cno > t2.cno
)
```

```
SNO SNAME             AGE
--- ---------- ----------
  1 AARON              20
  3 DOUG               20
  4 MAGGIE             19
  6 JING               18
```

Запрос 7

Требуется найти студентов, которые изучают и CS112, и CS114. Кроме этих двух предметов, CS112 и CS114, студенты могут изучать и другие предметы.

Эта задача аналогична решаемой в «Запросе 2», за исключением того, что в том случае студент мог изучать более двух предметов, тогда как теперь предметов должно быть *не менее* 2 (только студенты AARON и DOUG изучают оба предмета, CS112 и CS114). Решение из «Запроса 2»

можно без труда скорректировать для данной задачи. Окончательное результирующее множество должно быть таким:

```
SNO SNAME       AGE
--- ---------- ----
  1 AARON        20
  3 DOUG         20
```

MySQL и PostgreSQL

Чтобы найти студентов, изучающих оба предмета, CS112 и CS114, используйте агрегатные функции MIN и MAX:

```
1  select s.sno, s.sname, s.age
2    from student s, take t
3   where s.sno = t.sno
4     and t.cno in ('CS114','CS112')
5   group by s.sno, s.sname, s.age
6  having min(t.cno) != max(t.cno)
```

DB2, Oracle и SQL Server

Чтобы найти студентов, изучающих оба предмета, CS112 и CS114, используйте оконные функции MIN OVER и MAX OVER:

```
1  select distinct sno, sname, age
 2    from (
 3  select s.sno, s.sname, s.age,
 4         min(cno) over (partition by s.sno) as min_cno,
 5         max(cno) over (partition by s.sno) as max_cno
 6    from student s, take t
 7   where s.sno = t.sno
 8     and t.cno in ('CS114','CS112')
 9         ) x
10   where min_cno != max_cno
```

Обсуждение

В обоих решениях используется одна и та же техника. Оператор IN обеспечивает выбор только тех студентов, которые изучают CS112, или CS114, или оба предмета. Если студент не изучает оба предмета, MIN(CNO) будет равняться MAX(CNO), и данный студент будет исключен из рассмотрения. Чтобы наглядно показать, как это работает, ниже приведены промежуточные результаты решения с использованием оконной функции (T.CNO добавлен для ясности):

```
select s.sno, s.sname, s.age, t.cno,
       min(cno) over (partition by s.sno) as min_cno,
       max(cno) over (partition by s.sno) as max_cno
  from student s, take t
 where s.sno = t.sno
   and t.cno in ('CS114','CS112')

SNO SNAME       AGE CNO   MIN_C MAX_C
```

```
--- ---------- ---- ----- ----- -----
  1 AARON        20 CS114 CS112 CS114
  1 AARON        20 CS112 CS112 CS114
  2 CHUCK        21 CS112 CS112 CS112
  3 DOUG         20 CS114 CS112 CS114
  3 DOUG         20 CS112 CS112 CS114
  4 MAGGIE       19 CS112 CS112 CS112
  6 JING         18 CS114 CS114 CS114
```

Проанализировав результаты, просто заметить, что только для строк студентов AARON и DOUG выполняется условие MIN(CNO) != MAX(CNO).

Оригинальное решение

В оригинальном решении Розенштейна используется рефлексивное объединение таблицы TAKE. Ниже приведено это решение, которое прекрасно работает при условии правильной расстановки индексов:

```
select s.*
  from student s,
       take t1,
       take t2
 where s.sno  = t1.sno
   and t1.sno = t2.sno
   and t1.cno = 'CS112'
   and t2.cno = 'CS114'
```

```
SNO SNAME       AGE
--- ---------- ----
  1 AARON        20
  3 DOUG         20
```

Все решения основываются на том, что студент может изучать любое количество любых предметов, но два предмета, CS112 и CS114, являются обязательными. Если возникают вопросы относительно рефлексивного объединения, возможно, следующий пример будет более понятным:

```
select s.*
  from take t1, student s
 where s.sno   = t1.sno
   and t1.cno  = 'CS114'
   and 'CS112' = any (select t2.cno
                        from take t2
                       where t1.sno  = t2.sno
                         and t2.cno != 'CS114')
```

```
SNO SNAME       AGE
--- ---------- ----
  1 AARON        20
  3 DOUG         20
```

Запрос 8

Требуется найти студентов, которые старше, по крайней мере, двух других студентов.

Возможно, удобнее будет перефразировать задачу так: «Найти студентов, которые старше двух или более студентов». Здесь подойдет техника, аналогичная применяемой в «Запросе 5». Окончательное результирующее множество показано ниже (только студенты JING и MAGGIE не старше двух или более студентов):

```
SNO SNAME             AGE
--- ---------- ----------
  1 AARON              20
  2 CHUCK              21
  3 DOUG               20
  5 STEVE              22
  7 BRIAN              21
  8 KAY                20
  9 GILLIAN            20
 10 CHAD               21
```

MySQL и PostgreSQL

Чтобы найти студентов, которые старше как минимум двух других студентов, используйте агрегатную функцию COUNT и связанный подзапрос:

```
1  select s1.*
2    from student s1
3   where 2 <= ( select count(*)
4                  from student s2
5                 where s2.age < s1.age )
```

DB2, Oracle и SQL Server

Чтобы найти студентов, которые старше как минимум двух других студентов, используйте ранжирующую функцию DENSE_RANK:

```
1  select sno,sname,age
2    from (
3  select sno,sname,age,
4         dense_rank()over(order by age) as dr
5    from student
6         ) x
7   where dr >= 3
```

Обсуждение

Подробное обсуждение представлено в «Запросе 5». Техники аналогичны в обоих случаях, единственное отличие состоит в том, что окончательная операция осуществляется в первом случае над счетчиком, во втором – над рангом.

Оригинальное решение

Эта задача является вариантом «Запроса 6», разница в том, что теперь мы работаем только с таблицей STUDENT. Решение «Запроса 6» можно легко адаптировать для «поиска студентов, которые старше, по крайней мере, двух других студентов», что и показано ниже:

```
select distinct s1.*
  from student s1,
       student s2,
       student s3
 where s1.age > s2.age
   and s2.age > s3.age
```

```
SNO SNAME             AGE
--- ---------- ----------
  1 AARON              20
  2 CHUCK              21
  3 DOUG               20
  5 STEVE              22
  7 BRIAN              21
  8 KAY                20
  9 GILLIAN            20
 10 CHAD               21
```

Запросы с условием «точно»

Может показаться, что решать вопросы истинности чего бы то ни было просто. Во многих случаях так оно и есть. Но иногда ответ на вопрос об абсолютной истинности, особенно если это связано с объединением данных из связанных таблиц, может вызывать затруднения. Проблема произрастает из исключающей природы условия «точно». Возможно, удобнее будет рассматривать его как условие «только». К примеру, скажем, чем отличаются люди, которые носят туфли, и которые носят только туфли? Недостаточно просто удовлетворить условию; условие должно быть выполнено с гарантией того, что ни одно из других условий не выполняется.

Запрос 9

Найти преподавателей, которые читают именно один предмет.

Эту задачу можно сформулировать иначе: «Найти преподавателей, которые читают только один предмет». Какой конкретно предмет – не важно, важно то, что только один предмет. Окончательное результирующее множество должно выглядеть следующим образом:

```
LNAME      DEPT           SALARY  AGE
---------- ---------- ---------- ----
POMEL      SCIENCE           500   65
```

MySQL и PostgreSQL

Чтобы найти преподавателей, читающих только один предмет, используйте агрегатную функцию COUNT:

```
1  select p.lname,p.dept,p.salary,p.age
2    from professor p, teach t
3   where p.lname = t.lname
4   group by p.lname,p.dept,p.salary,p.age
5  having count(*) = 1
```

DB2, Oracle и SQL Server

Чтобы найти преподавателей, читающих только один предмет, используйте оконную функцию COUNT OVER:

```
1  select lname, dept, salary, age
2    from (
3  select p.lname,p.dept,p.salary,p.age,
4         count(*) over (partition by p.lname) as cnt
5    from professor p, teach t
6   where p.lname = t.lname
7         ) x
8   where cnt = 1
```

Обсуждение

Внутренним объединением таблицы PROFESSOR с таблицей TEACH мы гарантируем, что из рассмотрения исключаются все преподаватели, не читающие ни одного курса. В решении с агрегацией функция COUNT в конструкции HAVING возвращает преподавателей, читающих только один предмет. Во втором решении используется оконная функция COUNT OVER. Но обратите внимание, что в операторе PARTITION функции COUNT OVER этого решения и в операторе GROUP BY решения с агрегацией указаны разные столбцы таблицы PROFESSOR. В данном примере операторы GROUP BY и PARTITION BY могут отличаться, поскольку все фамилии в таблице TEACHER разные, т. е. исключение полей P.DEPT, P.SALARY и P.AGE из сегмента не влияет на операцию подсчета. В предыдущих решениях я намеренно использовал в операторе PARTITION решения с оконной функцией те же столбцы, что и в операторе GROUP BY решения с агрегацией, чтобы показать, что PARTITION является скользящей, более гибкой разновидностью GROUP BY.

Оригинальное решение

В этом решении используется та же техника, что и в «Запросе 3»: ответ дается в два этапа. Первый шаг – выбрать преподавателей, которые читают два или более предмета. Второй шаг – найти преподавателей, которые читают какой-то предмет, но которых нет среди выбранных в шаге 1. Полное обсуждение приведено в «Запросе 3». Решение показано ниже:

```
select p.*
  from professor p,
       teach t
 where p.lname = t.lname
   and p.lname not in (
select t1.lname
  from teach t1,
       teach t2
 where t1.lname = t2.lname
   and t1.cno   > t2.cno
)
```

```
LNAME      DEPT           SALARY        AGE
---------- ---------- ---------- ----------
POMEL      SCIENCE           500         65
```

Запрос 10

Требуется найти студентов, которые изучают только CS112 и CS114 (именно эти два предмета и никакие другие), но следующий запрос возвращает пустое результирующее множество:

```
select s.*
  from student s, take t
 where s.sno = t.sno
   and t.cno = 'CS112'
   and t.cno = 'CS114'
```

Ни в одной строке не может быть поля с двумя значениями одновременно (предполагаем, что рассматриваются простые скалярные типы данных, такие как используются в таблице STUDENT), поэтому этот запрос никак не может обеспечить правильный результат. В своей книге Розенштейн рассуждает о том, как написание запросов «в лоб» может привести к подобным ошибкам. DOUG – единственный студент, который изучает только CS112 и CS114, и только он должен быть возвращен для этой задачи.

MySQL и PostgreSQL

Чтобы найти студентов, которые изучают только CS112 и CS114, используйте выражение CASE и агрегатную функцию COUNT:

```
1 select s.sno, s.sname, s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno, s.sname, s.age
5 having count(*) = 2
6    and max(case when cno = 'CS112' then 1 else 0 end) +
7        max(case when cno = 'CS114' then 1 else 0 end) = 2
```

DB2, Oracle и SQL Server

Чтобы найти студентов, которые изучают только CS112 и CS114, используйте оконную функцию COUNT OVER в сочетании с выражением CASE:

```
 1  select sno,sname,age
 2    from (
 3  select s.sno,
 4         s.sname,
 5         s.age,
 6         count(*) over (partition by s.sno) as cnt,
 7         sum(case when t.cno in ( 'CS112', 'CS114' )
 8                  then 1 else 0
 9             end)
10         over (partition by s.sno) as both,
11         row_number()
12         over (partition by s.sno order by s.sno) as rn
13    from student s, take t
14   where s.sno = t.sno
15         ) x
16   where cnt  = 2
17     and both = 2
18     and rn   = 1
```

Обсуждение

В решении с агрегатной функцией используется тот же подход, что и в «Запросе 1» и «Запросе 2». Внутреннее объединение таблицы STUDENT с таблицей TAKE обеспечивает исключение всех студентов, не изучающих ни одного предмета. По выражению COUNT в конструкции HAVING выбираются только студенты, изучающие именно два предмета. Посредством суммирования результатов выражений CASE определяется, сколько из заданных предметов изучает студент. Эта сумма равна 2 только для студентов, изучающих оба предмета, CS112 и CS114.

В решении с оконной функцией используется техника, аналогичная технике в «Запросе 1» и «Запросе 2». Данная версия немного отличается: здесь значение выражения CASE возвращается в оконную функцию SUM OVER, и кроме того используется ранжирующая функция ROW_NUMBER, чтобы избежать применения ключевого слова DISTINCT. Результаты этого решения без завершающих фильтров показаны ниже:

```
select s.sno,
       s.sname,
       s.age,
       count(*) over (partition by s.sno) as cnt,
       sum(case when t.cno in ( 'CS112', 'CS114' )
                then 1 else 0
           end)
       over (partition by s.sno) as both,
       row_number()
```

```
       over (partition by s.sno order by s.sno) as rn
  from student s, take t
 where s.sno = t.sno

SNO SNAME   AGE  CNT BOTH   RN
--- ------ ---- ---- ---- ----
  1 AARON    20    3    2    1
  1 AARON    20    3    2    2
  1 AARON    20    3    2    3
  2 CHUCK    21    1    1    1
  3 DOUG     20    2    2    1
  3 DOUG     20    2    2    2
  4 MAGGIE   19    2    1    1
  4 MAGGIE   19    2    1    2
  5 STEVE    22    1    0    1
  6 JING     18    2    1    1
  6 JING     18    2    1    2
```

Проанализировав результаты, можно увидеть, что окончательное результирующее множество составляют записи, в которых поля BOTH и CNT равны 2. RN может быть равным или 1, или 2, это не важно; этот столбец существует только для того, чтобы можно было отфильтровать дубликаты, не применяя DISTINCT.

Оригинальное решение

В данном решении поиск студентов, изучающих не менее трех предметов, ведется с помощью подзапроса с множеством рефлексивных объединений. Следующий шаг – использовать рефлексивное объединение таблицы TAKE, чтобы найти студентов, изучающих оба предмета, CS112 и CS114. Заключительный шаг – выбрать только тех студентов, которые изучают оба предмета, CS112 и CS114, и не изучают три или более предметов. Решение показано ниже:

```
select s1.*
  from student s1,
       take t1,
       take t2
 where s1.sno  = t1.sno
   and s1.sno  = t2.sno
   and t1.cno = 'CS112'
   and t2.cno = 'CS114'
   and s1.sno not in (
select s2.sno
  from student s2,
       take t3,
       take t4,
       take t5
 where s2.sno = t3.sno
   and s2.sno = t4.sno
   and s2.sno = t5.sno
   and t3.cno > t4.cno
```

```
   and t4.cno > t5.cno
)

SNO SNAME      AGE
--- ---------- ---
  3 DOUG        20
```

Запрос 11

Требуется найти студентов, которые старше двух (и только двух) других студентов. Эту задачу можно сформулировать иначе: требуется найти третьего самого младшего студента(ов). Должно быть получено такое результирующее множество:

```
SNO SNAME             AGE
--- ---------- ----------
  1 AARON              20
  3 DOUG               20
  8 KAY                20
  9 GILLIAN            20
```

MySQL и PostgreSQL

Чтобы найти третьего самого младшего студента, используйте агрегатную функцию COUNT и связанный подзапрос:

```
1  select s1.*
2    from student s1
3   where 2 = ( select count(*)
4                  from student s2
5                 where s2.age < s1.age )
```

DB2, Oracle и SQL Server

Чтобы найти третьего самого младшего студента, используйте ранжирующую функцию DENSE_RANK:

```
1  select sno,sname,age
2    from (
3  select sno,sname,age,
4         dense_rank()over(order by age) as dr
5    from student
6         ) x
7   where dr = 3
```

Обсуждение

В решении с агрегатной функцией для поиска всех студентов, которые старше двух (и только двух) других студентов, используется скалярный подзапрос. Чтобы понять, что происходит, перепишем решение с использованием скалярного подзапроса. В следующем примере столбец CNT представляет, сколько студентов младше текущего:

```
select s1.*,
       (select count(*) from student s2
```

```
        where s2.age < s1.age) as cnt
  from student s1
  order by 4
```

```
SNO SNAME             AGE         CNT
--- ---------- ---------- ----------
  6 JING               18          0
  4 MAGGIE             19          1
  1 AARON              20          2
  3 DOUG               20          2
  8 KAY                20          2
  9 GILLIAN            20          2
  2 CHUCK              21          6
  7 BRIAN              21          6
 10 CHAD               21          6
  5 STEVE              22          9
```

Такая запись решения позволяет увидеть, кто из студентов является третьим по возрасту, начиная с самых младших (те, для которых поле CNT равно 2).

Решение с использованием ранжирующей функции DENSE_RANK аналогично решению со скалярным подзапросом в том, что все строки ранжируются на основании количества студентов, младше текущего студента (допускается повторение рангов для дублирующихся значений, и нет пропусков). Следующий запрос демонстрирует результат выполнения функции DENSE_RANK:

```
select sno,sname,age,
       dense_rank()over(order by age) as dr
  from student
```

```
SNO SNAME             AGE          DR
--- ---------- ---------- ----------
  6 JING               18           1
  4 MAGGIE             19           2
  1 AARON              20           3
  3 DOUG               20           3
  8 KAY                20           3
  9 GILLIAN            20           3
  2 CHUCK              21           4
  7 BRIAN              21           4
 10 CHAD               21           4
  5 STEVE              22           5
```

Заключительный шаг – поместить запрос во вложенный запрос и выбрать только те строки, в которых DR равно 3.

Оригинальное решение

В оригинальном решении используется двухэтапный подход: шаг 1 – находим студентов, которые старше трех или более студентов; шаг 2 – находим студентов, которые старше двух студентов и которых нет сре-

ди студентов, возвращенных в результате шага 1. Розенштейн перефразировал бы это так: «Находим студентов, которые старше, по крайней мере, двух студентов и не старше, по крайней мере, трех студентов». Решение показано ниже:

```
select s5.*
  from student s5,
       student s6,
       student s7
 where s5.age > s6.age
   and s6.age > s7.age
   and s5.sno not in (
select s1.sno
  from student s1,
       student s2,
       student s3,
       student s4
 where s1.age > s2.age
   and s2.age > s3.age
   and s3.age > s4.age
)
```

```
SNO SNAME      AGE
--- ---------- ---
  1 AARON       20
  3 DOUG        20
  9 GILLIAN     20
  8 KAY         20
```

В приведенном выше решении используется техника, показанная в «Запросе 5», где подробно обсуждается, как выявлять предельные значения с помощью рефлексивных объединений.

Запросы с условием «любой» или «все»

В задачах с условием «любой» или «все» обычно требуется найти строки, полностью удовлетворяющие одному или более условиям. Например, если необходимо найти людей, которые едят все овощи, по сути, выбираются люди, для которых нет овощей, которые они бы не ели. Такой тип постановки задачи обычно классифицируют как *реляционное деление*. В задачах в условием «любой» очень важно то, как сформулирован вопрос. Рассмотрим разницу между следующими двумя требованиями: «студент, который изучает любой предмет» и «самолет, который быстрее любого поезда». В первом случае подразумевается, что «необходимо найти студента, который изучает не менее одного предмета», тогда как во втором случае «необходимо найти самолет, который быстрее всех поездов».

Запрос 12

Требуется найти студентов, которые изучают все предметы.

Количество предметов, изучаемых студентом, в таблице TAKE должно быть равно общему количеству предметов в таблице COURSES. В таблице COURSES три предмета. Только студент AARON изучает всех их, и он должен быть единственным студентом в результате. Окончательное результирующее множество будет таким:

```
SNO SNAME   AGE
--- ------ ----
  1 AARON    20
```

MySQL и PostgreSQL

Чтобы найти студентов, изучающих все предметы, используйте агрегатную функцию COUNT:

```
1 select s.sno,s.sname,s.age
2   from student s, take t
3  where s.sno = t.sno
4  group by s.sno,s.sname,s.age
5 having count(t.cno) = (select count(*) from courses)
```

DB2 и SQL Server

Используйте оконную функцию COUNT OVER и внешнее объединение вместо подзапроса:

```
 1  select sno,sname,age
 2    from (
 3  select s.sno,s.sname,s.age,
 4         count(t.cno)
 5         over (partition by s.sno) as cnt,
 6         count(distinct c.title) over() as total,
 7         row_number() over
 8         (partition by s.sno order by c.cno) as rn
 9    from courses c
10         left join take t    on (c.cno = t.cno)
11         left join student s on (t.sno = s.sno)
12         ) x
13   where cnt = total
14     and rn  = 1
```

Oracle

Для Oracle 9*i* и последующих версий может использоваться решение для DB2. В качестве альтернативы в этих версиях применим собственный синтаксис Oracle для внешнего объединения. Но для пользователей 8*i* и более ранних версий это будет единственно возможное решение:

```
 1   select sno,sname,age
 2     from (
 3   select s.sno,s.sname,s.age,
 4          count(t.cno)
 5          over (partition by s.sno) as cnt,
 6          count(distinct c.title) over() as total,
```

```
 7          row_number() over
 8          (partition by s.sno order by c.cno) as rn
 9     from courses c, take t, student s
10    where c.cno = t.cno (+)
11      and t.sno = s.sno (+)
12          )
13    where cnt = total
14      and rn  = 1
```

Обсуждение

В решении с агрегатной функцией для получения общего числа доступных предметов используется подзапрос. Внешний запрос выбирает только тех студентов, которые изучают такое же количество предметов, как возвратил подзапрос. В решении с оконной функцией применен другой подход: вместо подзапроса используется внешнее объединение с таблицей COURSES. Здесь оконные функции также возвращают количество предметов, изучаемых студентом (под псевдонимом CNT), вместе с общим числом предметов, представленных в таблице COURSES (под псевдонимом TOTAL). Запрос ниже показывает промежуточные результаты, возвращаемые этими оконными функциями:

```
select s.sno,s.sname,s.age,
       count(distinct t.cno)
       over (partition by s.sno) as cnt,
       count(distinct c.title) over() as total,
       row_number()
       over(partition by s.sno order by c.cno) as rn
  from courses c
       left join take t    on (c.cno = t.cno)
       left join student s on (t.sno = s.sno)
 order by 1
```

```
SNO SNAME   AGE  CNT      TOTAL   RN
--- ------ ---- ---- ---------- ----
  1 AARON    20    3          3    1
  1 AARON    20    3          3    2
  1 AARON    20    3          3    3
  2 CHUCK    21    1          3    1
  3 DOUG     20    2          3    1
  3 DOUG     20    2          3    2
  4 MAGGIE   19    2          3    1
  4 MAGGIE   19    2          3    2
  5 STEVE    22    1          3    1
  6 JING     18    2          3    1
  6 JING     18    2          3    2
```

Студент, изучающий все предметы, – это тот, значение CNT которого равно значению TOTAL. Для отсеивания дубликатов из окончательного результирующего множества вместо DISTINCT используется функция ROW_NUMBER. Строго говоря, во внешних объединениях таблиц TAKE и STUDENT нет необходимости, поскольку нет таких предме-

тов, которые не изучались бы, по крайней мере, одним студентом. Если бы такой предмет (который не изучает ни один студент) существовал, для него CNT не равнялось бы TOTAL, и в столбцах SNO, SNAME и AGE было бы возвращено значение NULL. В примере ниже создается новый предмет, не изучаемый ни одним студентом. Следующий запрос демонстрирует промежуточный результат, который был бы получен в таком случае (для ясности включен столбец C.TITLE):

```
insert into courses values ('CS115','BIOLOGY',4)
select s.sno,s.sname,s.age,c.title,
       count(distinct t.cno)
       over (partition by s.sno) as cnt,
       count(distinct c.title) over() as total,
       row_number()
       over(partition by s.sno order by c.cno) as rn
  from courses c
       left join take t    on (c.cno = t.cno)
       left join student s on (t.sno = s.sno)
 order by 1
```

```
SNO SNAME  AGE TITLE      CNT TOTAL  RN
--- ------ --- ---------- --- ----- ---
  1 AARON   20 PHYSICS      3     4   1
  1 AARON   20 CALCULUS     3     4   2
  1 AARON   20 HISTORY      3     4   3
  2 CHUCK   21 PHYSICS      1     4   1
  3 DOUG    20 PHYSICS      2     4   1
  3 DOUG    20 HISTORY      2     4   2
  4 MAGGIE  19 PHYSICS      2     4   1
  4 MAGGIE  19 CALCULUS     2     4   2
  5 STEVE   22 CALCULUS     1     4   1
  6 JING    18 CALCULUS     2     4   1
  6 JING    18 HISTORY      2     4   2
               BIOLOGY      0     4   1
```

Из этого результата видно, что после применения окончательных фильтров не будет возвращено ни одной строки. Необходимо помнить, что оконные функции обрабатываются после оператора WHERE, поэтому важно при подсчете общего количества доступных предметов в таблице COURSES использовать ключевое слово DISTINCT (в противном случае будет получено общее для результирующего множества, что будет являться общим числом предметов, изучаемых всеми студентами, т. е. select count(cno) from take).

В данных, используемых для данного примера, в таблице TAKE нет дублирующихся значений, поэтому предлагаемое решение работает замечательно. Если бы в таблице TAKE были дубликаты, например, студент, трижды изучающий один и тот же предмет, решение дало бы сбой. Справиться с дубликатами в данном решении не составляет труда: просто добавляем DISTINCT при подсчете по T.CNO, и решение будет работать корректно.

Оригинальное решение

В оригинальном решении Розенштейн уходит от агрегатов, дьявольски разумно используя декартово произведение. Приведенный ниже запрос основывается на оригинале:

```
select *
  from student
 where sno not in
      ( select s.sno
          from student s, courses c
         where (s.sno,c.cno) not in (select sno,cno from take) )
```

Розенштейн изменил формулировку задачи: «Кого из студентов нет среди тех, для которых существует предмет, которого они не изучают?» Если взглянуть на задачу таким образом, получается, что мы имеем дело с отрицанием. Вспомним, как Розенштейн предлагает обрабатывать отрицание:

> Помните, настоящее отрицание требует двух проходов: чтобы найти, «кто не», сначала надо найти, «кто да», и затем избавиться от них.

Самый внутренний подзапрос возвращает все действительные сочетания SNO/CNO. Средний подзапрос, который использует декартово произведение таблиц STUDENT и COURSES, возвращает всех студентов и все предметы (т. е. все возможные пары студент-предмет) и отсеивает действительные сочетания SNO/CNO (оставляя только «фиктивные» сочетания SNO/CNO). Самый внешний запрос возвращает только те строки таблицы STUDENT, значений SNO которых нет среди возвращенных средним подзапросом. Следующие запросы, возможно, внесут ясность в решение. Для удобства здесь участвуют только записи студентов AARON и CHUCK (только AARON изучает все предметы):

```
select *
  from student
 where sno in ( 1,2 )
```

```
SNO SNAME       AGE
--- ---------- ----
  1 AARON        20
  2 CHUCK        21
```

```
select *
  from take
 where sno in ( 1,2 )
```

```
SNO CNO
--- -----
  1 CS112
  1 CS113
  1 CS114
  2 CS112
```

```
select s.sno, c.cno
```

```
  from student s, courses c
 where s.sno in ( 1,2 )
 order by 1

SNO CNO
--- -----
  1 CS112
  1 CS113
  1 CS114
  2 CS112
  2 CS113
  2 CS114
```

Данные запросы возвращают записи студентов AARON и CHUCK из таблицы STUDENT, предметы, изучаемые студентами AARON и CHUCK, и декартово произведение этих студентов и всех предметов соответственно. Декартово произведение для AARON соответствует результирующему множеству, возвращенному для него из таблицы TAKE, тогда как в декартовом произведении для CHUCK появляются две «фиктивные» строки, которым нет соответствия в таблице TAKE. Следующий запрос представляет средний подзапрос. В нем для выбора действительных сочетаний SNO/CNO используется оператор NOT IN:

```
select s.sno, c.cno
  from student s, courses c
 where s.sno in ( 1,2 )
   and (s.sno,c.cno) not in (select sno,cno from take)

SNO CNO
--- -----
  2 CS113
  2 CS114
```

Обратите внимание, что средний подзапрос не возвращает записи студента AARON (потому что AARON изучает все предметы). Результирующее множество среднего подзапроса содержит строки, поскольку они образованы в результате декартова произведения, а не потому что студент CHUCK на самом деле изучает эти предметы. Самый внешний запрос возвращает из таблицы STUDENT строки, содержащие такие значения в поле SNO, которых нет среди значений SNO, возвращенных средним подзапросом:

```
select *
  from student
 where sno in ( 1,2 )
   and sno not in
       (select s.sno from student s, courses c
         where s.sno in ( 1,2 )
           and (s.sno,c.cno) not in (select sno,cno from take))

SNO SNAME       AGE
--- ---------- ----
  1 AARON        20
```

Запрос 13

Требуется найти студентов, которые старше любого другого студента.

Эту задачу можно сформулировать иначе: «Найти самых старших студентов». Должно быть получено следующее результирующее множество:

```
SNO SNAME       AGE
--- ---------- ----
  5 STEVE        22
```

MySQL и PostgreSQL

Чтобы найти самых старших студентов, используйте в подзапросе агрегатную функцию MAX:

```
1 select *
2   from student
3  where age = (select max(age) from student)
```

DB2, Oracle и SQL Server

Чтобы найти самых старших студентов, используйте во вложенном запросе оконную функцию MAX OVER:

```
1  select sno,sname,age
2    from (
3  select s.*,
4         max(s.age)over() as oldest
5    from student s
6         ) x
7   where age = oldest
```

Обсуждение

В обоих решениях для поиска самого старшего студента используется функция MAX. В решении с подзапросом сначала по таблице STUDENT определяется наибольший возраст студентов. Подзапрос возвращает это значение во внешний запрос, который уже ведет поиск студентов соответствующего возраста. Версия решения с оконной функцией делает то же самое, но возвращает наибольший возраст для каждой строки. Вот промежуточные результаты запроса с оконной функцией:

```
select s.*,
       max(s.age) over() as oldest
  from student s
```

```
SNO SNAME       AGE     OLDEST
--- ---------- ---- ----------
  1 AARON        20         22
  2 CHUCK        21         22
  3 DOUG         20         22
  4 MAGGIE       19         22
  5 STEVE        22         22
```

```
 6 JING         18         22
 7 BRIAN        21         22
 8 KAY          20         22
 9 GILLIAN      20         22
10 CHAD         21         22
```

Чтобы выбрать самых старших студентов, просто выбираем строки, в которых AGE = OLDEST.

Оригинальное решение

В оригинальном решении поиск всех студентов, для которых есть старший студент, осуществляется посредством рефлексивного объединения в подзапросе. Внешний запрос возвращает всех студентов таблицы STUDENT, которых нет среди выбранных подзапросом. Эту операцию можно сформулировать как «найти всех студентов, которых нет среди тех студентов, которые младше, чем, по крайней мере, один студент»:

```
select *
  from student
 where age not in (select a.age
                     from student a, student b
                    where a.age < b.age)
```

Подзапрос использует декартово произведение, чтобы найти все возрасты в A, которые меньше, чем все возрасты в B. Единственный возраст, который не будет меньше любого другого, – наибольший. Его подзапрос не возвращает. Во внешнем запросе с помощью оператора NOT IN выбираются все строки таблицы STUDENT, значений AGE которых нет среди значений AGE, возвращенных подзапросом (если возвращается A.AGE, это означает, что в таблице STUDENT есть значение AGE, больше данного). Если есть какие-то трудности с пониманием того, как все это получается, рассмотрим следующий запрос. По сути, оба запроса работают одинаково, но приведенный ниже, наверное, проще:

```
select *
  from student
 where age >= all (select age from student)
```

Алфавитный указатель

Специальные символы

% (остаток от деления), функция (SQL Server), 218, 322, 326
% (подстановка), оператор, 46
+ (конкатенация), оператор (SQL Server), 39, 344
_ (подчеркивание), оператор, 46
|| (конкатенация), оператор (DB2/Oracle/PostgreSQL), 39, 343

A

ADD_MONTHS, функция (Oracle), 232, 233, 273, 315, 332
ADDDATE, функция (MySQL), 273, 316, 326, 335
AVG, функция, 197

C

CAST, функция (SQL Server), 326
CEIL, функция (DB2/MySQL/Oracle/PostgreSQL), 443, 444
CEILING, функция (SQL Server), 443, 482
COALESCE, функция, 44, 97, 198, 223, 357
CONCAT, функция (MySQL), 39, 325, 343
CONNECT BY, блок (Oracle)
  поддержка версиями, 247
CONNECT BY, оператор (Oracle)
  альтернативы, 516
  в иерархических структурах, 519, 523
  и вложенные представления, 374
  и оператор WITH, 374, 412
  поддержка версиями, 302, 329
CONNECT_BY_ISLEAF, функция (Oracle), 524, 528
CONNECT_BY_ROOT, функция (Oracle), 524, 528
COUNT OVER, оконная функция, 263, 486, 603
COUNT, функция, 203, 205, 597, 607
CREATE TABLE ... LIKE, команда (DB2), 104
CREATE TABLE, команда, 103
CUBE, расширение, 467, 477
CURRENT_DATE, функция (DB2/MySQL/PostgreSQL), 251, 294, 412

D

DATE, функция (DB2), 322
DATE_ADD, функция (MySQL), 233, 237, 270, 316, 318
DATE_FORMAT, функция (MySQL), 237, 251, 276, 340
DATE_TRUNC, функция (PostgreSQL), 254, 269, 274, 279, 334
DATEADD, функция (MySQL), 251
DATEADD, функция (SQL Server), 233, 273, 280
DATEDIFF, функция (MySQL/SQL Server), 235, 242, 244, 258, 273
DATENAME, функция (SQL Server), 238, 248, 257, 338, 340
DATEPART, функция (SQL Server), 275, 277, 286, 316, 319
DAY, функция (DB2), 275, 277, 279
DAY, функция (MySQL), 271, 278, 279, 326
DAY, функция (SQL Server), 272, 278, 280
DAYNAME, функция (DB2/MySQL/SQL Server), 236, 338
DAYOFWEEK, функция (DB2/MYSQL), 293, 339
DAYOFYEAR, функция (DB2/MySQL/SQL Server), 267, 272, 331, 335, 336

DAYS, функция (DB2), 244, 258, 272
DB2
  значения DATE в операторе ORDER BY, 489
  специальные рецепты, 23
DECODE, функция (Oracle), 386, 562, 588
DEFAULT VALUES, блок (PostgreSQL/SQL Server), 101
DEFAULT, ключевое слово, 101
DELETE, команда, 115, 116
DENSE_RANK OVER, ранжирующая функция (DB2/Oracle/SQL Server), 390, 400, 551, 552
DENSE_RANK, функция (DB2/Oracle/SQL Server), 389, 405, 409
DISTINCT, ключевое слово
  альтернативы, 63, 401
  и список SELECT, 49, 403, 592
  применение, 69, 87, 389

E

EXCEPT, функция, 68, 69, 77, 80
EXTRACT, функция (PostgreSQL/MySQL), 242, 370

G

GENERATE_SERIES, функция (PostgreSQL)
  альтернативы, 255, 305, 333, 521
  использование, 248
  параметры, 376
  применение, 265, 269, 284, 375
GETDATE, функция (SQL Server), 412, 414
GROUP BY, запросы, возвращение столбцов, не вошедших в них, 461
GROUP BY, оператор, 49, 403, 599, 601
GROUPING SETS, расширение (DB2/Oracle), 471, 473
GROUPING, функция (DB2/Oracle/SQL Server), 463, 464, 477, 498
GROUPING, функция (MySQL/PostgreSQL), 497

H

HOUR, функция (DB2), 275

I

IF-ELSE, операции, 39
INSERT ALL, выражение (Oracle), 105
INSERT FIRST, выражение (Oracle), 105
INSERT, выражение, 100, 103
INSTR, функция (Oracle), 541, 542, 546
INTERSECT, операция, 66, 67
INTERVAL, ключевое слово, 232, 233, 234
IS NULL, оператор, 44
ITERATE, команда (Oracle), 375
ITERATION_NUMBER, функция (Oracle), 375

J

JOIN, оператор
  в операторе FROM, 65
  поддержка Oracle, 370

K

KEEP, расширение (Oracle), 214, 215, 408, 409

L

LAG OVER, оконная функция (Oracle), 354, 366, 367, 397, 398, 399, 437
LAG, функция (Oracle), 260
LAST, функция (Oracle), 405, 409
LAST_DAY, функция (MySQL/Oracle), 265, 266, 278, 279, 288
LEAD OVER, оконная функция (Oracle)
  варианты, 361, 395, 398, 399
  и дубликаты, 360
  и рефлексивные объединения, 345, 347, 349, 352
  определение временных интервалов, 260
  поддержка версиями, 259
  преимущества, 260
  применение, 352, 354, 394, 395, 397
LEAD, функция (Oracle), 260, 261
LIKE, оператор, 45
LIMIT, оператор (MySQL/PostgreSQL), 41, 379, 381
LPAD, функция (Oracle/PostgreSQL/MySQL), 359, 451
LTRIM, функция (Oracle), 510

M

MAX OVER, оконная функция, 199,391, 398, 586, 587
MAX, функция, 199, 239

MEASURES, подоператор оператора MODEL (Oracle), 577
MEDIAN, функция (Oracle), 217
MERGE, выражение, 99, 114
MIN OVER, оконная функция (DB2/Oracle/SQL Server), 371, 391, 392, 398
MIN, функция, 199
MINUS, операция, 69, 77, 78, 80
MINUTE, функция (DB2), 275
MOD, функция (DB2), 218
MODEL, оператор (Oracle)
  использование, 211
  компоненты, 579
  поддержка версиями, 209, 211
  применение, 375, 536, 540, 548, 550
MODEL, подоператор (Oracle)
  компоненты, 577
MONTH, функция (DB2/MySQL), 241, 267, 275, 294
MONTHNAME, функция (DB2/MySQL), 338, 339
MONTHS_BETWEEN, функция (Oracle), 242, 244
MySQL
  специальные рецепты, 24

N

NEXT_DAY, функция (Oracle), 288, 292, 293
NOT EXISTS, предикат, 117
NOT IN, оператор, 70
NROWS, функция (DB2/SQL Server), 414
NTILE, оконная функция (Oracle/SQL Server), 446, 448
NULL, значения, 596
  вставка записей, 102
  и агрегатные функции, 223, 597, 607
  и группы, 596
  и оконные функции, 606
  и оператор NOT IN, 70
  и операции OR, 71
  и сортировка, 53, 60
  и функции MIN/MAX, 200
  и функция AVG, 198
  и функция COUNT, 203, 205
  и функция SUM, 202
  парадокс значений, 595
  сравнения, 44, 504
  удаление (DB2/Oracle/SQL Server), 426
  удаление (PostgreSQL/MySQL), 429, 430
NVL, функция (Oracle), 359

O

OFFSET, оператор (MySQL/PostgreSQL), 379, 381
Oracle
  и блок CONNECT BY, 247
  и внешние объединения, 73, 92, 96, 329, 387, 648
  и ключевое слово DEFAULT, 101
  и оконная функция LEAD OVER, 259
  и оператор CONNECT BY, 302, 329, 374
  и оператор JOIN, 370, 386
  и оператор KEEP, 214, 408
  и оператор MODEL, 209, 548
  и регулярные выражения, 581
  и типы TIMESTAMP, 233
  и функции CONNECT_BY_ROOT/CONNECT_BY_ISLEAF, 524
  и функции MEDIAN/PERCENTILE_CONT, 217, 220
  объектные типы, 567
  специальные рецепты, 24
ORDER BY, оператор, 43, 47, 49, 489, 607
OVER, ключевое слово, 59, 603

P

PARTITION BY, оператор, 604, 607
PERCENTILE_CONT, функция (Oracle), 217, 220
PIVOT, оператор (SQL Server), 532, 534
PostgreSQL
  специальные рецепты, 24
PRIOR, ключевое слово (Oracle), 509

Q

QUARTER, функция (DB2/MySQL), 316, 317, 319

R

RAND, функция, 42
RANDOM, функция (PostgreSQL), 42
RANGE BETWEEN, оператор, 556, 609
RANGE, оператор BETWEEN, 613
RANK OVER, ранжирующая функция, 400
RATIO_TO_REPORT, функция (Oracle), 574

REGEXP_REPLACE, функция (Oracle), 583
REPEAT, функция (DB2), 451
REPLACE, функция, 52, 140, 141, 156, 181
REPLICATE, функция (SQL Server), 451
ROLLUP, расширение GROUP BY (DB2/Oracle), 462,477, 496
ROW_NUMBER OVER, ранжирующая функция (DB2/Oracle/SQL Server)
  и оператор ORDER BY, 402
  применение, 379, 400, 427
  уникальность результата, 383, 390
ROW_NUMBER, ранжирующая функция (DB2/Oracle/SQL Server), 218
  применение, 380
ROWNUM, функция (Oracle), 41, 318, 381
RPAD, функция (Oracle), 553
RTRIM, функция (Oracle/PostgreSQL), 338
RULES, подоператор (Oracle), 539

S

SECOND, функция (DB2), 275
SELECT, выражения
  и GROUP BY, 200, 599, 601
  и ключевое слово DISTINCT, 49, 403, 592
  неполные, 25
  символ *, 33
  см. также извлечение записей, 33
  условная логика, 39
SIGN, функция (MySQL/PostgreSQL), 294
SQL Server
  отчеты с перекрестными ссылками
    см. отчеты с перекрестными ссылками, 532
  специальные рецепты, 24
START WITH, оператор (Oracle), 514, 523
STR_TO_DATE, функция (MySQL), 325
SUBSTR, функция (DB2/MySQL/Oracle/PostgreSQL), 50, 155, 171
SUBSTRING, функция (SQL Server), 50, 322, 326
SUM OVER, оконная функция (DB2/Oracle/SQL Server), 89, 92, 207, 209, 221
SUM, функция, 201, 241
SYS_CONNECT_BY_PATH, функция (Oracle), 167, 506, 530

T

TIMESTAMP, типы (Oracle), 233
TO_BASE, функция (Oracle), 548
TO_CHAR, функция (Oracle/PostgreSQL), 237, 276, 293
TO_DATE, функция (Oracle/PostgreSQL), 323, 324
TO_NUMBER, функция (Oracle/PostgreSQL), 276, 369
TRANSLATE, функция (DB2/Oracle/PostgreSQL), 52, 140, 141, 148, 156
TRUNC, функция (Oracle), 253, 274, 279, 318

U

UNION ALL, операция, 63, 78, 598
UNION, операция, 63, 64, 80, 598
UNPIVOT, оператор (SQL Server), 534, 536
UPDATE, выражение, 108, 109, 111, 112, 113

V

VALUES, оператор, 100

W

WHERE, оператор, 37
WITH ROLLUP (SQL Server/MySQL), 462
WITH, оператор (DB2/SQL Server), 519, 522, 523
WITH, оператор (Oracle), 284

X

XML, 19

Y

YEAR, функция (DB2/MySQL/SQL Server), 241, 275, 369, 371
YS_CONNECT_BY_PATH, функция (Oracle), 509

А

абстракция, аксиома, 595
агрегатные функции
  и группировка, 599, 600
  и значения NULL, 223, 597, 607
  и несколько таблиц, 85, 89, 90, 93
  оператор WHERE, использование, 37
  определение обрабатываемых строк, 604, 606
  сравнение с оконными функциями, 603
аксиома абстракции, 593, 595
аксиома Фреге, 593, 595
аналитические функции (Oracle), 601
антиобъединения, 73

Б

бизнес-логика, включение, 357
буквенно-цифровые данные, 50

В

версии обсуждаемые
  DB2, 23
  MySQL, 24
  Oracle, 24
  PostgreSQL, 24
  SQL Server, 24
версии, SQL, 23
вложенные представления
  обращение к столбцам по псевдонимам, 37
  преобразование данных, 584
  присваиваине имен, 27
  присваивание имен, 568
внешние ключи, получение списка, 126, 130
внешние объединения
  логика OR в них, 384, 387
  синтаксис Oracle, 73, 92, 96, 329, 387, 648
внутренние объединения, 64, 385
время
  группировка строк, 481, 484
время, даты, 481
вставка записей
  блокировка, 106
  в несколько таблиц, 104, 106
  копирование из другой таблицы, 103
  новых, 100
  со значениями NULL, 102
  со значениями по умолчанию, 100
выделение, схема аксиом, 594
вычисление промежуточной суммы, 205, 208, 228, 230
вычисление процентного соотношения, 220, 223, 573, 575

Г

Генник Джонатан, 507, 582
Георг Кантор, 14
гистограммы
  вертикальные, 453, 456
  горизонтальные, 451, 452
группировка, 596
  и агрегатные функции, 599, 600
  и значения NULL, 596
  и оператор SELECT, 200, 599, 601
  и функция COUNT, 203
  и функция SUM, 201
  определение, 589, 590
  основания, 590
  по интервалам времени, 481, 484
  примеры, 589
  характеристики, 591, 594

Д

данные с разделителями, преобразование в список оператора IN, 168, 174
даты
  арифметика, 231, 263
    будние дни в году, подсчет, 257
    дней в году, подсчет (Oracle), 543
    дни недели в году, подсчет, 246
    добавление/вычитание, 231, 234
    интервал между записью и следующей записью, 258, 263
    месяцы/годы между датами, 241
    рабочие дни между датами, 236, 241
    разность между датами, 234
    секунды/минуты/часы между датами, 244
  и оператор ORDER BY (DB2), 489
  работа с ними, 264, 347
    високосный год, 265, 272
    год, количество дней, 272, 275
    даты начала и конца квартала, 314, 319, 320, 327
    диапазоны дат, выявление наложений, 342, 347

даты
работа с ними
дни недели, все даты, выпадающие на определенный день недели, 280, 287
дни недели, первый/последний в месяце, 287, 295
единицы времени, извлечение, 275
единицы времени, поиск, 337
календарь, 295, 313
месяц, первый/последний дни, 277
отсутствующие даты, дополнение, 327, 337
сравнение записей, 339, 342
формат, 231
декартовы произведения, 83
диапазоны
начало/конец, определение, 363, 368
последовательные числовые значения, формирование, 373, 377
последовательные, выявление, 348, 354
пропущенные значения, вставка, 368, 373
разность между значениями строк группы, 354, 363
динамический SQL, создание, 130
дубликаты
исключение, 401, 403
уничтожение, 117

З

зависящие от данных ключи, сортировка, 60
задача с парикмахером, 593
записи
вставка
см. вставка записей, 99
извлечениеизвлечениезаписей, 99
обновление
см. изменение записей, 99
слияние, 115
сортировка
см. сортировка записей, 99
уничтожение
см. уничтожение записей, 115
запросы
GROUP BY, возвращение столбцов, не вошедших в них, 456
«любой» или «все», 647, 654
«не более», 628, 634
«не менее», 634, 640
по отрицанию, 619, 628
A или B, но не оба, 622, 625
не A, 619, 622
только A, 625, 628
«точно», 640
«точно равно», 647

И

иерархии
отношения прародитель-родитель-потомок, 505, 510
отношения родитель-потомок, 501, 505
родительские строки, поиск дочерних строк, 519, 523
сложная природа, 500
создание представлений, 510, 519
сравнение древовидной и рекурсивной структур данных, 523
тип узла, определение, 523, 531
извлечение записей
простое, 33, 46
все строки/столбцы, 33
из нескольких таблиц
см. несколько таблиц, 62
неопределенные значения, 43
случайные, 42, 43
соответствия шаблонам, 45
столбцы, 33, 35, 38
присваивание имен возвращаемым, 36
строки, 34
ограничение числа, 40
упорядочивание результатов
см. сортировка записей, 47
условная логика, 39
сложные вопросы, 420, 499
агрегация скользящего диапазона значений, 487, 494
гистограммы,
вертикальные, 453, 456
горизонтальные, 451, 452
группировка строк по интервалам времени, 481, 484
группы фиксированного размера, организация данных в них, 441, 444
группы, создание заданного количества, 445, 450
одновременная агрегация групп/сегментов, 485, 487

извлечение записей
сложные вопросы
повторяющиеся значения, исключение, 436, 440
подсуммы для всех сочетаний, вычисление, 466, 475
подсуммы, простые, вычисление, 462, 466
разреженные матрицы, создание, 480
столбцы, не перечисленные в GROUP BY, возвращение, 456, 461
строки, маркировка, 478
строки, не представляющие подсуммы, выявление, 476
см. также разворачивание, 420
изменение записей
значениями из другой таблицы, 110, 113
изменение данных строки, 107
использование запросов для получения новых значений, 112
когда существуют соответствующие строки, 109
изменение имен в результатах запроса
см. псевдонимы, 36
имена, извлечение инициалов, 150, 154
индексы
получение списка, 123
создание, 18
информационная схема (MySQL/PostgreSQL/SQL Server), 122

К

кадрирование, оператор, 613
Кайт, Том, 548
календари, создание, 295, 313
Кантор, Георг, 14
каталоги
см. метаданные, 121
ключевое слово AS, 36
ключи
внешние, 126, 130
зависящие от данных, 60
сохранение, 112
конкатенация
оператор (+) (SQL Server), 39, 344
оператор (||) (DB2/Oracle/PostgreSQL), 39, 343
см. также функция CONCAT, 39
столбцы, 38
контактная информация, 28

Л

логарифмы, 209
логика, включение бизнес-логики, 357

М

максмимальные значения, поиск
см. MAX, функция, 199
матрицы, разреженные, создание, 480
медианы, вычисление, 216, 220
метаданные, 121, 133
внешние ключи без индексов, 126, 130
генерирование SQL, 130
индексированные столбцы таблицы, 123
наложенные на таблицу ограничения, 125
словарь данных, описание, 132, 133
столбцы таблицы, 122
таблицы схемы, 121
минимальные значения, поиск, 199, 371, 391, 392, 398
множества с повторяющимися элементами, 597
мода, вычисление, 213, 216
мультимножества, 597, 598

Н

несколько таблиц
извлечение данных, 62
значения, которых нет в других таблицах, 72
столбцы одного типа, 62

О

обновление записей
см. изменение записей, 108
обратное разворачивание результирующих множеств, 431, 435
объединения
анти-, 73
внешние, 73
внутренние, 64, 385
выбор столбцов, 67
и скалярные подзапросы, 76
описание, 62
рефлексивные
см. рефлексивные объединения, 344
экви-, 64, 72

объектно-ориентированные
возможности, 19
ограничения
наложение, 18
получение списка, 125
оконные функции
и значения NULL, 606, 607
и отчеты, 614
определение временных интервалов, 260
платформы их поддерживающие, 551, 556
поведение, 601
поддерживающие их платформы, 434, 489
подоператор ORDER BY, 607
порядок обработки, 352, 487, 603
преимущества, 459, 554, 613, 616
сегменты, 604, 607
специальные функции, 89
сравнение с агрегатными функциями, 603
оперативного анализа данных (OLAP)
функции (DB2), 601
оператор кадрирования, 556, 609
операции над множествами, 78
в общем, 62, 63, 67
операции над числами
нахождение/вычисление
доли от целого в процентном выражении, 573, 575
медианы, 216, 220
минимального/максмимального значения, 199
моды, 213, 216
подсуммы для всех сочетаний, 466, 475
подсуммы, простой, 462, 466
промежуточной суммы, 205, 208
изменение значений, 228, 230
промежуточного произведения, 209, 212
промежуточной разности, 212
процента от целого, 220, 223
среднего, 197
среднего без учета наибольшего/наименьшего значений, 224
столбцов, которые могут содержать значения NULL, агрегация, 223
подсчет
значений столбца, 205
строк, 203
преобразование
буквенно-цифровых строк, 226
целых чисел в их двоичное представление (Oracle), 547, 550
суммирование значений столбца, 201
остаток от деления, вычисление
%, оператор (SQL Server), 218
%, функция (SQL Server), 322, 326
MOD, функция (DB2), 218
ROW_NUMBER, функция (DB2/SQL Server), 218
отчеты с перекрестными ссылками
обратное разворачивание (SQL Server), 534, 536
создание (SQL Server), 532, 534

П

парадокс значений NULL, 595, 599
парадокс Рассела, 593
переносимость, код, 19
подзапросы
преобразование скалярных в составные (Oracle), 566, 568
принудительная последовательность выполнения, 584
связанные, 71
подстановки оператор (%), 46
подсуммы
вычисление для всех сочетаний, 466, 475
простые, вычисление, 462, 466
разворачивание содержащего их результирующего множества, 495, 499
подчеркивания оператор (_), 46
поиск, 378, 419
верхние n записей, выбор, 389, 391
внешние объединения, логика OR в них, 384, 387
дубликаты, исключение, 401, 403
значения строк, смещение, 396, 399
наибольшее/наименьшее значения, поиск, 391
прогнозы, простые, формирование, 411, 419
результаты, разбиение на страницы, 378, 381
результаты, ранжирование, 400
строки со взаимообратными значениями, выявление, 387, 389
строки таблицы, как пропустить, 381, 384

строки, сбор информации из последующих, 393, 396
текста, не соответствующего шаблону (Oracle), 581, 583
ход конем, поиск значений, 403, 410
предельные значения, поиск, 391
представления словаря данных (Oracle), 132, 133
проверка существования значения в группе, 585, 588
прогнозы, простые, формирование, 411, 419
промежуточные произведения, 209, 212
промежуточные разности, 212
псевдонимы
вложенные представления, 27, 568
для выражения CASE, 39
обращение к столбцам по псевдонимам, 37
порядок выполнения приложения, 37

Р

разбиение на страницы
см. поиск, 378
разворачивание
в несколько строк, результаты, 423, 430
в одну строку, результаты, 420
вычисления, в которых участвуют данные разных строк, 440
обратное, 431, 435
оператор MODEL (Oracle), 536
описание, 420
подсуммы, результирующее множество их содержащее, 495, 499
ранжированные результирующие множества, 550, 555
разность множеств, 68
разрешения на использование, 27
ранжирующая функция, применение, 315
Рассел, Бертран, 593
регулярные выражения (Oracle), 581, 583
результаты запроса, изменение имен
см. псевдонимы, 36
результирующее множество, транспонирование (Oracle), 536, 540
реляционное деление, 647, 654
рефлексивные объединения
альтернативы, 345, 347, 349, 352
применение, 344, 388
рецепты, 16
переносимость, 19
разрешения на использование, 27
решения, 16
для конкретных баз данных, 19
решения Розенштейна
запросы «любой» или «все», 651, 652
запросы «не более», 629, 632, 634
запросы «не менее», 636, 638, 640
запросы по отрицанию
A или B, но не оба, 624
не A, 622
только A, 628
запросы «точно», 641
запросы «точно равно», 644, 646
Розенштейн Дэвид, 23, 617

С

связанные подзапросы, 71
сегменты, 607
и оператор ORDER BY, 607
сериализованные данные, синтаксический разбор в строки таблицы, 569, 573
символ * в выражениях SELECT, 33
скалярные подзапросы
и объединения, 76
использование, 207, 208, 210
использование в операторе WHERE, 37
использование для поиска разности между значениями строк, 354
преобразование в составные (Oracle), 566, 568
слияние записей, 115
см. записи
слияние, 113
словари данных
см. метаданные, 121
случайные записи, извлечение, 42, 43
соглашения
по написанию кода, 26
типографские, 26
сортировка записей, 47, 60
и неопределенные значения, 53, 60
по зависящему от данных ключу, 60
по нескольким полям, 48
по одному полю, 47
по подстрокам, 49
смешанные буквенно-цифровые данные, 50, 53
строки, 154, 155, 161

составление отчетов
см. извлечение записей, сложные вопросы, 420
составные подзапросы, преобразование скалярных подзапросов в (Oracle), 566, 568
списки
значений, разделенных запятыми, создание (Oracle), 575, 581
оператора IN, преобразование данных с разделителями, 168, 174
с разделителями, создание, 161, 168
средние, вычисление, 197
стандарт ANSI, 20
столбцы
добавление заголовков в дважды развернутые результирующие множества, 555, 565
конкатенация, 38
преобразование в строки
см. разворачивание, 433
Столл, Роберт, 593
строки
в общем, 134
запросы, 134, 196
буквенно-цифровой статус, определение, 145, 150
буквенно-цифровые, сортировка смешанных данных, 50
извлечение элементов, 541, 543
инициалы, извлечение из имени, 150, 154
кавычки, как вставить, 137
обход, 134, 137
подстроки, извлечение, 187, 193
подсчет количества экземпляров символа, 138
поиск смешанных буквенно-цифровых, 545, 547
разделение числовых и символьных данных, 141, 145
расположение в алфавитном порядке, 174, 180
символы, удаление ненужных, 139
синтаксический разбор IP-адреса, 194, 196
синтаксический разбор в строки таблицы, 569, 573
списки оператора IN, преобразование данных, 168, 174
списки с разделителями, создание, 161, 168
упорядочивание по части, 154
упорядочивание по числу, 155, 161
числовое содержимое, выявление, 180, 187
преобразование в столбцы
см. разворачивание, 431
со взаимообратными значениями, выявление, 387, 389
суммирование значений столбцов, 201, 241
схема аксиом выделения, 594
схемы
см. метаданные, 121
сценарии, генерирование, 130

T

таблиц несколько, 62
вставка данных, 104, 106
извлечение данных, 98
взаимосвязанные строки, объединение, 64
внешние объединения при использовании агрегатов, 90, 93
добавление объединений к существующим объединениям, 74
значения NULL в операциях/сравнениях, 97
значения, которых нет в других таблицах, 68
и декартовы произведения, 83
несовпадающие строки, 72
объединения при использовании агрегатов, 85, 89
отсутствующие данные из нескольких таблиц, возвращение, 93, 97
поиск общих строк, 66
сравнение, 76, 83
таблицы
описание, 100
пример, 24
создание и копирование описания, 103
структура примера, 24
таблицы сводные, пример, 25
транспонирование результирующих множеств (Oracle), 536, 540

У

уничтожение записей
  всех, 115
  дублирующихся, 117
  которые нарушают целостность, 117
  на которые есть ссылки в другой таблице, 119
  одной, 116
  определенных, 116
уничтожение строк
  содержащих значения NULL (DB2/Oracle/SQL Server), 426
  содержащих значения NULL (PostgreSQL/MySQL), 429, 430
условная логика в выражениях SELECT, 39

Ф

формирование строк, автоматическое, 373, 411, 413
Фреге, Готтлоб, 595

Х

ход конем, поиск значений, 403, 410

Ц

целостность ссылочных данных, уничтожение нарушающих ее записей, 117
Цермело, Эрнст, 594
циклы, отсутствие в SQL, 134

Ш

шаблоны
  поиск несоответствующего текста (Oracle), 581, 583
  поиск соответствий, 45, 46

Э

эквиобъединения операции, 64, 72

Я

Янг, Кей, 404

По договору между издательством «Символ-Плюс» и Интернет-магазином «Books.Ru – Книги России» единственный легальный способ получения данного файла с книгой ISBN 5-93286-125-8, название «SQL. Сборник рецептов» – покупка в Интернет-магазине «Books.Ru – Книги России». Если Вы получили данный файл каким-либо другим образом, Вы нарушили международное законодательство и законодательство Российской Федерации об охране авторского права. Вам необходимо удалить данный файл, а также сообщить издательству «Символ-Плюс» (piracy@symbol.ru), где именно Вы получили данный файл.