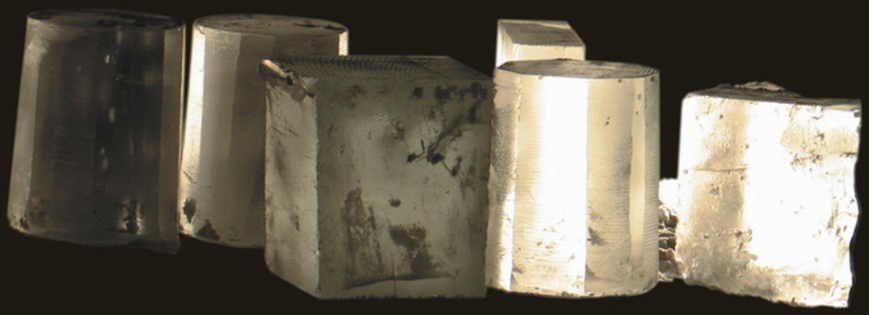
АЛЕКСАНДР БОНДАРЬ

InterBase и Firebird

практическое руководство для умных пользователей и начинающих разработчиков

bhv



Подробное описание языков DDL и DML серверов InterBase и Firebird

Практические рекомендации по созданию структур баз данных

Детальное рассмотрение аспектов манипулирования данными и управления транзакциями

Реальные примеры клиентских приложений на Delphi с использованием FIBPlus и IBX

+ CD

Александр Бондарь

InterBase и Firebird

практическое руководство для умных пользователей и начинающих разработчиков

Санкт-Петербург
«БХВ-Петербург»
2007

УДК 681.3.06
ББК 32.973.26-018.2
Б81

Бондарь А. Г.

Б81 InterBase и Firebird. Практическое руководство для умных пользователей и начинающих разработчиков. — СПб.: БХВ-Петербург, 2007. — 592 с.: ил. + CD-ROM

ISBN 978-5-9775-0098-2

Рассматриваются возможности серверов баз данных InterBase и Firebird. Описываются объекты базы данных (таблицы, домены, индексы, представления и т. д.), синтаксис и семантика операторов SQL, используемых для работы с метаданными и с данными базы данных. Рассматриваются вопросы проектирования реляционных баз данных для решения реальных задач предметной области, на которой проводятся всевозможные исследования. Приводится множество примеров использования операторов манипулирования данными. Детально описаны транзакции, взаимодействие параллельных процессов с различными характеристиками и уровнями изоляции транзакций. Создается учебная база данных и множество программ в среде Delphi, иллюстрирующих возможности серверов базы данных. Компакт-диск содержит примеры из книги, а также программное обеспечение для разработчиков БД: InterBase 2007 Developer Edition, Firebird 2.0 и др.

Для разработчиков баз данных и программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Корректор	*Зинаида Дмитриева*
Дизайн обложки	*Инны Тачиной*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.04.07.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 47,73.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 194354, Санкт-Петербург, ул. Есенина, 5Б.

Санитарно-эпидемиологическое заключение на продукцию
№ 77.99.02.953.Д.006421.11.04 от 11.11.2004 г. выдано Федеральной службой
по надзору в сфере защиты прав потребителей и благополучия человека.

Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 978-5-9775-0098-2

Оглавление

Эту книгу я посвящаю
памяти моего очень близкого друга
Владимира Васильевича Рыбина.
Мне страшно жаль, Володя,
что ты не можешь прочесть эти строки.

Введение

Говорят, что мало кто читает введения, предисловия. Если это действительно так, то не стану здесь много рассказывать, какие же замечательные системы управления базами данных InterBase и Firebird. Поскольку вы держите в руках эту книгу, то наверняка знаете, что это компактные и мощные серверы баз данных. К тому же Firebird — абсолютно бесплатная система, а версия InterBase 2007 Developer Edition — бесплатная для разработчиков.

Для кого эта книга?

У нас в стране очень много книг серии "для чайников". В оригинале, правда, это звучит несколько иначе — "for dummies", что все же более правильно следует перевести как "для придурков". Появилась еще одна серия, в названии которой присутствуют слова complete, idiot's, guide. В зависимости от допустимой перестановки слов в названии этой серии может получиться либо "полное руководство", либо "для полных идиотов"[1].

Я же считаю, что человек, ринувшийся заниматься программированием, не может быть ни dummy, ни тем более idiot. По этой причине в название этой книги я поместил слово "умный".

Основная задача книги — дать возможность "потрогать руками" различные средства этих серверов баз данных по принципу "делай раз, делай два". Выполняя предложенные действия, вы действительно освоите множество полезных вещей в работе с базами данных. Я, по крайней мере, освоил многое. В том числе и благодаря написанию этой книги.

Чтение книги не требует предварительных знаний в области реляционных баз данных. Откровенно говоря, это не совсем так. Человек, вообще не

[1] Я настоятельно рекомендую некоторым моим студентам тщательно изучать книги именно этих серий.

знакомый с программированием, его основными принципами, вряд ли получит настоящее удовольствие от использования данной книги. Конечно же, для подобной деятельности нужны определенные знания, некая "программерская" культура.

При этом книга, конечно же, в первую очередь предназначена для начинающих использовать эти замечательные серверы баз данных, хотя — бьюсь об заклад — мало кто из корифеев InterBase и Firebird так же тщательно выполнял все те работы, которые описаны в книге. Уверен, что книга будет полезна и тем, кто много и плодотворно работает с этими серверами.

Основную целевую аудиторию этого издания я бы разделил на три категории.

- Люди, желающие естественным, структурированным, путем получить необходимые практические знания о системах управления базами данных вообще и о системах InterBase и Firebird в частности. Их будет в первую очередь интересовать язык SQL, его возможности, операторы, а также существующие программные средства для выполнения запросов SQL. Написание собственных программ работы с базами данных их, похоже, не очень в настоящий момент интересует. Им следует пока смело пролистывать страницы, посвященные разработке программ, и внимательно изучать материал, посвященный серверам баз данных и конструкциям языка SQL. К написанию программ можно приступить и потом. Уверен, что этим они со временем обязательно займутся.
- Те, кого интересует именно разработка программ. Про SQL они слышали, а может быть, и довольно активно использовали его возможности. Они могут лишь бегло просматривать все описания, относящиеся к реляционным базам данных, операторам SQL, и выполнять подробно описанные действия по созданию собственных программ. Со временем они могут вернуться к начальным главам и более глубоко освоить действия с базой данных с помощью средств SQL, тем более что эти действия того стоят.
- Люди, которые, доверившись автору, выполняют все предлагаемые в книге действия — используют утилиты командной строки и программу графического интерфейса, выполняя операторы SQL для получения желаемого результата, запускают среду Delphi, пишут программы, предлагаемые автором, находят более элегантные решения, чем те, которые описаны в книге, получают от этого большое удовольствие и намекают автору, что следовало бы быть повнимательнее при выборе вариантов решения задач. Наверное, нет необходимости говорить, что такая категория людей всегда добьется в жизни много больше, чем другие.

Вообще говоря, существует еще одна категория читателей. Это люди, которые просто из любопытства начинают просматривать книгу, потом вдруг по-

нимают, что реляционные базы — это действительно очень интересно, и со временем становятся крупными специалистами в этой области деятельности. Удачи!

Содержание книги

В *главе 1 "Реляционные базы данных InterBase и Firebird"* очень кратко рассказывается о работе с внешними данными, с базами данных. Рассматриваются объекты реляционных баз данных. Разбираются отношения в базах данных и их реализация применительно к реляционным базам данных. Здесь же мы познакомимся с такой важной деятельностью, как нормализация таблиц, выполним небольшую работу по созданию нескольких простых таблиц. В конце главы рассказывается о средствах описания синтаксиса формальных языков, дается формальное описание некоторых базовых конструкций языка SQL.

Если вам захочется поподробнее узнать об истории, принципах и — страшно сказать — математических основах реляционных баз данных, вы всегда сможете найти подходящую литературу.

Сама наша работа с базами данных начинается в *главе 2 "Создание базы данных"*. Мы начнем с создания и изменения учетных записей пользователя. После этого различными способами будем создавать нашу учебную базу данных, с которой будем работать на протяжении всей этой книги. Научимся регистрировать базу данных в программе IBExpert. Освоим различные средства копирования и восстановления баз данных. Там же мы создадим собственные программы для работы с учетными записями пользователя, для создания базы данных и для выполнения копирования и восстановления баз данных.

Затем в *главе 3 "Работа с доменами"* рассмотрим домены, типы данных, используемые в серверах InterBase и Firebird. Проведем множество экспериментов с различными типами данных. Научимся создавать, изменять и удалять домены. Наконец, создадим несколько доменов, которые будем использовать при описании таблиц нашей базы данных.

Достаточно объемная и важная *глава 4 "Работа с таблицами"*. В ней рассматривается основной и самый сложный объект реляционной базы данных — таблицы. Даются рекомендации по выбору первичного ключа. Рассматривается весьма сложный синтаксис оператора создания таблиц. Проектируются и создаются все таблицы, используемые в нашей учебной базе данных. Кратко рассматриваются индексы. Предлагается для решения несколько задач по проектированию и созданию таблиц. Там же я даю и свои варианты решений этих задач. Ваши могут быть лучше.

В *главе 5 "Добавление, изменение, удаление и выборка данных"* подробно рассматриваются операторы, выполняющие соответствующие действия. Не-

сколько предварительных слов сказано об операторе выборки данных. Здесь мы добавляем и изменяем данные нашей базы данных.

Глава 6 называется *"Для особо одаренных. Написание программ работы с базой данных"*. Здесь мы с вами сделаем небольшой перерыв в рассмотрении средств и языковых конструкций систем управления базами данных и разом напишем множество программ для решения различных полезных задач работы с базами данных.

Самый сложный и интересный оператор — оператор `SELECT`, позволяющий осуществлять выборку данных из базы. Ему полностью посвящена весьма объемная *глава 7 "Поиск данных"*. Мы достаточно подробно рассмотрим богатый синтаксис и необыкновенные по своей мощности возможности оператора `SELECT`, выполним много различных операций поиска данных в нашей учебной базе данных. Будем использовать в операторе различные функции, выполним операции соединения (`JOIN`). Там же исследуются представления (view), создается некоторое количество довольно простых (и не очень) полезных представлений.

В *главе 8 "Транзакции"* рассматривается такой замечательный и на самом деле не слишком сложный механизм, как транзакции — различные их характеристики и варианты использования. Мы напишем тестовую программу, или несколько программ, которые позволят нам детально исследовать отдельные характеристики транзакций и их влияние на параллельные процессы. Если у вас есть возможность работать в локальной сети, вы сможете в полном объеме оценить достоинства транзакций.

В *главе 9 "Хранимые процедуры и триггеры"* мы рассмотрим процедурное расширение языка SQL, используемое для написания хранимых процедур и триггеров. Вы сами или с моей помощью напишете несколько полезных триггеров и хранимых процедур. Исследование возможностей этих объектов базы данных займет достаточно много времени, и это время не будет потрачено впустую.

Завершает книгу *глава 10 "Для особо одаренных. Полезные программы работы с базой данных"*, где мы с вами выполним совместную разработку некоторого количества довольно сложных и, главное, интересных программ. Для их создания мы будем использовать все полученные в процессе работы над книгой знания и умения. Поверьте, результат будет замечательным.

В приложениях содержатся дополнительные полезные данные.

В *приложении 1* приведено подробное описание инсталляции всех программ и компонентов, которые вам понадобятся для выполнения заданий этой книги и вообще в вашей профессиональной деятельности за исключением Delphi.

Приложение 2 содержит полное описание наборов символов и порядка сортировки, используемых в системах InterBase и Firebird. Поясняется структура

соответствующих системных таблиц. Мы с вами напишем программы, отображающие состав наборов символов и порядка сортировки, фактически присутствующие в вашей реальной базе данных.

Приложение 3 содержит описание кодов ошибок, с которыми вы можете столкнуться в вашей работе.

В *приложении 4* приведена структура некоторых системных таблиц. Создаются программы для отображения в удобном виде данных из этих таблиц.

Приложение 5 описывает содержимое сайта, где расположены необходимые материалы для работы с книгой.

Многие ваши работы при использовании этой книги выполняются в три этапа.

1. Сначала выполняются необходимые действия с использованием утилит командной строки, входящих в комплект поставки серверов базы данных. В любом случае это следует проделать на начальных этапах, чтобы оценить все удобства (или неудобства) таких утилит. Потом использование командной строки вам, конечно, надоест, и вы перейдете к работе только с программой графического интерфейса, тем более что при попытках отобразить русский текст из строковых полей базы данных в командной строке вы получите не совсем то, что ожидали.
2. Затем те же действия выполняются с использованием программы графического интерфейса. Здесь и в повседневной программистской жизни я использую IBExpert. Практически с тем же успехом можно использовать и любую другую программу, например, BlazeTop.
3. Третий этап (главы и разделы начинаются со слов "для особо одаренных") дает возможность написать собственную программу, выполняющую те же самые действия. Для этого нужна среда Delphi версии не ниже 7 и компоненты FIBPlus или IBX для работы с базами данных.

Разработка программ описывается очень подробно, временами даже слишком подробно. В первую очередь это касается разработки начальных программ. У некоторых очень внимательных читателей и продвинутых программистов это может вызвать раздражение, однако менее внимательные и начинающие программисты будут мне благодарны за многочисленные повторы.

Примерно 80% материала книги относится к основным сведениям по серверам баз данных и языковым средствам, остальная часть связана с написанием качественных программ по работе с базами данных.

Если же вам не захочется сразу же проявить свою одаренность, вы можете отложить написание программ на более позднее время или же вовсе скрыть

от окружающих и от себя самого (самой) свои необыкновенные способности по написанию полезных программ.

Примеры не содержат ошибок. Последнее время это стало весьма распространенной программистской шуткой. Однако должен сказать, что *все* примеры и программы проверены мною в Firebird 1.5.3, Firebird 2.0 и InterBase 2007 Developer Edition на локальном компьютере в операционной системе Windows XP и в локальной сети, включающей 15 компьютеров, под управлением Windows 2003.

Данная книга, с моей точки зрения, содержит большое количество достоинств. Вот лишь некоторые из них.

- **Практический подход к освоению систем управления базами данных.** Я стараюсь избегать специфической терминологии, наукообразия в изложении материала. Это дает возможность, не вдаваясь в философские и математические детали, лежащие в основе реляционных баз данных, научиться за короткий промежуток времени использовать описываемые СУБД в своей практической деятельности. Такой подход, я надеюсь, позволит вам создавать качественные базы данных и столь же хорошие программы работы с этими базами данных.
- **Последовательность в изложении материала.** Это не справочник и не избранные темы относительно реляционных баз данных вообще и конкретных СУБД в частности. Последовательно изучая материал книги и выполняя предложенные действия, вы, в конце концов, приобретете глубокие и твердые знания, научитесь на практике применять эти знания.
- **Доступность изложения.** Книга написана нормальным человеческим языком для нормальных людей. Любой желающий при некоторых интеллектуальных усилиях освоит предлагаемый материал.

При этом книга, разумеется, не свободна и от некоторых мелких недостатков. Собственно говоря, их ровно три.

- **Практический подход к освоению систем управления базами данных.** Это не позволяет читателю получить глубокие знания в области реляционной алгебры, истории возникновения и развития реляционных баз данных.
- **Последовательность в изложении материала.** Это не дает возможности читателю рассматривать данную книгу только как справочник, позволяющий быстро найти ответ на интересующий его частный вопрос.
- **Доступность изложения.** Материал в книге представлен очень свободно, без надлежащей математической строгости. По прочтении такой книги нет причин невзначай сообщить знакомым, что вот, мол, освоил я таки столь сложную книгу.

Если вас устраивают достоинства и не слишком огорчают недостатки — приступайте к этой интереснейшей деятельности!

Надеюсь, вы понимаете, что освоение нового материала или упорядочение знаний по уже известной тематике всегда требует определенных усилий, временами довольно больших. Мне нравится сравнение такой деятельности с прохождением горных маршрутов. Есть периоды, когда нужно напрячь все силы и подниматься по камням, по снегу, по фирну все выше и выше к перевалу. Сердце сильно бьется в груди. Воздух разряжен, дышать трудно. Каждый шаг дается с трудом. Такое напряжение всегда вознаграждается, когда в состоянии необыкновенного восторга ты стоишь на высокой точке своего маршрута, понимая, какой сложный путь ты преодолел. После этого идет плавный быстрый спуск. Вот ты уже на равнине, ласково светит солнце, кругом зеленая травка, цветочки, птички поют, пчелки летают. И ты готовишься к выходу на следующий перевал, прекрасно понимая, что там будет еще сложнее, еще лучше, еще интереснее...

Я с трудом скрываю свою зависть, зависть белую, к тем людям, которые впервые пойдут по этому маршруту.

Где найти нужные сведения и программные средства?

Скрипты, используемые для создания базы данных, всех ее объектов, помещения в таблицы данных, а также тексты программ находятся на сайте издательства по адресу: ftp://ftp.bhv.ru/9785977500982.zip.

Знающим английский язык (*настоящий* программист должен знать этот язык) можно порекомендовать использовать полный комплект документации по InterBase 2007. Эта документация станет доступной только после инсталляции сервера базы данных. С Firebird 2.0 вы получаете три документа в формате PDF. Если вы хотите серьезно заниматься этими базами данных, имеет смысл изучить или хотя бы просмотреть все перечисленные документы. На русском языке существует объемная книга Хелен Борри "Firebird: руководство разработчика баз данных", а также книга А. Ковязина и С. Вострикова "Мир InterBase". Книгу Хелен Борри можно было бы назвать энциклопедией Firebird. Вообще по реляционным базам данных существует море литературы как на английском, так и на русском языках. Порекомендовать какую-либо одну из них я не решусь.

Если вы хотите выполнять предлагаемые примеры, вам, по меньшей мере, понадобится сервер базы данных не менее InterBase 6 — это бесплатная версия, правда, содержащая большое количество ошибок.

На сайтах **sourceforge.net** и **firebirdsql.org** можно найти бесплатный Firebird последних версий. Самые последние сведения об InterBase (русскоязычный вариант сайта) находятся на **embarcadero.com/ru/**.

Я также использую программу графического интерфейса для работы с базами данных — IBExpert. Получить бесплатную для стран бывшего СССР (граждане которых не делают вид, что не знают русского языка) новую версию можно на сайте **www.ibexpert.com**.

Бесплатную пробную, "триальную" версию компонентов работы с базами данных FIBPlus вы найдете на сайте **devrace.com**.

Компоненты FastReports на сайте **fast-report.com/**.

Компоненты RichView находятся на сайте **trichview.com/**.

Для выполнения всех работ, описанных в этой книге, вам понадобится еще и среда разработки Delphi версии не менее 7. Это платная система и по нашим меркам довольно дорогая, но она того стоит.

Благодарности

Это, пожалуй, самая приятная часть введения. Автор испытывает чувство признательности к очень многим людям и организациям, благодаря которым оказалась написанной эта книга.

В первую очередь мне хотелось бы поблагодарить ребят (и девчат), которые создавали для нас с вами такие замечательные СУБД, как InterBase и Firebird.

Спасибо ребятам, разработавшим очень удобную программу работы с этими СУБД — IBExpert. Именно ее я использую в своей повседневной работе. И другим советую.

Большая благодарность разработчику компонентов FIBPlus для работы с базами данных, созданных в InterBase и Firebird, Сергею Бузаджи. Спасибо тебе, Сергей, за прекрасную разработку и постоянное совершенствование этого, казалось бы и без того совершенного продукта. Спасибо тебе также и за твое долготерпение и подробные объяснения в то время, когда я заваливал тебя различными вопросами (признаюсь — не всегда самыми умными) по тонкостям использования этих компонентов.

Спасибо Сергею Ткаченко за довольно сложные, но очень мощные полнофункциональные компоненты RichView для работы с текстами в обогащенном формате RTF. Эти компоненты называются автором средствами для отображения, редактирования и печати гипертекста.

Спасибо компании FastReports во главе с Михаилом Филиппенко за компоненты FastReport, очень удобные и красивые, позволяющие довольно легко создавать практически любые отчеты.

Огромная благодарность Сергею Вострикову, научному редактору этой книги, который внимательно перечитывал постоянно меняющиеся главы и давал ценные советы. Спасибо, Сережа, а те грубые слова, которые ты мне говорил по поводу моей привычки обрабатывать ошибочные ситуации базы данных, я уже почти забыл.

Хочу поблагодарить коллектив издательства "БХВ-Петербург" за их высокий уровень профессионализма. По своему предыдущему опыту сотрудничества с этим издательством могу отметить действительно профессиональную работу, в первую очередь, корректоров и редакторов.

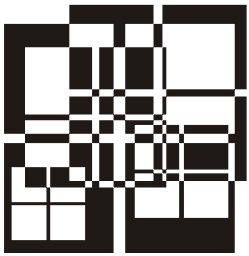
И, наконец, самая большая благодарность всем моим студентам. За их поддержку, за их попытки изобразить полное непонимание отдельных вопросов работы с базами данных. В результате книга стала еще более доступной для большинства людей, проявляющих в той или иной мере интерес к реляционным базам данных. Я очень надеюсь, что читателя не будут так уж сильно раздражать многочисленные повторы, встречающиеся в книге.

Что там за перевалом?

В следующей главе мы начнем рассмотрение основ реляционных баз данных вообще и СУБД InterBase и Firebird в частности.

Мы рассмотрим объекты базы данных, реализацию отношений в реляционных базах данных, способы и практическую необходимость нормализации таблиц, а также основные синтаксические конструкции языка SQL.

ГЛАВА 1



Реляционные базы данных InterBase и Firebird

За сравнительно короткий промежуток времени идеи и технологии использования баз данных прошли большой и, надо сказать, впечатляющий путь.

Сначала все обрабатываемые данные носили исключительно последовательный характер. Это были колоды перфокарт, печатающие устройства. Позже появились магнитные ленты.

Технические предпосылки сложной организации данных возникли только после появления внешних запоминающих устройств прямого доступа — магнитных барабанов, пакетов магнитных дисков. В это время и появились первые идеи создания баз данных и систем управления базами данных.

В настоящее время большинство прикладных программ связано с обработкой большого объема данных, имеющих сложную структуру. Примерами таких программ являются поисковые системы (библиографические и так называемые "базы знаний"), программные системы организационного управления (обработка данных для управления предприятиями, для решения учетных задач, например, существует одна из наиболее емких в плане количества обрабатываемых данных подсистем — бухгалтерский учет), системы управления космическими объектами и др. Для хранения данных при решении подобных задач используются базы данных.

База данных — это один или несколько связанных между собой файлов, в которых хранятся все обрабатываемые данные, а также описания этих данных — метаданные. Программные средства работы с данными в базе данных называются системой управления базой данных (сокращенно СУБД) или иногда, не слишком удачно, сервером базы данных. Мы будем использовать оба этих термина. Есть еще один термин, связанный с такими программами.

Ядро СУБД называют engine, дословно двигатель, движок. Языковые средства, используемые в базах данных, дают возможность, во-первых, описывать данные в базе данных, а во-вторых, осуществлять манипулирование данными — выполнять добавление, изменение, удаление и выборку данных.

Понятно, что сама по себе база данных просто так никому не нужна. Она используется вместе с соответствующими программами, обрабатывающими эти данные. Если подняться с программистского уровня рассмотрения баз данных на общечеловеческий, то можно сказать, что данные в базе данных описывают состояние какой-то конкретной предметной области человеческой деятельности, а пользовательские, клиентские программы решают задачи из этой предметной области.

Одним из важных принципов баз данных является то, что описание данных (метаданные) хранится также в самой базе данных. Это позволяет уменьшить зависимость программ от структуры хранимых данных.

В *реляционных базах данных* все обрабатываемые данные хранятся в виде однородных таблиц. Эти таблицы часто называют плоскими, двумерными, поскольку все строки одной таблицы имеют одинаковую структуру. Программные средства работы с реляционными базами данных называются реляционной СУБД. В качестве языковых средств используется SQL — Structured Query Language, язык структурированных запросов. Этот язык включает в себя две части — язык описания данных (DDL, Data Definition Language) и язык манипулирования данными (DML, Data Manipulation Language).

Несмотря на то, что существуют международные стандарты SQL, реальные системы управления базами данных временами довольно сильно отличаются друг от друга как по языковым средствам, так и по выполняемым функциям. Мы будем рассматривать семейство серверов баз данных InterBase и Firebird. Они очень похожи. Это компактные и довольно мощные серверы баз данных. Большинство описываемых здесь возможностей существует в InterBase, начиная с версии 6.0, и в Firebird версии 1.5 и выше.

При написании программ работы с базами данных, поддерживаемыми этими СУБД, а также при выполнении таких программ, нам практически все равно, где находится сама база данных — на нашем, локальном, компьютере или на сервере в локальной сети. Иными словами, поддержка архитектуры "клиент-сервер" в этих СУБД осуществляется легко и непринужденно. От нас почти не требуется никаких дополнительных затрат сил и времени для того, чтобы наши программы работали как на локальном компьютере, так и в архитектуре "клиент-сервер", когда с одной базой данных одновременно работает несколько пользовательских программ. Единственная задача — так организовать использование транзакций и их характеристик, чтобы минимизировать блокировки при одновременной работе многих пользователей с одной базой данных. Транзакциям мы посвятим отдельную главу.

В реляционных базах данных существует своя система строгих понятий и принципов, которые и делают подобные СУБД очень удобными в работе.

Давайте договоримся о самой начальной терминологии — в базах данных хранятся, а программами обрабатываются только *данные*. Термин же *информация*, и в плане хранения, и в плане обработки, я предпочитаю применять только к человеку.

1.1. Основные объекты и понятия реляционной базы данных

Рассмотрим некоторые понятия и объекты реляционной базы данных, а также отдельные их характеристики. В соответствующих главах мы рассмотрим эти объекты подробнее.

Все данные в реляционной базе данных хранятся в виде таблиц — плоских двумерных структур. В реляционной базе данных хранятся собственно данные, организованные в виде таблиц, и *метаданные* — описания этих данных. Метаданные хранятся в *системных таблицах*.

Каждая *таблица (table)* содержит произвольное количество *строк (row)*. Другие названия для строки: запись (record), реже — кортеж. В русскоязычной литературе практически одинаково часто используются термины "строка" и "запись", равно как и в англоязычной — аналогичные английские варианты. Все строки одной таблицы имеют одинаковую структуру. Они состоят из одного или нескольких *столбцов (column)*, иногда называемых полями (field), элементами данных (data element), колонками или атрибутами (attribute). У нас обычно используются термины "столбец" и "поле". Каждый столбец имеет конкретный тип данных.

Таблицу любители реляционной алгебры часто называют отношением. С математической точки зрения это правильно, однако у большинства программистов такой термин вызывает негативную реакцию.

Тип данных (datatype) — важнейшая характеристика столбца, определяющая множество возможных значений, которые может принимать данный столбец, и операции, допустимые для данного столбца. Например, тип данных `INTEGER` позволяет хранить в столбце целые числа в определенном диапазоне; для такого столбца можно использовать четыре арифметические операции — сложение, вычитание, умножение и деление. Для строковых типов данных множеством допустимых значений являются строки, содержащие произвольные символы. К ним применяется одна операция конкатенации — соединения двух строк в одну. У логического типа данных множество допустимых значений состоит из двух элементов, истина (`TRUE`) и ложь (`FALSE`). В реляционных базах данных для логического типа данных используется еще и неопределен-

ное, или неизвестное, (NULL) значение. Множеством допустимых операций являются, как минимум, отрицание, дизъюнкция (логическое ИЛИ) и конъюнкция (логическое И). Существуют также типы данных даты, времени, чисел с плавающей точкой и др.

Домен (domain) — в математической терминологии это множество допустимых значений и множество допустимых операций для элемента данных. В реляционных базах данных существует объект домен, который описывает некоторые характеристики элемента данных (как минимум, тип данных) и на который можно ссылаться при описании столбцов таблицы.

Пустое значение (NULL) — в реляционных базах данных столбец и локальная переменная в хранимых процедурах и триггерах помимо "нормального" значения может иметь также пустое, или неизвестное значение. Интересным и даже на первый взгляд несколько странным свойством этого значения является то, что два поля, имеющие своим значением NULL, не равны друг другу. Любая операция сравнения, в которой принимают участие такие поля, не даст значений ни TRUE (истина), ни FALSE (ложь). Результатом сравнения в этом случае будет NULL (неизвестный результат). До сих пор еще ведутся споры относительно необходимости использования и правильности интерпретации пустого значения. Мы же с вами не станем тратить на это время и примем сложившуюся практику так, как она есть. Тем более что способы работы с пустыми значениями хорошо отработаны. При этом только нужно постоянно помнить, что прежде чем сравнивать соответствующие объекты базы данных, для которых допустимо пустое значение, требуется обязательная проверка их на значение NULL. Лично я об этом иногда забываю.

Индекс (index). Для таблицы может быть задано любое количество индексов[2]. Индекс состоит из одного или нескольких столбцов таблицы. В базе данных он представлен упорядоченным списком значений и адресов. В каждом элементе списка находится адрес строки таблицы, содержащей это значение. По этому адресу нужная строка быстро отыскивается на внешнем носителе. Индексы используются для ускорения выборки данных по запросу и упорядочения результатов запроса, а также иногда для обеспечения уникальности значений (unique index — уникальный индекс). В отличие от простых "настольных" (desktop) систем управления базами данных, где без индексов просто нельзя было обойтись, в реляционных СУБД индексы не играют столь боль-

[2] Вообще-то ограничение существует. Для одной таблицы может быть создано не более 65 536 индексов. Я плохо представляю себе таблицу, для которой кому-то захочется создавать такое количество индексов. Мне также трудно представить и человека, который решит создавать столько индексов. Лучше заняться чем-нибудь действительно полезным, например, написанием книг по базам данных.

шой роли. Здесь основное их назначение — повышение производительности работы системы.

В создаваемой нами учебной базе данных мы не станем использовать индексы.

Первичный ключ (`PRIMARY KEY`) — столбец или несколько столбцов таблицы, значение которых однозначно определяет конкретную строку таблицы. Основным требованием к первичному ключу является его уникальность — в таблице не может быть двух разных строк, имеющих одинаковое значение первичного ключа. Ни один столбец, входящий в состав первичного ключа, не может иметь пустого значения (`NULL`). Таблица может иметь только один первичный ключ. Первичный ключ — один из важнейших элементов в системе реляционных баз данных. Помимо однозначной идентификации конкретной строки в таблице он позволяет в связке с внешними ключами реализовать все возможные отношения между данными в базе данных. Для каждого первичного ключа система автоматически создает индекс, позволяющий ускорить выборку необходимых записей. Ни в коем случае нельзя явно создавать для таблицы индекс, соответствующий по структуре первичному ключу.

Часто бывает сложно выбрать существующие столбцы таблицы в качестве первичного ключа. В таких случаях используется искусственный первичный ключ. В таблицу добавляется числовой столбец, он будет использован в качестве первичного ключа. Значение для него выбирается из генератора (см. далее). Более подробно о первичных ключах мы поговорим позже.

Уникальный (`UNIQUE`) ключ. Во многом схож с первичным ключом. Он также однозначно определяет строку таблицы. Может принимать участие в реализации отношений в базе данных — на него могут ссылаться внешние ключи других таблиц или той же самой таблицы. Кроме того, в таблице может существовать несколько уникальных ключей. Для уникального ключа системой также автоматически создается индекс. Вы не должны явно создавать индекс, имеющий аналогичную структуру. Лично в моей практике работы с базами данных я не припомню случая, когда бы мне приходилось использовать уникальные ключи. У меня есть смутные подозрения, что наличие в таблице уникальных ключей, которые к тому же используются в связке с внешними ключами, говорит, скорее всего, об ошибках при проектировании базы данных. Впрочем, я могу ошибаться.

Внешний ключ (`FOREIGN KEY`) — столбец или несколько столбцов таблицы (дочерней таблицы), которые ссылаются на первичный или уникальный ключ другой или той же самой таблицы (родительской таблицы). Либо внешний ключ весь должен быть пустым (иметь значения `NULL` для всех столбцов, входящих в состав ключа), либо в таблице, на которую ссылается внешний ключ, должна существовать строка с соответствующим значением первичного (или

уникального) ключа. Понятно, что количество столбцов и их типы данных для внешнего ключа и первичного (уникального) ключа, на который ссылается внешний ключ, должны совпадать. Для внешнего ключа система также автоматически создает индекс. Индекс такой же структуры для таблицы вы создавать не должны.

Связка "внешний ключ/первичный (уникальный) ключ" используется для поддержания *ссылочной целостности* (referential integrity) базы данных и для реализации отношений между строками различных таблиц.

Здесь следует отдавать себе отчет, что никаких *структурных* связей между внешними ключами дочерней таблицы и соответствующими им первичными ключами родительской таблицы не существует. Внешний ключ никоим образом не ссылается на первичный с помощью, например, каких-то указателей, как подобные вещи были реализованы в предыдущих поколениях СУБД. Просто существуют индексы и некоторые программные средства (автоматически создаваемые триггеры — см. далее), которые позволяют поддерживать нужные соответствия внешних ключей первичным. Эти вопросы мы рассмотрим позже довольно подробно.

Сама идея реализации отношений таким простым способом изумляет, во-первых, именно своей простотой и, во-вторых, необыкновенной гибкостью, т. е. возможностью добавлять в существующую базу данных новые отношения, практически ничего не меняя в существующих данных.

Генератор (generator) — наиболее простой объект базы данных, используемый для генерации уникального числового значения. Для получения очередного значения из генератора используется функция `GEN_ID`. При первом обращении к функции возвращается единица, при последующих обращениях возвращается значение, обычно на единицу больше предыдущего значения. Используется для получения уникальных значений искусственного первичного ключа. Об использовании искусственных первичных ключей мы поговорим более подробно, когда займемся проектированием наших таблиц.

Хранимая процедура (stored procedure) — программа, обычно выполняющая какие-то действия с данными из базы данных (на самом деле требование работы с данными из базы данных не является обязательным; процедура может выполнять любые действия с любыми доступными ей данными). Хранится в области метаданных базы данных, в системных таблицах. К хранимой процедуре могут обращаться любые программы, работающие с этой базой данных: обычные клиентские программы, хранимые процедуры и триггеры. Допустима также рекурсия — хранимая процедура может обращаться и сама к себе.

Хранимые процедуры выполняются на стороне сервера (серверной машины, где расположена база данных), а не на стороне клиента. В этом проявляется высокая эффективность использования хранимых процедур, когда они обра-

батывают большой объем данных именно на стороне сервера, минимизируя сетевой трафик, т. е. пересылку данных по локальной сети.

Хранимые процедуры могут получать от вызывающей программы параметры и возвращать произвольное количество значений.

Одно из наиболее простых действий, которое можно доверить хранимым процедурам, — получение из генератора значения искусственного первичного ключа. При формировании программой значений столбцов новой строки таблицы, помещаемой в базу данных, выполняется обращение к хранимой процедуре за новым значением первичного ключа. Такие процедуры называются выполняемыми (executable) хранимыми процедурами. Второй вид — хранимые процедуры выбора (selectable), которые позволяют на основании сколь угодно сложных алгоритмов осуществлять выборку данных из произвольных таблиц базы данных.

Триггер (trigger) — как и хранимая процедура является программой, выполняющей действия с данными базы данных. Однако обращение напрямую к триггеру невозможно. Он автоматически вызывается (по-английски trigger fires, т. е. вспыхивает, загорается) при наступлении какого-то события базы данных — удаление записи, помещение новой записи в базу данных, модификация записи. Триггер может вызываться как до наступления такого события, так и сразу после него. Это называется фазой события. Выполнив не слишком сложное умножение двух чисел, получаем шесть различных событий, при которых может срабатывать триггер. Триггер, как и хранимая процедура, выполняется на стороне сервера. Если для его работы требуется обращение к большому количеству данных, то в этом случае, как и при хранимых процедурах, мы получаем большой выигрыш в плане уменьшения сетевого трафика. Триггер может обращаться к любой хранимой процедуре этой базы данных. Триггер не может получать параметров и возвращать какие-либо значения.

Основная прелесть триггеров в том, что программисту не нужно заботиться о выполнении некоторых важных действий в процессе модификации данных в базе данных. Такие действия можно поручить триггеру, который выполнит все необходимое, независимо от того, помнит программист об этих действиях или нет. Например, получение значения искусственного ключа из генератора можно поручить и триггеру, который будет автоматически вызываться перед добавлением новой строки в таблицу. Если программист забыл получить из хранимой процедуры значение такого ключа, то триггер молча исправит такую невнимательность[3].

---

[3] На самом деле мы с вами будем использовать еще более умные и надежные средства работы с искусственными ключами. Но об этом чуть позже, когда мы начнем писать серьезные программы работы с нашей учебной базой данных.

Правда, существует и обратная сторона такой прелести. Приходилось наблюдать случаи, когда в результате внесения изменений в существующую программную систему программист получал совсем не те результаты, которые ожидал увидеть. Часто программисты забывают о существовании триггеров, которые, выполняясь автоматически, вносят непонятные корректировки в данные базы данных.

Для поддержания отношений с помощью связки "внешний ключ/первичный (уникальный) ключ" система автоматически создает системные триггеры. На них мы с вами со временем посмотрим внимательно.

Пользовательские исключения (exception). Для реализации эффективной обработки ошибочных ситуаций можно создавать в конкретной базе данных исключения, задавая в них текст, объясняющий ошибочную ситуацию. В хранимых процедурах и триггерах на основании каких-то проверок данных при выявлении ошибок вы можете выдавать соответствующее исключение, текст которого возвращается вызвавшей программе или программе, инициировавшей вызов триггера. Это весьма удобное средство обработки ошибок. Позже мы с вами создадим несколько исключений в нашей базе данных и используем их по назначению.

События базы данных (event). Из хранимых процедур и триггеров существует возможность отправлять события всем клиентам, подключенным к базе данных и "прослушивающим" конкретные события. Очень интересное средство, позволяющее выполнить, например, синхронизацию состояния текущих наборов данных у всех клиентов при изменении таблиц одним из клиентов.

Язык PSQL. Мы говорили, что в составе SQL выделяются язык описания данных (DDL) и язык манипулирования данными (DML). Существует еще и так называемое расширение языка SQL — процедурный SQL, PSQL. Он используется для написания хранимых процедур и триггеров. Основной особенностью языков DDL и DML является их *декларативность* — на этих языках только описывается, *что* должно быть сделано, но ни при каких обстоятельствах не указывается, *как* это должно быть сделано. Язык PSQL добавляет в реляционные базы данных *императивность*, функциональность — он позволяет описывать именно то, *как* должны быть выполнены действия с данными. В этом языке представлены (хотя и довольно скромно) все основные конструкции, характерные для языков программирования — объявление внутренних, локальных переменных, входных и выходных параметров, присваивание, вычисления, проверка условий, выполнение различных циклов и т. д. Кроме этого, для триггеров допустимо использование так называемых контекстных переменных, позволяющих обращаться к значениям любых столбцов записи до их изменения пользователем (вариант `OLD`) и после изменения (вариант `NEW`).

Представление (view) — динамический результат одной или нескольких реляционных операций над базовыми таблицами с целью создания новой таблицы. Представление является *виртуальной* таблицей, которая в базе данных не хранится, но создается по требованию отдельного пользователя. То, что сейчас было написано про представление, правильно, но, на мой взгляд, для нормального человека (не работавшего с представлениями) не очень понятно. Мы займемся представлениями ближе к концу этой книги. Рассмотрим примеры.

Транзакция (transaction) — механизм базы данных, позволяющий либо подтвердить группу выполняемых операций с базой данных, либо отменить все действия этой группы операций. Любые действия с базой данных — как с данными, так и с метаданными — выполняются в контексте (иными словами — под управлением) какой-либо транзакции. Транзакция стартует перед началом любого действия с базой данных — изменения данных базы данных или выборки существующих данных. По завершении группы операций все действия можно или подтвердить (оператор `COMMIT`), или отменить (оператор `ROLLBACK`). Рассмотрению транзакций мы посвятим отдельную главу.

Функции, определенные пользователем (User Defined Functions, UDF) — функции, написанные на любом языке программирования и хранящиеся вне базы данных, но описанные в базе. Могут использоваться для расширения возможностей языка SQL. В стандартных комплектах поставки серверов базы данных содержится довольно большое количество UDF. В основном это математические функции.

Предметная область — реальная область человеческой деятельности, для решения задач которой мы и создаем наши базы данных и пишем клиентские программы графического интерфейса. Примерами предметной области могут служить система решения задач бухгалтерского учета на предприятии, библиографическая поисковая система, очень сложная система управления ресурсами предприятия (ERP) и многое другое. Именно для решения задач конкретных предметных областей и создаются базы данных и соответствующие клиентские приложения.

Клиентское приложение — программа, использующая данные базы данных для решения задач предметной области. Имеет графический интерфейс, удобный для пользователя. Выполняется на стороне клиента — клиентском компьютере в локальной вычислительной сети. На самом деле нам совершенно все равно, где выполняется клиентское приложение — на сервере или на клиентском компьютере. Необходимые согласования выполнит система управления базами данных.

1.2. Реализация отношений в реляционной модели

Существует три вида отношений в любых базах данных: "один-к-одному", "один-ко-многим" и "многие-ко-многим". Рассмотрим реализацию этих отношений в реляционной модели. При любом отношении все вопросы решаются связкой "внешний ключ/первичный (уникальный) ключ".

1.2.1. Отношение "один-к-одному"

Вообще говоря, если между двумя таблицами базы данных появляется такое отношение, то лучше объединить эти таблицы в одну. Основной причиной использования этого отношения является экономия памяти и увеличение скорости выполнения запросов.

Такое отношение используется в том случае, если связь не является обязательной.

Рассмотрим пример, к сожалению, довольно глупый и надуманный. Не могу припомнить, чтобы в реальной жизни у меня была потребность в реализации отношения "один-к-одному".

Правда, в нашей учебной базе данных будет одно такое отношение внутри одной и той же таблицы. Случай довольно специфический. Это связь между супругами в таблице людей. Вообще, связь, которую мы используем, обычно называют циклической. Далеко не все модели данных ее поддерживают. Реляционные базы данных поддерживают.

Итак, пусть есть две таблицы. Первая содержит некоторые сведения о человеке. Вторая — сведения о его автомобиле: марка, год выпуска и т. д. Считается, что человек может иметь не более одного автомобиля.

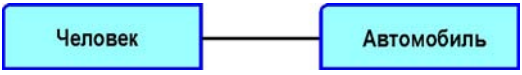


Рис. 1.1. Отношение "один-к-одному"

Здесь связь между человеком и автомобилем не является обязательной. Очень много людей не имеют личного транспорта.

Можно было бы сведения об автомобиле поместить в первую таблицу, однако, поскольку далеко не каждый человек имеет машину, это привело бы к большому количеству пустых полей в таблице и некоторому перерасходу внешней памяти. Правда, перерасход памяти не будет слишком большим, поскольку наши серверы баз данных хранят пустые значения (как и все другие) весьма компактно.

Для реализации отношения "один-к-одному" в данном случае сделаем первичный ключ во второй таблице таким же, как и в первой. Этот первичный ключ также будет и внешним ключом, ссылающимся на первичный ключ первой таблицы.

1.2.2. Отношение "один-ко-многим"

Отношение "один-ко-многим" реализуется связкой "внешний ключ/первичный (уникальный) ключ" (рис. 1.2).

Рис. 1.2. Отношение "один-ко-многим"

Рассмотрим вполне реальный пример. Пусть у нас есть таблица, содержащая список стран (табл. 1.1). Вторая таблица содержит список регионов каждой страны (республики, края, области, в некоторых странах — штаты, графства, земли) (табл. 1.2). Первичным ключом будет некоторый код страны. Во второй таблице нужно ввести поле внешнего ключа (код страны), которое будет ссылаться на первичный ключ первой таблицы.

Таблица 1.1. *Страны*

Краткое название	Полное название	Код страны
РОССИЯ	Российская Федерация	558
США	Соединенные Штаты Америки	604

"Код страны" здесь является первичным ключом.

Таблица 1.2. *Регионы*

Код страны	Код региона	Центр региона	Регион
558	32	Брянск	Брянская область
558	25	Владивосток	Приморский край
558	15	Владикавказ	Республика Северная Осетия
558	33	Владимир	Владимирская область
558	34	Волгоград	Волгоградская область
558	35	Вологда	Вологодская область

Первичный ключ для этой таблицы состоит из двух полей — "Код страны" и "Код региона". Внешний ключ: "Код страны", он ссылается на первичный ключ ("Код страны") таблицы стран. Эти таблицы мы с вами спроектируем более полно чуть позже и поместим в нашу учебную базу данных.

Отношение "один-ко-многим" является универсальным в реляционных, да и не только в реляционных базах данных. С помощью такого отношения мы можем в базе данных представить любые необходимые нам связи.

1.2.3. Отношение "многие-ко-многим"

Для реализации отношения "многие-ко-многим" между двумя таблицами необходимо ввести третью, вспомогательную (или связующую), таблицу.

Рассмотрим пример с книгами и авторами книг. Одна книга может быть написана несколькими авторами, один автор может написать несколько книг. Это типичный пример отношения "многие-ко-многим" (рис. 1.3). Пусть мы имеем две таблицы — авторов (табл. 1.3) и книг (табл. 1.4).

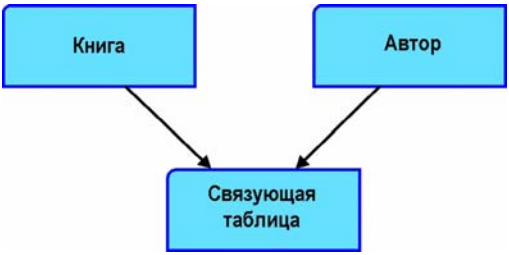


Рис. 1.3. Отношение "многие-ко-многим"

Таблица 1.3. *Авторы*

Код автора	Фамилия автора
2284	Иванов
3187	Петров
1254	Голицына
3492	Максимов
2473	Данилов

Таблица 1.4. *Книги*

Код книги	Название книги
01	Реляционные базы данных
02	Введение в язык SQL
03	Практическое руководство по SQL
04	Структуры данных

Понятно, что в таблице авторов первичным ключом является "Код автора", а в таблице книг — "Код книги".

Для реализации нужного нам отношения необходимо ввести третью, вспомогательную или связующую таблицу (табл. 1.5). Она будет содержать всего два столбца.

Таблица 1.5. *Вспомогательная таблица*

Код автора	Код книги
1254	01
1254	04
2284	02
2284	04
2473	04
3187	01
3187	04

Во вспомогательной таблице первичным ключом является составной ключ "Код автора" и "Код книги".

"Код автора" также является и внешним ключом, ссылающимся на первичный ключ "Код автора" таблицы авторов. "Код книги" — внешний ключ, ссылающийся на первичный ключ "Код книги" таблицы книг.

Просматривая эту таблицу, можно получить полные и точные данные о книгах и их авторах. Например, можно увидеть, что авторами книги "Реляционные базы данных" (код книги 01) являются Голицына и Петров (коды авторов 1254 и 3187 соответственно). Иванов же (код 2284) принимал участие в на-

писании книг "Введение в язык SQL" (код 02) и "Структуры данных" (код 04).

Мы с вами создадим немало таблиц, между которыми будут отношения "один-ко-многим" и "многие-ко-многим".

1.3. Нормализация таблиц

В основе реляционных баз данных лежит строгий математический аппарат, хорошо формализованные принципы создания баз данных и управления ими (на практике, правда, эти принципы постоянно нарушаются разработчиками СУБД и использующими их программистами).

Все данные в реляционной базе данных хранятся в однородных таблицах. Термин "однородный" означает, что в одной таблице *все* строки имеют одинаковую структуру. От правильности проектирования таблиц зависит простота эксплуатации созданной базы данных, возможность ее развития в дальнейшем. Хорошо спроектированная база данных помимо всех других прелестей обладает еще одним замечательным свойством — из этой базы данных можно получать такие данные, о которых никто и не думал в процессе проектирования базы. Для получения "правильных" таблиц используется процесс нормализации.

Существует пять общепризнанных так называемых нормальных форм таблиц. На практике чаще всего используется третья нормальная форма. Мы и рассмотрим довольно подробно первые три, а про четвертую и пятую нормальную форму упомянем лишь вскользь. Каждая последующая нормальная форма требует соблюдения требований предыдущей нормальной формы и устанавливает дополнительные требования к структуре таблицы.

Выполнение правил нормализации обычно приводит к разделению нормализуемой таблицы на две или более таблиц с меньшим количеством столбцов. Эти таблицы за счет связки "внешний ключ/первичный (уникальный) ключ" снова могут быть объединены в процессе выборки данных с помощью операции соединения.

Одним из основных результатов разделения таблиц в соответствии с правилами нормализации является уменьшение избыточности данных.

1.3.1. Первая нормальная форма

Первая нормальная форма требует, чтобы значение любого столбца было единственным, атомарным. Иными словами, в таблице не должно быть повторяющихся групп. Общепринятое сокращенное название для первой нормальной формы — *1НФ*.

Пример. Пусть имеется справочник стран — таблица COUNTRY (табл. 1.6). Для каждой страны в записи указан список регионов.

Таблица 1.6. *Страны*

Код страны	Название страны	Центр региона
558	Россия	Брянск Владивосток Владикавказ
		Владимир Волгоград

Таблица содержит повторяющуюся группу: "Центр региона", что нарушает требования первой нормальной формы. В данном случае нужно список регионов вынести в отдельную таблицу REGION (табл. 1.7), связав эту таблицу с главной таблицей COUNTRY с помощью внешнего ключа. В таблице же COUNTRY нужно оставить столбцы "Код страны" (первичный ключ) и "Название страны".

Таблица 1.7. *Регионы*

Код страны	Код региона	Центр региона
558	32	Брянск
558	25	Владивосток
558	15	Владикавказ
558	33	Владимир
558	34	Волгоград
558	35	Вологда

Первичным ключом здесь будет составной ключ — "Код страны" и "Код региона". Кроме того, "Код страны" является внешним ключом, ссылающимся на первичный ключ таблицы стран.

1.3.2. Вторая нормальная форма

Вторая нормальная форма (2НФ) требует, чтобы соблюдались условия первой нормальной формы, и чтобы любой неключевой столбец зависел от всего первичного ключа. То есть таблица не должна содержать неключевых столб-

цов, зависящих только от части составного первичного ключа. Разумеется, это правило относится только к тому случаю, когда первичный ключ образован из нескольких столбцов.

Пример. Пусть таблица регионов `REGION` имеет следующий вид — табл. 1.8.

Таблица 1.8. *Регионы*

Код страны	Код региона	Центр региона	Страна
558	32	Брянск	Россия
558	25	Владивосток	Россия
558	15	Владикавказ	Россия
558	33	Владимир	Россия
558	34	Волгоград	Россия
558	35	Вологда	Россия

Первичный ключ для этой таблицы состоит из двух полей — "Код страны" и "Код региона". Столбец "Страна" зависит от части первичного ключа: "Код страны". Этот столбец следует из таблицы убрать. Мы всегда можем получить название страны из таблицы стран, используя значение внешнего ключа "Код страны".

1.3.3. Третья нормальная форма

Третья нормальная форма (ЗНФ) требует соблюдения второй нормальной формы, а также чтобы ни один неключевой столбец не зависел от другого неключевого столбца.

Для примера рассмотрим таблицу, описывающую отделы организации (табл. 1.9).

Таблица 1.9. *Отделы*

Код отдела	Название отдела	Код руководителя	Фамилия руководителя
01	Продажи	384	Петров
02	Маркетинг	291	Иванов
03	Бухгалтерия	124	Макаров

"Код отдела" является первичным ключом. Столбец "Фамилия руководителя" зависит не от первичного ключа, а от неключевого столбца "Код руководителя". Столбец "Фамилия руководителя" следует убрать из таблицы (табл. 1.10).

Таблица 1.10. *Отделы (после удаления столбца)*

Код отдела	Название отдела	Код руководителя
01	Продажи	384
02	Маркетинг	291
03	Бухгалтерия	124

При этом необходимо добавить в базу данных новую таблицу — список сотрудников (табл. 1.11). Эта таблица должна содержать, как минимум, код сотрудника и фамилию сотрудника.

В таблице отделов (табл. 1.10) нужно сделать столбец "Код руководителя" внешним ключом, ссылающимся на первичный ключ таблицы сотрудников.

Таблица 1.11. *Сотрудники*

Код сотрудника	Фамилия сотрудника
384	Петров
291	Иванов
124	Макаров

1.3.4. Четвертая и пятая нормальные формы

Четвертая нормальная форма (4НФ) требует, естественно, выполнения условий третьей нормальной формы и запрещает независимые отношения типа "один-ко-многим" между ключевыми и неключевыми столбцами. Мне, честно скажу, не хочется рассматривать требования 4НФ, поскольку считаю их несколько надуманными (поправьте меня, если я ошибаюсь).

А вот *пятая нормальная форма (5НФ)* доводит процесс нормализации до логического конца (то есть до бессмысленности), разбивая таблицы на минимально возможные части для устранения в них всей избыточности данных (однако добавляя избыточное повторение ключевых полей в различных таблицах).

1.3.5. Контрольное задание

Тренировки ради выполните разработку таблицы или таблиц, которые позволят хранить данные из накладной, содержащей сведения о полученных товарах от какой-то организации.

В накладной, как правило, содержатся следующие данные:

- номер накладной;
- дата;
- название организации-поставщика.

Список поставленных товаров. Каждый элемент списка содержит:

- наименование товара;
- цена товара;
- количество поставленного товара;
- стоимость поставленного товара.

Кроме того, в накладной содержится и итоговая сумма всех поставленных товаров.

Приступайте к разработке.

***

Думаю, что прямо сейчас вы поразмышляли и разработали необходимые структуры данных. А я покажу свой вариант решения этой задачи.

Разумеется, одной таблицей эти данные представлять нельзя, потому что наличие повторяющейся группы нарушает требование первой нормальной формы. Мы создадим две таблицы. Одна будет содержать общие данные по накладной (табл. 1.12). Другая должна включать данные по каждому товару из накладной (табл. 1.13).

Опишем таблицы базы данных с помощью следующих обычных таблиц.

Таблица 1.12. *Накладная*

Поле	Тип данных
Номер накладной. Первичный ключ	Строка
Дата накладной	Дата
Организация-поставщик	Строка
Итоговая сумма по накладной	Число

В табл. 1.12 мы приняли допущение, что номер накладной является уникальным. По этой причине мы выбираем его в качестве первичного ключа. В реальной же жизни мы, скорее всего, в качестве первичного ключа использовали бы искусственный первичный ключ, добавив в таблицу столбец с числовым типом данных, и создав в базе данных генератор.

В детальной строке, описывающей отдельный товар накладной, мы из естественных реквизитов уж точно не можем найти нормального кандидата в первичные ключи. По этой причине сразу принимаем решение использовать искусственный первичный ключ. Вообще-то можно было бы в эту таблицу ввести реквизит "Номер строки", задающий порядковый номер товара в накладной. Тогда в состав первичного ключа можно включить два столбца — "Номер накладной" и "Номер строки".

Детальная строка может выглядеть следующим образом — табл. 1.13.

Таблица 1.13. *Детальная строка накладной*

Поле	Тип данных
Искусственный первичный ключ	Число
Номер накладной. Внешний ключ	Строка
Наименование товара	Строка
Цена товара	Число
Количество поставленного товара	Число
Стоимость поставленного товара	Число

Между накладной и детальной строкой существует отношение "один-ко-многим". По этой причине мы добавили к детальной строке столбец "Номер накладной", который является внешним ключом, ссылающимся на первичный ключ таблицы накладной.

Задача решена.

> ***Замечание***
> Кому-то подобная задача может показаться неинтересной и нудной. Однако хочу напомнить вам, что все наши реальные разработки должны решать задачи, стоящие перед человечеством, т. е. из конкретных предметных областей. Те же накладные являются одной из таких реальностей.

1.4. 12 правил Кодда

Доктор Кодд предложил 13 правил определения реляционных систем. Их обычно почему-то называют "12 правил доктора Кодда".

Я приведу эти правила просто для порядка, чтобы вы потом не упрекали меня, что я что-то от вас скрыл. При этом у меня нет твердой уверенности, что они нам с вами когда-нибудь пригодятся, поскольку мы все-таки используем существующие системы управления базами данных, а не разрабатываем их.

Правило 0 — фундаментальное правило.

> Любая реляционная СУБД должна быть способна управлять данными исключительно с помощью реляционных функций.

Правило означает, что в СУБД не должны применяться какие-либо нереляционные операции для определения данных и манипулирования ими.

Правило 1 — представление данных.

> Все данные в реляционной базе данных представляются в явном виде на логическом уровне и только одним способом — в виде значений в таблицах.

Все данные, включая метаданные, должны храниться в виде отношений и управляться с помощью тех же функций, которые используются для работы с данными.

Правило 2 — гарантированный доступ.

> Для каждого элемента данных должен быть гарантирован логический доступ на основе использования комбинации имени таблицы, значения первичного ключа и имени столбца.

Правило 3 — обработка неопределенных значений (*NULL*).

> Неопределенные значения являются способом представления отсутствующих или неприемлемых данных независимо от типа данных.

Неопределенные значения — значения, отличные от пустой строки, строки с пробельными символами, а также от нуля или любого другого числа.

Правило 4 — динамический каталог, основанный на реляционной модели.

> Описание данных должно быть представлено на логическом уровне таким же образом, что и обычные данные.

Правило дает возможность пользователям для обращения к этому описанию применять тот же реляционный язык, что и при обращении к обычным данным.

Правило 5 — исчерпывающий подъязык данных.

Реляционная система может поддерживать несколько языков. Однако должен существовать по крайней мере один язык, который позволял бы выражать следующие конструкции:

1) определение данных;

2) определение представлений;

3) операторы манипулирования данными;

4) ограничения целостности;

5) авторизация пользователей;

6) поэтапная организация транзакций.

Правило 6 — обновление представлений.

Все представления, которые являются теоретически обновляемыми, должны быть обновляемыми в данной системе.

Правило 7 — высокоуровневые операции добавления, обновления и удаления.

Способность обрабатывать базовые и производные (т. е. представления) таблицы в виде отдельного оператора на языке высокого уровня.

Правило 8 — физическая независимость от данных.

Прикладные программы и средства работы с терминалами должны оставаться логически неизменными при внесении любых изменений в способы хранения данных или методы доступа к ним.

Правило 9 — логическая независимость от данных.

Прикладные программы и средства работы с терминалами должны оставаться логически неизменными при внесении в базовые таблицы любых не меняющих данные изменений, которые теоретически не должны затрагивать прикладное программное обеспечение.

Правило 10 — независимость ограничений целостности.

Ограничения целостности данных должны определяться на языке реляционных данных и храниться в базе данных, а не в прикладных программах.

Правило 11 — независимость от распределения данных.

> Язык манипулирования данными должен позволять прикладным программам и запросам оставаться логически неизменными, независимо от того, как хранятся данные — физически централизованно или в распределенном виде.

Правило 12 — запрет обходных путей.

> Если система имеет язык низкого уровня, он не может быть использован для отмены или обхода правил и ограничений целостности, составленных на языке более высокого уровня.

На языке низкого уровня можно обрабатывать одновременно одну запись. Язык высокого уровня одновременно обрабатывает сразу несколько записей.

1.5. Язык SQL

1.5.1. Назначение языка SQL

В реляционных базах данных используется язык SQL. В составе языка выделяется язык описания данных (DDL — Data Definition Language) и язык манипулирования данными (DML — Data Manipulation Language).

DDL используется для создания, изменения и удаления метаданных базы данных — таблиц, доменов, индексов, генераторов, хранимых процедур, триггеров.

Операторы DML используются для манипулирования собственно данными базы данных — добавление, изменение, удаление данных, а также выборка данных из таблиц базы данных. Язык SQL InterBase и Firebird очень близок к стандарту SQL-92.

1.5.2. Синтаксические конструкции

Общаясь с разными категориями программистов или людей, собирающихся стать программистами, я заметил одну очень странную для меня деталь — оказывается мало кто знает такое слово, как "синтаксис" применительно к формальным языкам (для нас это, в первую очередь, языки программирования и язык SQL, который по способам построения мало отличается от обычного языка программирования). А в некоторых учебных группах слово "семантика" воспринималось вообще как неприличное. Про прагматические аспекты языков я стараюсь вообще нигде не упоминать. Иначе меня могут при-

влечь к административной ответственности за использование ненормативной лексики в общественном месте.

На самом деле все очень просто. Под синтаксисом языка SQL мы будем понимать множество правильных конструкций языка, а также способы построения таких правильных конструкций. Семантика же — это смысл, значение правильно построенных конструкций. Прагматика связана с практической стороной использования языковых конструкций, их полезностью.

Для описания синтаксиса языка будем использовать нотации (систему обозначений) Бэкуса-Наура, принятые в программной документации. Для описания синтаксических конструкций различных формальных языков, с которыми мне приходится работать, я часто использую так называемые R-графы — необыкновенно удобное и наглядное средство. Чисто технические сложности не позволяют мне сейчас поделиться с вами умением использовать такую замечательную графику. В будущем обязательно покажу вам эти средства.

Семантика любых языков определяется обычным человеческим языком.

В нотациях Бэкуса-Наура символы `::=` означают "по определению есть".

Нетерминальные символы (т. е. те, которые будут в дальнейшем уточняться) заключаются в угловые скобки `<` и `>`.

Конструкция, которая не является обязательной и может быть опущена, заключается в квадратные скобки `[` и `]`.

Символ вертикальной черты `|` означает "или".

Если группа элементов заключена в фигурные скобки `{` и `}` и разделена символами вертикальной черты `|`, то следует выбрать из списка ровно один элемент.

Если группа элементов заключена в квадратные скобки `[` и `]` и разделена символами вертикальной черты `|`, то можно выбрать из списка произвольное количество элементов или ни одного элемента.

Многоточие (`...`) означает, что предыдущий фрагмент синтаксиса может повторяться несколько раз. В большинстве синтаксических конструкций, используемых для описания операторов работы с базами данных, очень часто именно в этом месте допускаются неприятные ошибки. Их мы с вами будем смело исправлять.

Символы `::=`, `<`, `>`, `|`, `[`, `]`, `{`, `}` и `...` называются *металингвистическими переменными*, т. е. переменными, которые не входят в состав описываемого языка, а используются в *метаязыке* — языке, предназначенном для описания

нашего языка SQL[4]. Обратите внимание на использование в нашем, человеческом, языке приставки мета- (метаданные, метаязык).

Дадим сначала формальные определения таких простейших понятий, как "цифра" и "буква".

```
<цифра> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<буква> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
            a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z
```

То есть буквой в языке SQL является прописная или строчная буква латинского алфавита. Символы кириллицы не входят в состав букв, впрочем, как и буквы других алфавитов, отличных от латинского.

Продолжим.

```
<оператор сравнения> ::= <> | = | > | < | >= | <= | != | !> | !<
```

Обратите внимание, что восклицательный знак используется в языке в качестве логического отрицания.

В нотациях Бэкуса-Наура "имя" определяется следующим образом:

```
<имя> ::= <буква> | <имя><буква> | <имя><цифра> | <имя><подчеркивание> |
<имя>$
```

Это, пожалуй, первая достаточно сложная синтаксическая конструкция, которую изучает человек, начинающий пользоваться такими нотациями. Мы видим здесь, что имя должно начинаться с буквы, за которой следует произвольное количество букв, цифр, символов подчеркивания и символов доллара. Другие символы, например, буквы кириллицы, в имени недопустимы.

Имя в языке SQL нечувствительно к регистру. Например, такие имена, как `Country`, `COUntry` и `COUNTRY`, являются одинаковыми.

Максимальное количество символов в именах разных объектов базы данных различно. Встречаются величины 27, 31, 32 (из них только 8 значимые) и 67 (это InterBase 7.*x*). Если для именования объектов вы будете использовать не более 27 символов, то не ошибетесь. Да и это много. Использование слишком длинных имен весьма утомительно, поверьте мне. Каждый раз приходится долго и нудно вводить большое количество символов (конечно, можно их копировать и вставлять в нужные места текста). Кроме появляющейся уста-

---

[4] То, что ни один из этих символов не входит в состав языка SQL, не совсем верно. Квадратные скобки используются при задании размерности массивов. Мы с вами, по крайней мере в рамках этой книги, массивами пользоваться не будем по двум причинам — во-первых, надобность в них не так уж велика, а во-вторых, их использование нарушает требование первой нормальной формы. Кроме этого, символы < и > входят в состав операторов сравнения.

лости в пальчиках рук сильно увеличивается вероятность ошибки. Слишком длинные имена к тому же плохо узнаваемы человеком.

Помимо обычных имен в SQL, начиная с InterBase версии 6.0, появились еще имена с ограничителями (delimited name):

```
<имя с ограничителями> ::= "<любой символ ASCII> ..."
```

Имя с ограничителями заключается в кавычки и может содержать любые символы, включая буквы кириллицы и пробелы. Такое имя *чувствительно* к регистру. Заметьте, что здесь при описании синтаксиса такого имени мы использовали многоточие, которое означает, что предыдущая конструкция (любой символ ASCII) может повторяться произвольное количество раз.

В этой книге мы с вами не будем пользоваться именами с ограничителями. В какой-то версии какого-то сервера базы данных при каких-то обстоятельствах я нашел ошибку, повлиявшую на правильность выполнения некоторых операций с базой данных. После этого я решил использовать только обычные имена. Вы, конечно же, выберете то, что вам по вкусу.

Тренировки ради попробуйте прямо сейчас самостоятельно описать синтаксис таких конструкций, как "целое", "целое со знаком" и "число с плавающей точкой".

Первые две я опишу, потому что для меня это очень просто, число же с плавающей точкой опишите сами, поскольку у меня нет достаточно красивой и компактной версии описания такого синтаксиса.

```
<целое> ::= <цифра>...
```

Проще некуда. Это цифра, которая может повторяться произвольное количество раз.

```
<целое со знаком> ::= { + | – }<целое>
```

Если помните, в числе с плавающей точкой можно указать порядок, степень числа 10, которая может быть со знаком. Значению степени предшествует строчная или прописная латинская буква e. Например, число –27.1 может быть представлено в виде –0.271E+2. Это так называемый нормализованный вариант. То же число можно представить множеством различных способов, в том числе и так: –271e–1.

1.5.3. Основные операторы SQL

Как вы помните, в базе данных хранятся собственно данные и метаданные — описания этих данных.

Язык DDL позволяет работать с метаданными. Для создания любого объекта метаданных используется оператор `CREATE`, для изменения объекта — оператор `ALTER`, для удаления объекта — оператор `DROP`.

Для работы с данными базы данных используется язык DML. Для помещения новой строки в таблицу применяется оператор `INSERT`, для изменения — `UPDATE`, для удаления — `DELETE`. Выборка данных осуществляется с помощью оператора `SELECT`.

> ***Замечание***
>
> По правде сказать, не очень мне нравится такое свободное использование терминологии. DDL и DML, несмотря на их названия, не являются самостоятельными языками. Это подмножества языка SQL, равно как и PSQL. В отечественной научно-технической литературе до ее американизации существовали довольно высокие требования к использованию терминов, что сейчас, к сожалению, утрачено.

Все эти операторы и многие другие мы достаточно подробно будем рассматривать в данной книге.

1.6. Обращение к программам системы управления базами данных

При работе с базами данных можно выделить четыре уровня взаимодействия пользователей с базой данных.

- Первый, самый низкий уровень, называется интерфейсом прикладного программирования, API (Application Programming Interface). Пользователем такого уровня является так называемый системный программист. В приложении формируется список параметров для выполнения каких-либо действий с базой данных и выполняется обращение к ядру СУБД при использовании соответствующей функции API. В оригинале ядро СУБД называется engine — движок, двигатель. Такая форма работы с базой данных весьма трудоемкая и утомительная. Вряд ли мы с вами когда-либо будем этим заниматься. Разработано множество компонентов, облегчающих наш программистский труд. Использование таких компонентов — это уже второй уровень.
- На втором, более высоком, уровне нам предлагаются различные наборы компонентов, с помощью которых мы можем выполнять все необходимые действия с базой данных, не влезая в дебри и тонкости API. Мы используем свойства, события и методы таких компонентов для работы с базой данных. В результате все равно обращение к СУБД будет происходить через вызовы функций API, однако от нас это, к счастью, скрыто. Нам не понадобится создавать и инициализировать дескрипторы, формировать блоки параметров, описывать какие-то константы и использовать не очень понятные имена функций API, например, `isc_attach_database`. На этом уровне работает большинство программистов, которых называют проблемными или обычными программистами.

- На третьем уровне нам предлагается язык SQL, на котором мы можем описать все наши потребности по созданию, изменению, удалению и поиску всевозможных данных в базе данных. Для выполнения операторов SQL у нас есть возможность использования различных программ. Это и утилита командной строки isql, входящая в стандартный комплект поставки соответствующей СУБД, это и множество программ графического интерфейса, например, IBExpert или BlazeTop. Так или иначе, эти программы используют функции API для обращения к базе данных, в большинстве случаев не напрямую, а опосредованно, через соответствующие компоненты. Программы IBExpert и BlazeTop используют, насколько мне известно, компоненты FIBPlus, которые и мы с вами в процессе работы с этой книгой научимся применять с большой эффективностью.
- Четвертый уровень доступен нормальному пользователю, имеющему некоторые навыки работы с компьютером (трудно сейчас найти людей, не имеющих таких навыков). Для пользователей этого уровня пишется специальная программа, решающая задачи конкретной предметной области, или группа программ. Такие программы имеют графический интерфейс. Пользователю не нужно знать SQL, он привычным для себя образом использует мышь и клавиатуру для выполнения необходимых действий и полагается на то, что программа разработана по принципу "интуитивного интерфейса" — содержит главное меню, кнопки быстрого доступа, контекстные меню, подсказки, диалоговые окна для выбора режимов, функций и т. д. Кто же создаст этому пользователю такие замечательные программы? Разумеется, мы с вами. Для написания хороших программ нам с вами нужно будет освоить достаточно большой объем информации из области программирования и реляционных баз данных. Собственно, для этого и предназначена книга, которую вы держите в руках или разглядываете на мониторе.

Итак, мы решили, что первым уровнем мы пока заниматься не будем. Может быть, со временем у нас появятся соответствующее желание и возможности.

Вам наверняка приходилось работать пользователем на четвертом уровне. Используя какую-либо программу для решения своих задач, вы не слишком задумывались над тем, как она устроена, с какими работает данными и как организованы эти данные.

Основная наша деятельность на сегодняшний день будет связана со вторым и третьим уровнями. Мы будем активно осваивать возможности операторов языка SQL, выполнять множество интересных действий с базой данных — создавать и изменять метаданные, помещать новые данные в базу, изменять и удалять существующие данные, выполнять всевозможные, порой очень сложные, выборки данных из базы.

Кроме того, мы напишем немало хороших программ, осуществляющих работу с нашей учебной базой данных. Эти программы будут не только иллюстрацией методов использования реляционных баз данных, но смогут послужить хорошим прототипом для создания реальных, промышленных, программ, которые вам, возможно, понадобится разрабатывать в ближайшем будущем.

Что там за перевалом?

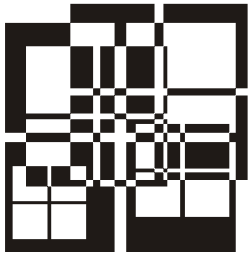
Здесь мы вкратце рассмотрели основные идеи, понятия, принципы реляционных баз данных. Достаточно подробно поговорили о таблицах, о правилах и целях нормализации таблиц. Рассмотрели способы реализации отношений в реляционных базах данных. Познакомились с некоторыми самыми простыми языковыми конструкциями SQL. В дальнейшем нам предстоит рассмотреть довольно много различных синтаксических конструкций, и подчас весьма сложных.

Теперь пора приступать к практической деятельности. Устанавливаем на своем компьютере какую-либо версию InterBase или Firebird, ставим IBExpert или BlazeTop *(см. приложение 1)* и начинаем исследовательскую и практическую работу.

Вначале мы создадим новую учетную запись пользователя. Затем будем создавать, удалять и снова создавать учебную базу данных. Научимся копировать и восстанавливать базу данных. Все это будем делать, используя утилиты командной строки, входящие в стандартный состав поставки сервера базы данных, а также программу графического интерфейса — я подробно описываю работу с IBExpert.

Особо одаренные (см. название этой книги) напишут множество собственных программ (клиентских приложений) для выполнения всех этих и многих других действий. Под моим чутким руководством.

ГЛАВА 2



Создание базы данных

2.1. Учетные записи пользователя

Для доступа различных пользователей ко всем базам данных одной машины (обычно сервера) в InterBase/Firebird используется база данных безопасности. Это файл базы данных, хранящийся в корневом каталоге инсталляции сервера базы данных и имеющий имя:

- isc4.gdb — для InterBase 6.*x* и Firebird 1.0.*x*;
- admin.ib — для InterBase 7.*x*;
- security.fdb — для Firebird версии 1.5 и выше.

После инсталляции системы база данных безопасности содержит ровно одну учетную запись — пользователь `SYSDBA` с паролем `masterkey`. Это особый пользователь системы. Он имеет неограниченный доступ и неограниченные полномочия к любой базе данных, находящейся на серверной машине. Изменить имя этого пользователя или удалить его из базы данных безопасности невозможно, но можно поменять пароль, что весьма рекомендуется сделать сразу после инсталляции системы в промышленно работающей вычислительной сети.

Пользователь `SYSDBA` может создавать любое количество учетных записей для различных категорий пользователей (табл. 2.1).

Обязательными являются только имя пользователя и пароль. Никакое значение не может содержать букв кириллицы.

Для работы с учетными записями базы данных безопасности (отображение, добавление, изменение, удаление) используется утилита gsec или какая-либо из программ графического интерфейса.

Таблица 2.1. *Структура учетной записи пользователя*

Столбец	Описание
`UserName`	Имя пользователя. Максимальная длина 31 символ. Имя не чувствительно к регистру
`Password`	Пароль пользователя. Чувствителен к регистру. Максимальное количество символов 32, однако только первые восемь имеют значение
`UID`	Целое число, задающее идентификатор пользователя
`GID`	Целое число, задающее идентификатор группы
`FirstName`	Имя. Может содержать до 17 символов
`MiddleName`	Второе имя. Можем использовать в качестве отчества. До 17 символов
`LastName`	Фамилия. До 17 символов

2.2. Создание новой учетной записи пользователя

Для работы с учебной базой данных нужно создать новую учетную запись пользователя, которую вы будете использовать при создании базы данных, объектов базы данных, при заполнении базы данных и при поиске в ней необходимых данных.

Для этих целей можно использовать утилиту gsec, входящую в комплект поставки сервера базы данных, и программу графического интерфейса, например, IBExpert или BlazeTop. Кроме того, используя компоненты FIBPlus или стандартные компоненты IBX, мы можем написать собственную программу, выполняющую необходимые действия и имеющую подходящий интерфейс. Рассмотрим все эти варианты.

2.2.1. Использование gsec

Создадим новую учетную запись с использованием утилиты командной строки gsec.

Вызовите окно командной строки **Пуск | Все программы | Стандартные | Командная строка**. Появится обычное окно командной строки DOS.

> ***Совет***
> Если на вашем компьютере вы по какой-либо причине не можете найти элемент **Командная строка** (мне попадались такие компьютеры), щелкните по кнопке **Пуск**, выберите элемент **Выполнить** и в появившемся окне **Запуск программы** введите текст `cmd` и щелкните по кнопке **ОК**. Тогда точно появится окно командной строки.

Сделайте текущим диск, где находится установленный сервер базы данных, введя имя диска и двоеточие, например:

```
c:
```

После этого войдите в подкаталог \Bin корневого каталога инсталляции сервера базы данных, например:

```
cd \Program Files\Firebird\Firebird_1_5\Bin
```

> ***Совет***
> Если вам сложно сразу в одной строке набрать полный путь к каталогу и при этом не ошибиться, можете разбить команду на несколько частей, постепенно продвигаясь к нужному каталогу, например:
>
> ```
> cd \Program Files
> cd Firebird
> cd Firebird_1_5\Bin
> ```

> ***Замечание***
> Здесь указан путь для Firebird 1.5. В случае Firebird 2.0 следует задать:
>
> ```
> cd \Program Files\Firebird\Firebird_2_0\Bin
> ```
>
> Для InterBase 2007 требуется другой путь:
>
> ```
> cd \Program Files\Borland\InterBase\bin\
> ```

Запустите утилиту gsec, введя:

```
gsec -user sysdba -password masterkey
```

Помните, что все переключатели, команды, операторы и значения не являются чувствительными к регистру (кроме пароля) — вы можете вводить их как строчными, так и прописными буквами. Например, предыдущую строку можно ввести и в следующем виде:

```
gsec -UsEr sysdba -pASSWord masterkey
```

Это также работает, хотя не думайте, что подобная манера написания команд и операторов вызовет у кого-то одобрение.

Если вы неверно введете имена переключателей или ошибетесь при вводе имени пользователя или пароля, то получите ошибку и сообщение о неверном имени и пароле приблизительно следующего вида:

```
Your user name and password are not defined. Ask your database adminis-
trator to set up a Firebird login.
unable to open database
(Ваше имя пользователя и пароль не определены. Спросите у вашего адми-
нистратора базы данных, как установить связь с Firebird.
Невозможно открыть базу данных)
```

> **Совет**
>
> Если при вводе команды DOS вы допустили ошибку, вы сможете снова вернуться к этой команде, нажав клавишу "стрелка вверх". Внесите исправления в команду и снова запустите ее на выполнение, нажав клавишу <Enter>. Аналогично, в любой утилите командной строки при работе в диалоговом режиме вы можете просматривать и корректировать ранее введенные строки, используя клавиши управления курсором "стрелка вверх" и "стрелка вниз".

Исправьте ошибки и снова повторите ввод. Появится подсказка утилиты:

```
GSEC>
```

Отобразите учетные записи, набрав `display` и нажав клавишу <Enter>.

Если вы не добавляли учетных записей пользователей, то будет отображена только учетная запись пользователя `SYSDBA`. Обратите внимание, что пароль не выводится. Вообще в окне DOS отображение не очень удобное для восприятия, потому что размер экрана (80 символов) явно маловат для размещения всей строки.

Добавьте новую учетную запись с именем `WIZARD` и паролем `master`. Для этого используется команда `ADD`. Переключатели этой команды приведены в табл. 2.2.

Таблица 2.2. *Переключатели команды* `ADD`

Переключатель	Назначение
Нет	Первый позиционный параметр. Имя пользователя
`-pw`	Пароль пользователя
`-uid`	Идентификатор пользователя
`-gid`	Идентификатор группы
`-fname`	Имя
`-mname`	Второе имя
`-lname`	Фамилия

Введите:

```
GSEC> add wizard -pw master
```

Если вы попытаетесь ввести пользователя, который уже существует в базе данных безопасности, то после выдачи сообщения об ошибке программа завершит свою работу. Следите за текстом подсказки на экране.

Отобразите обновленный список пользователей, используя команду `display`.

Поэкспериментировав с программой, замечаем, что в именах и пароле мы можем использовать любые символы, находящиеся на клавиатуре, за исключением букв кириллицы (кириллица невозможна в Firebird версии 1.5.0 и более ранних). Всевозможные знаки препинания также не дают ошибок. Например, допустимо следующее "имя":

```
GSEC> add *1-.,:+ -pw ...
```

Во избежание в дальнейшем возможных неприятностей и с целью сохранения общих правил именования объектов базы данных используйте в именах и пароле только буквы латинского алфавита, цифры, знак доллара и символ подчеркивания. Ни в коем случае не пытайтесь использовать буквы кириллицы. Такие записи, созданные в одной программе (если они действительно будут созданы), например в isql, будут неправильно читаться другими программами.

Чтобы изменить значение любого поля учетной записи, кроме имени, используется команда `modify`, параметры которой совпадают с командой `ADD`. Для удаления значения какого-либо столбца в записи нужно указать переключатель и не задавать никакого значения.

Если вы работаете в сети, то, используя утилиту gsec, вы можете с клиентского компьютера работать с базой данных безопасности на серверной машине или на любой другой машине, находящейся в сети. Для этого нужно при запуске утилиты добавить переключатель `-database`. Например, можно ввести (одной строкой):

```
gsec -user sysdba -password masterkey –database
server:c:\Firebird\security.fdb
```

После этого в диалоговом режиме можно отображать и изменять учетные записи пользователей на сервере.

ВНИМАНИЕ!

Имя базы данных в сети всегда можно задать для протокола TCP/IP. Этот протокол поддерживается всеми нормальными операционными системами. Однако для отдельных инсталляций операционной системы можно использовать и протокол Named Pipes (именованные каналы), который иногда называют протоколом NetBEUI. В этом случае путь к базе данных будет следующим:

```
–database \\server\c:\Firebird\security.fdb
```

Если в именах каталогов, которые включаются в путь к базе данных безопасности, присутствуют пробелы, то все значение переключателя `-database` должно заключаться в кавычки. Например, необходимо использовать кавычки в следующем случае (это размещение базы данных безопасности Firebird 1.5 по умолчанию):

```
"server:c:\Program Files\Firebird\Firebird_1_5\security.fdb"
```

> ***Замечание***
>
> Серверы InterBase/Firebird дают возможность с клиентского компьютера обращаться и к базам данных сервера, находящимся на дисках, недоступных средствами Windows клиентскому компьютеру.

Поля `UID`, `GID`, `fname`, `mname` и `lname` сервером базы данных в настоящий момент не используются. Вы можете размещать в них любые данные, кроме букв кириллицы. (На самом деле в некоторых версиях сервера базы данных туда могут помещаться и символы кириллицы, однако в разных программах они будут отображаться по-разному, что нам, конечно, не очень нужно.)

Для удаления любой учетной записи пользователя кроме `SYSDBA` используется команда `DELETE`:

```
GSEC> delete wizard
```

Вы можете смело вводить и выполнять команду удаления пользователя `SYSDBA`, это не даст никаких результатов.

Для завершения работы с утилитой gsec введите:

```
GSEC> QUIT
```

и нажмите клавишу <Enter>. Для выхода из окна командной строки введите `exit` или закройте окно обычным для Windows способом.

Если вы используете любимое в нашем отечестве программное средство Total Commander, то можете вызвать утилиту gsec следующим образом. Сделайте на одной из панелей текущим каталог \Bin, находящийся в корневом каталоге инсталляции сервера базы данных, и в командной строке (нижняя часть окна) введите:

```
gsec -user sysdba -password masterkey
```

Появится обычное окно командной строки с подсказкой утилиты. После этого вы можете выполнять необходимые команды утилиты. Для завершения работы с утилитой и с окном командной строки в этом случае введите:

```
GSEC> QUIT
```

2.2.2. Использование IBExpert

Те же действия можно выполнить при использовании программы IBExpert. И с гораздо большим комфортом.

Запустите программу IBExpert. Щелкните мышью по элементу меню **Tools** и в списке выберите **User Manager**. Появится окно **User Manager**. В выпадающем списке **Server** выберите **(local)**.

Щелкните по кнопке **Connect** справа от выпадающего списка **Server**. Появится окно **Server Login** (рис. 2.1).

> ***Замечание***
>
> Если вы работаете на компьютере, который не находится в локальной сети, то сразу после вызова **User Manager** в поле **Server** будет задано **(local)** и появится окно **Server Login**, в котором нужно вводить имя и пароль пользователя. Впрочем, подобное поведение я наблюдал и на компьютерах, объединенных в сеть. И наоборот, на локальном компьютере появлялась кнопка **Connect**. С чем это связано, я не разобрался.

Рис. 2.1. Диалоговое окно соединения с сервером базы данных в IBExpert

В поле **Login** уже будет задано имя пользователя `SYSDBA`. Вам нужно в поле **Password** ввести пароль `masterkey` (или другой, если вы изменили значение пароля) и щелкнуть по кнопке **OK**. Появится список учетных записей, который будет содержать только одного пользователя `SYSDBA`, если вы не добавляли новых пользователей (рис. 2.2).

Щелкните по кнопке **Add** (Добавить), в появившемся окне **New User** (рис. 2.3) введите имя пользователя (**Name**) `WIZARD` (имя пользователя вы можете вводить в любом регистре, символы все равно будут появляться в верхнем регистре) и дважды введите пароль `master` в поля **Password** (Пароль) и **Confirm Password** (Подтверждение пароля). Щелкните по кнопке **OK**. В список будет добавлена новая учетная запись пользователя.

Для изменения существующей записи (изменять можно любое поле кроме имени пользователя) нужно щелкнуть по кнопке **Edit** (Редактировать) или дважды щелкнуть мышью по нужной строке и в появившемся окне изменить желаемые поля.

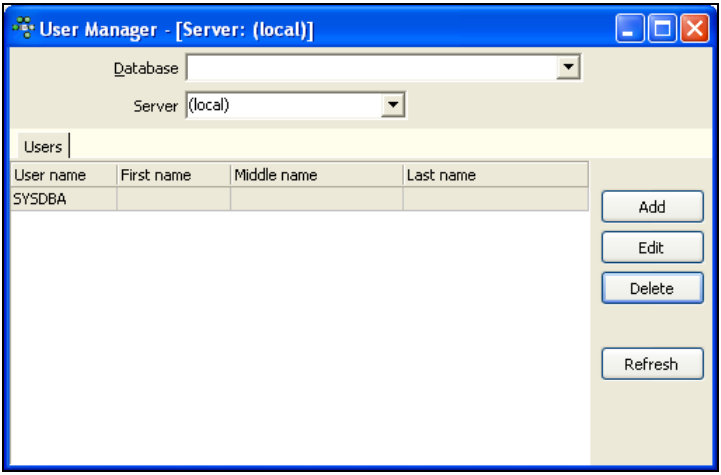


Рис. 2.2. Окно просмотра и изменения списка пользователей в IBExpert

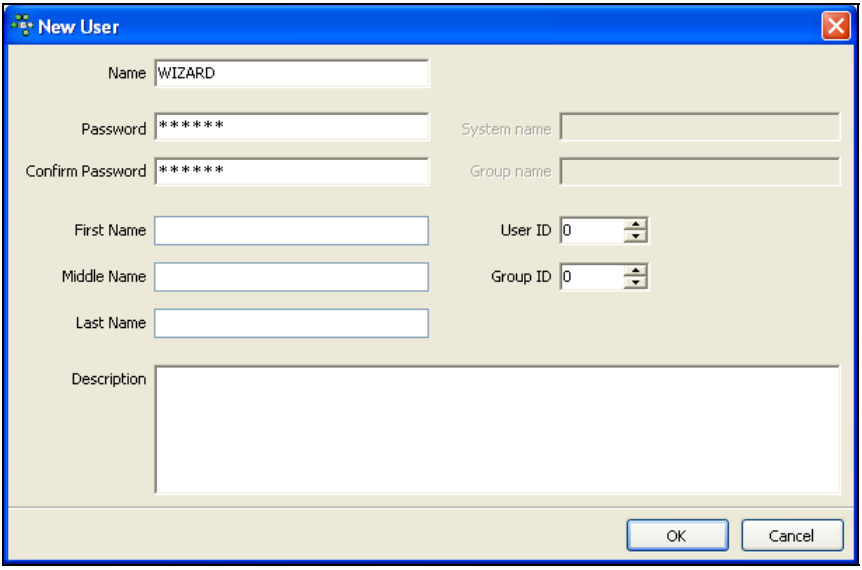


Рис. 2.3. Добавление в список нового пользователя в IBExpert

Для удаления существующей записи щелкните по кнопке **Delete** (Удалить) и в диалоговом окне подтвердите удаление.

Чтобы изменить имя пользователя, вам нужно удалить старую запись и ввести новую с другим именем и теми же или другими характеристиками.

Если вы одновременно работаете в IBExpert и используете утилиту gsec или другую программу для добавления или изменения учетных записей, то для того чтобы увидеть в списке IBExpert новое состояние списка, нужно щелкнуть по кнопке **Refresh** (Обновить).

Как и в случае с утилитой gsec, вы можете в IBExpert просматривать и корректировать учетные записи пользователей и на других компьютерах сети. Для этого в поле **Server** нужно набрать имя компьютера в сети и щелкнуть по кнопке **Connect** (Соединиться). Здесь вы также можете работать и в том случае, когда база данных безопасности располагается на диске с запрещенным доступом с других компьютеров в сети.

Все то же самое можно выполнить и при использовании другой программы графического интерфейса — BlazeTop.

2.2.3. Для особо одаренных. Создание собственной программы для работы с учетными записями пользователей

Используя среду разработки Delphi или C++Builder и компоненты работы с базой данных FIBPlus или стандартные (то есть входящие в комплект поставки Delphi и C++Builder) IBX, вы можете написать собственную программу для работы с учетными записями пользователя. Напишем упрощенный вариант программы в Delphi. Она будет отображать только имена пользователей. Использование визуальных средств в C++Builder такое же, а программный код отличается только синтаксисом.

2.2.3.1. Использование компонентов FIBPlus

Запустите Delphi, создайте новое приложение — меню **File | New | Application** (Файл | Новое | Приложение).

Поменяйте в Инспекторе объектов (Object inspector) имя формы с `Form1` на `FormMain` (свойство `Name`). Мы всегда для главной формы создаваемой программы будем задавать такое имя. Измените текст заголовка на `User Account` (свойство `Caption`). Установите размещение формы (свойство `Position`) по центру экрана — выберите из выпадающего списка свойства значение `poScreenCenter`. По умолчанию форма размещалась бы в том же месте экрана, где она находилась в процессе проектирования, в этом случае значением свойства `Position` было бы `poDesigned`.

С вкладки **Standard** Палитры компонентов поместите на форму панель (компонент `Panel`), очистив в Инспекторе объектов поле `Caption` и сделав выравнивание по верхнему краю (в выпадающем списке свойства `Align` выберите `alTop`)[5]. Это будет панель инструментов, куда вы поместите кнопки быстрого доступа, дублирующие некоторые (в нашем случае — все) элементы главного меню.

Разместите на панели с вкладки **Standard** Палитры компонентов четыре кнопки (`Button`), которые будут выполнять следующие функции:

- **Refresh** (установите это значение в свойстве `Caption` у кнопки) с именем `Name` = `BRefresh` для выполнения обновления списка;
- **Add** (имя `BAdd`) для добавления новой записи;
- **Edit** (`BEdit`) для изменения текущей записи;
- **Delete** (`BDelete`) для удаления текущей записи.

С вкладки **Standard** Палитры компонентов поместите на форму компонент главного меню (`MainMenu`), дважды щелкните мышью по компоненту и в появившемся диалоговом окне создайте следующие элементы меню (рис. 2.4):

- **File** (имя `MFile`) — элемент меню верхнего уровня, который будет виден сразу под заголовком окна; для него создайте дочерние элементы:
  - **Refresh** (`MRefresh`) для выполнения обновления списка, в Инспекторе объектов установите его свойство `ShortCut` в `F5` (можно выбрать из выпадающего списка или набрать текст в поле редактирования); при работающей программе вы можете вызвать этот элемент, просто нажав клавишу <F5>;
  - **Add** (`MAdd`) для добавления новой записи, `ShortCut` установите в `Shift+Ins` (также можно выбрать из списка или набрать вручную); аналогично, вызов соответствующих действий по добавлению записи можно будет осуществить нажатием клавиш <Shift>+<Ins>;
  - **Edit** (`MEdit`) для изменения текущей записи, `ShortCut` = `Enter` (в выпадающем списке отсутствует, нужно набрать вручную);
  - **Delete** (`MDelete`) для удаления текущей записи, `ShortCut` = `Shift+Del`.

---

[5] Имеет смысл регулярно напоминать, что для установки свойств каких-либо компонентов или написания для них обработчиков любых событий нужно предварительно выделить соответствующий компонент на форме. Для этого нужно либо щелкнуть мышью по компоненту, либо в Инспекторе объектов из выпадающего списка в верхней части окна выбрать нужный компонент. Относительно данного конкретного случая — я слишком часто наблюдаю, как студенты, поместив на форму панель, щелкают после этого мышью по форме и устанавливают именно для нее значения этих двух свойств.

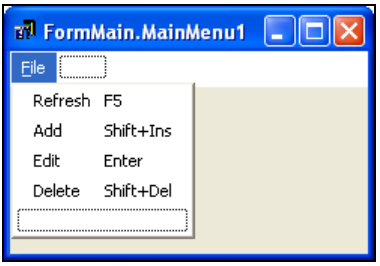


Рис. 2.4. Главное меню собственной программы работы с учетными записями пользователей

Поместите на форму с вкладки **Standard** контекстное меню (`PopupMenu`), дважды щелкните мышью по компоненту и создайте у него два элемента — **Edit** (имя `PEdit`) и **Delete** (`PDelete`).

Таким образом вы создали основные элементы управления программой.

Совет

Здесь подробно рассказывается, какие имена следует присвоить элементам меню и кнопкам. Имена кнопок я начинаю с буквы "B" (от Button), элементов главного меню — с "M", элементов контекстного меню — с "P". Такое внимание к именованию объектов поможет вам в дальнейшем при разглядывании ранее написанных программ не теряться в странных именах, присваиваемых по умолчанию: N1, N2...

Еще одно замечание для любителей Объектного Паскаля (Object Pascal) — этот язык не видит разницы между прописными и строчными буквами. В отличие от человека. Не стесняйтесь использовать в именах ваших объектов и прописные, и строчные буквы. Этим вы сделаете имена еще более понятными.

Замечание

В этой и других программах данной главы все тексты в меню, на кнопках, надписи на формах представлены на английском языке. Это сделано лишь для попыток совместимости с теми существующими программами, которые вы здесь используете. В остальных программах, которые мы с вами напишем в этой книге, мы поступим более патриотично и будем все надписи выполнять на родном русском языке.

С вкладки **Additional** поместите на форму компонент `StringGrid`, в котором и будут размещаться все строки списка пользователей.

Для этого компонента задайте в Инспекторе объектов выравнивание по всей свободной поверхности формы (`Align` = `alClient`; выбирается из выпадающего списка), количество столбцов `ColCount` = `1` (мы будем размещать на сетке только имена пользователей), количество фиксированных столбцов `FixedCols` = `0` (в левой части сетки не будет фиксированных столбцов серого цвета), количество фиксированных строк `Fixed Rows` = `1` (здесь, в фиксиро-

ванной строке, будет располагаться заголовок вашей таблицы). Раскройте свойство `Options`, щелкнув мышью слева от названия свойства по символу +, и установите в `True` значение подсвойства `goRowSelect` (будет отмечаться вся выбранная строка, а не отдельные ячейки сетки). Выберите из выпадающего списка `PopupMenu` имя вашего контекстного меню (`PopupMenu1`). Установите в `2` количество строк `RowCount`, чтобы выводились, по меньшей мере, заголовок и строка пользователя `SYSDBA`.

С вкладки **FIBPlusServices** (надо полагать, вы уже установили компоненты FIBPlus в вашей среде разработки Delphi) поместите на форму компонент `SecurityService`. Установите его имя в `SecurityService1`. В поле `UserName` введите `SYSDBA`.

Теперь ваша форма должна выглядеть приблизительно следующим образом — рис. 2.5

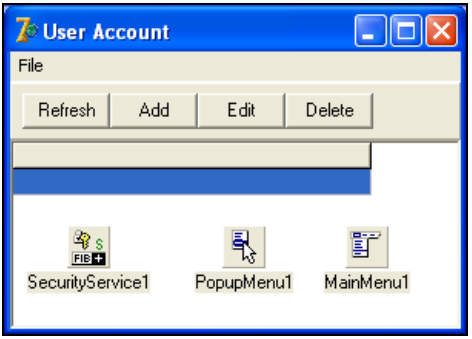


Рис. 2.5. Главная форма программы работы с учетными записями пользователей

Кстати, чтобы в среде Delphi отображались имена невизуальных компонентов, таких как главное и всплывающее меню или компонент `SecurityService`, нужно выбрать в меню **Tools** пункт **Environment Options**, а в появившемся окне **Environment Options** щелкнуть по вкладке **Designer** и установить флажок **Show component captions** (Показывать заголовки компонентов).

Основная задача в разработке этого приложения — написать обработчики различных событий вашей программы.

Вначале создадим процедуру отображения списка пользователей и функцию соединения с базой данных безопасности. В соответствии с требованиями языка Delphi (раньше он назывался Объектным Паскалем) в раздел `private` описания формы нужно поместить прототипы этих функций:

```
private
  procedure ShowUsers;
```

```
  function ConnectToDatabase: Boolean;
  { Private declarations }
```

Функцию соединения с базой данных безопасности следует разместить в разделе реализации сразу после строк:

```
implementation
{$R *.dfm}
```

Сама функция соединения с базой данных безопасности выглядит так, как показано в листинге 2.1 (вы должны ввести вручную *все* указанные ниже строки для функции `ConnectToDatabase` и процедуры `ShowUsers`; при написании же обработчиков различных событий заготовка процедуры будет создаваться автоматически, об этом в очередной раз более подробно мы поговорим несколько позже).

Листинг 2.1. Функция соединения с базой данных

```
function TFormMain.ConnectToDatabase: Boolean;
begin
  Result := True;
  try
    SecurityService1.Active := True;
  except
    Result := False;
  end;
  if Result = False then
    ShowMessage(
'Ошибки при соединении с базой данных безопасности. Работа прекращается')
  else
    SecurityService1.Active := False;
  SecurityService1.LoginPrompt := False;
end;
```

В блоке `try` делается попытка соединиться с базой данных безопасности. При этом автоматически появляется окно запроса на ввод пароля пользователя — это выполняют компоненты FIBPlus. Если возникли ошибки (например, неверный пароль), то выдается сообщение и функция возвращает `False`. При удачном соединении функция возвращает `True`. Свойство `LoginPrompt` компонента `SecurityService1` устанавливается в `False`, чтобы исключить в дальнейшем повторный запрос пароля пользователя при обращении к различным методам этого компонента.

Процедура `ShowUsers` (листинг 2.2) отображает список пользователей в сетке `StringGrid`. Поместите текст этой процедуры сразу после функции `ConnectToDatabase`, также набрав все строки вручную.

Листинг 2.2. Процедура отображения списка пользователей

```
procedure TFormMain.ShowUsers;
var i: Integer;
begin
  SecurityService1.Active := True;
  SecurityService1.DisplayUsers;
  StringGrid1.RowCount := 2;
  StringGrid1.FixedRows := 1;
  for i := 0 to SecurityService1.UserInfoCount – 1 do
  begin
    StringGrid1.Cells[0, i + 1] :=
      SecurityService1.UserInfo[i].UserName;
    StringGrid1.RowCount := StringGrid1.RowCount + 1;
  end;
  StringGrid1.RowCount := StringGrid1.RowCount – 1;
  SecurityService1.Active := False;
end;
```

Для отображения списка пользователей нужно вызвать метод `DisplayUsers` компонента `SecurityService`. В результате выполнения этого метода все пользователи, описанные в базе данных безопасности, помещаются в список `UserInfo`. Количество элементов в списке содержится в свойстве `UserInfoCount`.

Каждая запись списка `UserInfo` является типом `TUserInfo`, который имеет следующую структуру:

```
TUserInfo = class
public
  UserName: string;
  FirstName: string;
  MiddleName: string;
  LastName: string;
  GroupID: Integer;
  UserID: Integer;
end;
```

Мы можем напрямую обращаться к любому элементу учетной записи пользователя, кроме пароля (`Password`).

Теперь нужно написать обработчик события отображения формы (OnShow). Сделайте форму текущим элементом проекта. Поскольку вся форма закрыта другими компонентами, нужно щелкнуть мышью по любому компоненту формы и нажимать клавишу <Esc>, пока не станет текущей сама форма. Разумеется, если форма не вся закрыта другими компонентами, то для выделения формы достаточно щелкнуть по ней мышью.

Другой способ — в Инспекторе объектов сразу под заголовком располагается список всех компонентов формы. Нужно мышью раскрыть этот список и, прокручивая его, выбрать строку формы FormMain.

После того как вы сделали форму текущим элементом, в Инспекторе объектов перейдите на вкладку **Events** (События) и дважды щелкните мышью справа от события OnShow. В редакторе кода появится заготовка для обработчика этого события (листинг 2.3). Система сама присвоит этой процедуре подходящее имя.

Листинг 2.3. Заготовка для обработчика события отображения формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin

end;
```

При этом также автоматически будет создан прототип процедуры в разделе type. Вам нужно только написать несколько строк кода между операторами **begin** и **end** (листинг 2.4). Удивительно, но самая распространенная ошибка начинающих программистов — попытка написать все строки процедуры, включая ее заголовок.

Листинг 2.4. Обработчик события отображения формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin
  StringGrid1.Cells[0,0] := 'User Name';
  if not ConnectToDatabase then
    Application.Terminate;
  ShowUsers;
  StringGrid1.Row := 1;
end;
```

В этом обработчике формируется заголовок сетки отображения списка учетных записей (в режиме проектирования этого сделать нельзя), происходит обращение к функции соединения с базой данных безопасности и при отсутствии ошибок соединения отображается список пользователей. В противном

случае работа программы завершается в результате вызова метода `Terminate` объекта `Application`.

Для создания обработчика щелчка по кнопке обновления (**Refresh**) нужно выделить эту кнопку на форме (сделать ее текущим элементом), щелкнув по ней мышью. Затем перейти к вкладке **Events** в Инспекторе объектов и дважды щелкнуть мышью справа от события `OnClick`. Будет создана заготовка для обработчика. Другой, более простой вариант создания заготовки обработчика события щелчка по этой кнопке состоит в том, что вы просто дважды щелкаете мышью по самой этой кнопке. Имя обработчика будет весьма "мнемоничным" — оно состоит из имени кнопки, за которым следует слово Click (щелчок мышью). Аналогичным образом формируется и имя обработчика щелчка мышью по элементу меню. Разумеется, чтобы получились действительно хорошие имена обработчиков, нужно чтобы имена кнопок и элементов меню также были осмысленными.

Сам обработчик щелчка по кнопке обновления выглядит совсем просто (листинг 2.5).

Листинг 2.5. Процедура обновления списка учетных записей

```
procedure TFormMain.BRefreshClick(Sender: TObject);
begin
  ShowUsers;
end;
```

Далее с этим обработчиком нужно связать соответствующий элемент главного меню. Дважды щелкните мышью по компоненту главного меню (`MainMenu`), в появившемся окне выделите элемент меню **Refresh**, в Инспекторе объектов на вкладке **Events** щелкните по событию `OnClick`, в правой части этой строки щелкните по кнопке со стрелкой, направленной вниз, для просмотра списка соответствующих обработчиков событий и из выпадающего списка выберите строку `BRefreshClick`. "Подходящими" будут те обработчики событий, состав и тип параметров у которых соответствуют нашему обработчику.

Удаление текущего элемента списка осуществляется при щелчке по кнопке удаления и при выборе в главном или контекстном меню элемента удаления. Как и в предыдущем случае, сначала создайте обработчик события щелчка по кнопке удаления (**Delete**) на панели инструментов, а затем свяжите с этим обработчиком соответствующие элементы главного и контекстного меню. Удаление выполняется следующей процедурой — листинг 2.6.

Листинг 2.6. Процедура удаления текущего пользователя

```
procedure TFormMain.BDeleteClick(Sender: TObject);
begin
  if StringGrid1.Cells[0, StringGrid1.Row] = 'SYSDBA' then
  begin
    Application.MessageBox('Вы не можете удалять пользователя SYSDBA!',
      'Error', MB_OK + MB_ICONEXCLAMATION);
    exit;
  end;
  if Application.MessageBox(PAnsiChar('Удалить пользователя ' +
    StringGrid1.Cells[0, StringGrid1.Row] + '?'),
     'Подтверждение', MB_OKCANCEL + MB_ICONQUESTION) = mrOk then
  begin
    SecurityService1.Active := True;
    SecurityService1.UserName :=
      StringGrid1.Cells[0, StringGrid1.Row];
    try
      SecurityService1.DeleteUser;
    except
      ShowMessage('Ошибки при удалении пользователя');
    end;
    SecurityService1.Active := False;
    ShowUsers;
  end;
end;
```

Вначале проверяется, не является ли текущий элемент пользователем `SYSDBA`. Если да, то после выдачи предупреждающего сообщения происходит выход из процедуры. Иначе после подтверждения пользователем удаления текущего элемента в свойство `UserName` компонента `SecurityService` помещается имя пользователя и вызывается метод `DeleteUser` для удаления пользователя с этим именем.

> ***Замечание***
>
> Хочу напомнить, что такая проверка является, с одной стороны, несколько избыточной, потому что пользователя `SYSDBA` система все равно ни при каких обстоятельствах не удалит, не выдавая при этом никаких предупреждающих или диагностических сообщений. С другой стороны, это все же хорошая практика, когда пользователь нашей программы открытым текстом информируется о его попытке выполнить недопустимую операцию. Этого, кстати, не делают другие используемые нами программы.

В методе `MessageBox` первый параметр должен иметь тип данных `PAnsiChar`. Если вы выполняете конкатенацию (объединение) нескольких строк для по-

лучения нужного текста, вам потребуется явное преобразование полученной строки к этому типу. По крайней мере, это нужно для Delphi версии 7.

Для выполнения добавления нового пользователя и изменения данных существующего пользователя нужно создать новую форму. Выберите в главном меню **File | New | Form**. Появится новая форма. Присвойте ей имя `FormAddUser`. Установите свойство `BorderStyle` в `bsDialog` (выберите из выпадающего списка), чтобы форма выглядела как диалоговое окно и пользователь не мог бы изменять ее размеры. Размер формы и расположение размещенных на ней компонентов удовлетворяют наши эстетические потребности, и мы не хотим, чтобы в процессе выполнения программы эти красивые пропорции нарушались неумелыми действиями пользователя программы. Установите размещение формы (свойство `Position`) по центру экрана — выберите из выпадающего списка этого свойства значение `poScreenCenter`. Заголовок формы (свойство `Caption`) можно сейчас не менять — его значение будет динамически изменяться при вызове этой формы в зависимости от того, требуется добавление новой записи или изменение существующей.

Сохраните модуль с именем `UserAdd`.

Разместите на форме три метки (`Label`), три поля редактирования (`Edit`) для ввода имени пользователя, пароля и подтверждения пароля и две кнопки (`Button`) — **OK** и **Cancel**. Для меток в свойстве `Caption` задайте тексты, которые должны располагаться на форме слева от соответствующих полей редактирования. Полям редактирования присвойте соответствующие имена — `UserName`, `Password` и `ConfirmPassword`.

Для кнопки **OK** задайте свойства: `Default` = `True`, чтобы к ней происходило обращение и в том случае, когда пользователь нажмет клавишу <Enter>; из выпадающего списка у свойства `ModalResult` выберите значение `mrOk`, это значение будет возвращаться вызвавшему модулю, если пользователь щелкнет по кнопке мышью или нажмет клавишу <Enter>. Имя компоненту, естественно, присвойте `BOK`.

Для кнопки **Cancel** задайте `Cancel` = `True` (к кнопке будет осуществлено обращение, когда пользователь нажмет клавишу <Esc>), `ModalResult` = `mrCancel`, имя `BCancel`.

Получившаяся форма будет выглядеть так, как показано на рис. 2.6.

В описании формы в разделе `public` опишите переменную:

```
public
    ActionKind: Integer;
```

Рис. 2.6. Форма для добавления/изменения записи пользователя

Вызывающий главный модуль будет присваивать этой переменной значение `0`, если форма используется для добавления новой записи, и `1`, если к ней происходит обращение для изменения существующей записи.

Напишите следующий обработчик события `OnShow` (возникает, когда форма становится видимой, т. е. когда к ней обращается вызывающий модуль)[6]:

Листинг 2.7. Обработчик события отображения формы добавления пользователя

```
procedure TFormAddUser.FormShow(Sender: TObject);
begin
  if ActionKind = 0 then  // Добавление нового пользователя
  begin
    Caption := 'Новый пользователь';
    UserName.Enabled := True;
    UserName.SetFocus;
    BOK.Enabled := False;
  end
  else                    // Редактирование существующего пользователя
  begin
    Caption := 'Редактирование пользователя';
    UserName.Enabled := False;
    Password.SetFocus;
    BOK.Enabled := True;
  end;
end;
```

Если требуется добавление нового пользователя, то поле ввода имени делается доступным, на него устанавливается фокус ввода, а кнопка **OK** делается

[6] Вы не обидитесь, если я опять напомню, что, прежде чем создавать обработчик этого события для формы, сначала мышью нужно щелкнуть по самой форме?

недоступной. Манипуляции с доступностью этой кнопки — тема отдельной песни.

В случае же редактирования существующей записи поле имени делается недоступным для изменения, фокус ввода устанавливается на поле ввода пароля.

Для события `OnChange` (при изменении) для поля редактирования имени пользователя (`UserName`) напишите процедуру (листинг 2.8).

Листинг 2.8. Процедура обработки изменения данных в полях редактирования

```
procedure TFormAddUser.UserNameChange(Sender: TObject);
begin
  if ActionKind = 0 then  // Добавление нового пользователя
    BOK.Enabled := ((Trim(Password.Text) <> '') and
       (Trim(ConfirmPassword.Text) <> '') and
       (Trim(Password.Text) = Trim(ConfirmPassword.Text))) and
       (Trim(UserName.Text) <> '')
  else                    // Редактирование существующего пользователя
    BOK.Enabled := ((Trim(Password.Text) = '') and
       (Trim(ConfirmPassword.Text) = '')) or
       (Trim(Password.Text) = Trim(ConfirmPassword.Text));
end;
```

Здесь как раз и содержится текст той самой песни по изменению доступности кнопки **OK**. Разберитесь с ним самостоятельно. Основная идея состоит в том, что при добавлении нового пользователя непробельные имя и пароль должны быть обязательно заданы, а при изменении сведений о существующем пользователе текст пароля может отсутствовать (тогда существующее значение пароля не будет изменяться). И в обоих случаях значения в поле пароля и в поле подтверждения пароля должны быть одинаковыми. Для удаления пробелов (как начальных, так и конечных) из полей ввода очень часто используется функция `Trim`.

С этой процедурой свяжите также событие `OnChange` для поля пароля (`Password`) и для поля подтверждения пароля (`ConfirmPassword`). Для этого, как вы помните, нужно выбрать соответствующий элемент редактирования (кстати, можно выбрать сразу оба, держа нажатой клавишу <Shift> и щелкая мышью по выделяемым элементам на форме), щелкнуть мышью в Инспекторе объектов на вкладке **Events** по имени события `OnChange` и из выпадающего списка выбрать строку `UserNameChange`.

Вернитесь в главный модуль и напишите код для выполнения добавления нового пользователя (щелчок по кнопке **Add**) (листинг 2.9). Не поленюсь по-

вторить — прежде чем выбирать в Инспекторе объектов событие `OnClick`, выделите на форме соответствующую кнопку.

Листинг 2.9. Добавление нового пользователя

```
procedure TFormMain.BAddClick(Sender: TObject);
begin
  FormAddUser.UserName.Text := '';
  FormAddUser.Password.Text := '';
  FormAddUser.ConfirmPassword.Text := '';
  FormAddUser.ActionKind := 0;
  if FormAddUser.ShowModal = mrOk then
  begin
    SecurityService1.Active   := True;
    SecurityService1.UserName := Trim(FormAddUser.UserName.Text);
    SecurityService1.Password := FormAddUser.Password.Text;
    try
      SecurityService1.AddUser;
    except
      ShowMessage('Ошибки при добавлении пользователя');
    end;
    SecurityService1.Active := False;
    ShowUsers;
  end;
end;
```

Думаю, что код здесь совершенно очевидный. Вначале полям ввода присваиваются пустые значения. Затем происходит вызов формы `FormAddUser` (метод `ShowModal`). Если пользователь вышел из этой формы при щелчке по кнопке **OK**, то возвращаемым значением будет `mrOk`. В этом случае формируются значения полей записи пользователя, добавляется новая запись (метод `AddUser`), и заново отображается весь список. Иначе в процедуре не выполняется никаких действий.

Свяжите с этим обработчиком выбор элемента **Add** главного меню.

Не забудьте связать главный модуль с модулем добавления/изменения учетной записи `UserAdd`. Для этого войдите в меню **File**, выберите **Use Unit** (Использование модулей), в появившемся списке выберите единственный модуль `UserAdd` и щелкните по кнопке **OK**.

Аналогичным образом выполняется изменение данных текущего пользователя (это также обработчик щелчка по кнопке) (листинг 2.10).

Листинг 2.10. Изменение текущего пользователя

```
procedure TFormMain.BEditClick(Sender: TObject);
begin
  FormAddUser.UserName.Text := StringGrid1.Cells[0, StringGrid1.Row];
  FormAddUser.Password.Text := '';
  FormAddUser.ConfirmPassword.Text := '';
  FormAddUser.ActionKind := 1;
  if FormAddUser.ShowModal = mrOk then
  begin
    SecurityService1.Active := True;
    SecurityService1.UserName := FormAddUser.UserName.Text;
    if Trim(FormAddUser.Password.Text) <> '' then
      SecurityService1.Password := FormAddUser.Password.Text;
    try
      SecurityService1.ModifyUser;
    except
      ShowMessage('Ошибки при изменении пользователя');
    end;
    SecurityService1.Active := False;
    ShowUsers;
  end;
end;
```

Для изменения данных пользователя используется метод `ModifyUser` компонента `SecurityService`.

Свяжите с этим обработчиком соответствующие элементы главного и контекстного меню, а также событие двойного щелчка мышью по сетке `StringGrid`. Для этого выделите на форме сетку, в Инспекторе объектов на вкладке **Events** для события `OnDblClick` выберите из выпадающего списка имя соответствующего обработчика.

Запустите вашу программу на выполнение. Вначале появится окно ввода пароля пользователя, где вам нужно ввести пароль пользователя `SYSDBA`, который установлен на вашем компьютере. Просматривайте, изменяйте, добавляйте и удаляйте пользователей из списка.

Текст программы находится в каталоге Chapter02\UserAccount\FIBPlus.

2.2.3.2. Использование компонентов IBX

Такие же результаты можно получить при использовании компонентов IBX (InterBase Express). Однако мне пришлось довольно много повозиться с подключением к базе данных безопасности. Дело в том, что в моей версии

Delphi, если я указываю необходимость вызова стандартного окна подключения, где пользователь задает имя и пароль (свойство `LoginPrompt`), то работа программы завершается со странным сообщением, что, якобы, пользователь отменил операцию, т. е. щелкнул по кнопке **Cancel**, хотя никакое диалоговое окно не появляется. По этой причине программу пришлось усложнить, добавив собственную форму задания имени и пароля.

Скопируйте уже созданный проект в другой каталог. Замените компонент `FIBSecurityService` на `IBSecurityService` из вкладки **InterBase Admin**, присвоив ему имя `SecurityService1`.

Создадим новую форму (меню **File | New | Form**), присвоим ей имя `FormServerLogin` и сохраним модуль с именем `ServerLogin`. Заголовок формы зададим `Server Login`, размещение — по центру экрана (`Position` = `poScreenCenter`), стиль окна — диалог (`BorderStyle` = `bsDialog`). Поместим на форму метки (`Label`), элементы редактирования (`Edit`), присвоив им имена `Login` и `Password`, и две кнопки (`Button`) `OK` и `Cancel`. В поле редактирования имени пользователя внесем имя `SYSDBA` (свойство `Text`). Форма должна иметь следующий вид — рис. 2.7.

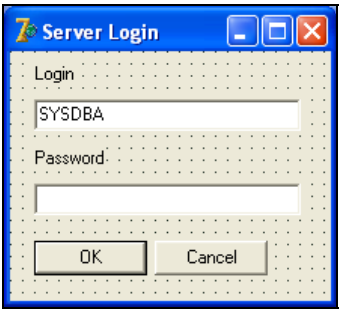


Рис. 2.7. Форма ввода имени и пароля пользователя

В самое начало функции соединения с базой данных `ConnectToDatabase`, текст которой вы так долго набирали вручную, нужно добавить следующий код, осуществляющий обращение к новой форме — листинг 2.11.

Листинг 2.11. Функция соединения с базой данных

```
function TFormMain.ConnectToDatabase: Boolean;
begin
  Result := True;
  if (FormServerLogin.ShowModal <> mrOk) then
    Result := False
  else
  begin
```

```
    SecurityService1.Params.Clear;
    SecurityService1.Params.Add('user_name=sysdba');
    SecurityService1.Params.Add('password=' +
      FormServerLogin.Password.Text);
    SecurityService1.Password := FormServerLogin.Password.Text;
    try
      SecurityService1.Active := True;
    except
      Result := False;
    end;
  end;
  if Result = False then
...
```

Здесь, в компонентах IBX, приходится вручную формировать параметры в свойстве `Params`. Компоненты FIBPlus выполняли это автоматически. Вначале выполняется "очищение" списка (метод `Clear`), затем в список добавляются две строки (метод `Add`), задающие имя пользователя и его пароль.

Запустите программу на выполнение, проверьте правильность выполнения всех функций.

Замечание

В Delphi 7 при работе программы происходит одно неприятное явление. Если в форме ввода имени и пароля нажать клавишу <Enter>, то сразу после этого вызывается форма редактирования пользователя. Более того, если нажать кнопки быстрого доступа к элементам главного меню, то будет вызвана соответствующая функция. Чтобы избежать такого неправильного поведения, нужно в режиме проектирования сделать все элементы меню недоступными (`Enabled = False`), а после удачного соединения с базой данных безопасности перевести их в доступное состояние:

```
MRefresh.Enabled := True;
MAdd.Enabled := True;
MEdit.Enabled := True;
MDelete.Enabled := True;
```

Хочется отметить один приятный момент — в отличие от программы IBExpert у нас происходит редактирование существующей записи не только при щелчке по кнопке редактирования, но также и в следующих случаях: выбор соответствующего элемента в главном или контекстном меню, нажатие клавиши <Enter>, двойной щелчок мышью по сетке. Это тот самый случай, когда разнообразие будет только на пользу нашему клиенту. По-моему, мы молодцы! А вы как считаете?

Текст программы находится в каталоге Chapter02\UserAccount\IBX.

2.3. Создание учебной базы данных

Вся дальнейшая работа будет проводиться с учебной базой данных WORK.FDB, которую вы сейчас создадите. Владельцем этой базы данных будет пользователь `WIZARD`, которого вы только что поместили в список учетных записей.

Для создания базы данных мы опять же можем использовать как утилиту командной строки, входящую в комплект поставки сервера базы данных (на этот раз isql — самая богатая по своим функциональным возможностям утилита), так и программу IBExpert. Кроме этого, мы — конечно же — напишем собственную программу, даже две.

Перед созданием базы данных вам нужно с помощью, например, программы Мой компьютер на диске D: создать новый каталог с именем BestDatabase, в котором будет располагаться ваша база данных. Если на вашем компьютере нет диска D:, вам придется вносить соответствующие изменения в ваши программы.

Для создания базы данных используется оператор SQL `CREATE DATABASE`. Вот его несколько упрощенный синтаксис:

```
CREATE {DATABASE | SCHEMA} '<спецификация файла>'
   USER '<имя пользователя>' PASSWORD '<пароль>'
   [PAGE_SIZE [=] <целое>]
   [DEFAULT CHARACTER SET <набор символов>];
```

Вы можете писать `CREATE DATABASE` или `CREATE SCHEMA` — это синонимы.

> ***Замечание***
>
> В IBExpert некоторых версий использование `CREATE SCHEMA` приводит к ошибке — выдается сообщение, что база данных не открыта (?). Оператор же `CREATE DATABASE` работает нормально.

Далее в апострофах указывается полный путь к файлу и имя файла, например,

```
'D:\BestDatabase\work.fdb'
```

Если вы создаете базу данных не на локальном компьютере, а на сервере, то необходимо в спецификацию файла включить имя сервера, например,

```
'server:D:\BestDatabase\work.fdb'
```

ВНИМАНИЕ!

Операционные системы позволяют в именах файлов и каталогов использовать буквы кириллицы. Однако это может привести к большим неприятностям при создании, изменении, копировании и восстановлении базы данных. Например, в некоторых версиях серверов базы данных при использовании утилиты isql вы вообще не сможете создать базу данных, содержащую в названии файла или в пути к файлу буквы кириллицы. В других программах вы можете получать совершенно загадочные сообщения об ошибках, например, что у вас неверное имя или пароль пользователя. Это связано с особенностями реализации сервера базы данных.

При этом в правильно созданную базу данных вы можете помещать любые символы, включая буквы кириллицы.

Создаваемый файл может находиться и на диске с запрещенным доступом для других компьютеров в локальной сети.

Предложения `USER` и `PASSWORD` задают имя пользователя и его пароль. Пользователь должен существовать в списке учетных записей в базе данных безопасности на том компьютере, где создается база данных. Пользователь с этим именем станет *владельцем* созданной базы данных и будет иметь к ней неограниченные полномочия по просмотру и изменению любых данных и метаданных. Такими же полномочиями к *любой* базе данных на этом компьютере обладает и пользователь `SYSDBA`.

Предложение `PAGE_SIZE` определяет размер страницы базы данных в байтах. Может иметь значения 1024, 2048, 4096, 8192 или 16 384. Если размер страницы не указан, то ему присваивается значение по умолчанию — 4096 (это для Firebird 1.5; для других серверов баз данных значение по умолчанию может быть другим).

Предложение `DEFAULT CHARACTER SET` определяет набор символов для базы данных по умолчанию. Если не указан, то выбирается набор символов `NONE`. Не вдаваясь в подробности, скажем, что во всех случаях следует задать набор символов, отличный от `NONE`, чтобы избежать в дальнейшем ненужных приключений при попытках поместить в строковые столбцы русский текст, да и не только русский. Для использования букв кириллицы и, естественно, всех других символов, имеющихся на клавиатуре нашего компьютера, мы всегда будем задавать набор символов `WIN1251`.

Список наборов символов представлен в *приложении 2*. Там же описывается процесс создания программ, осуществляющих просмотр системных таблиц, содержащих допустимые наборы символов и списки соответствующих порядков сортировки.

Еще одной характеристикой базы данных является ее диалект. В InterBase/Firebird существует два диалекта — 1 и 3. Диалект 1 сохраняется для поддержа-

ния совместимости с предыдущими версиями InterBase (5 и более ранними). Диалект 3 является более удобным, и мы для новой базы данных будем использовать именно его.

Существует еще и диалект 2. Он используется в целях проверки правильности преобразования базы данных диалекта 1 в базу данных диалекта 3. Рассматривать его мы не станем.

Чтобы задать диалект клиента (то есть диалект выполняющейся программы), нужно выполнить оператор:

```
SET SQL DIALECT 3;
```

Этот диалект клиента будет присвоен создаваемой базе данных.

2.3.1. Использование скриптов

Удобнее всего не вводить в диалоговом режиме большое количество операторов для создания и изменения базы данных, а создать скрипт — текстовый файл обычно с расширением sql, который содержит все необходимые операторы SQL для выполнения каких-то действий с базой данных. При необходимости мы всегда сможем внести изменения в существующие скрипты. Кроме того, они также являются и средством документирования нашей программы (программной системы).

Скрипт можно создавать и изменять в обычном Блокноте. Многие программы, например, тот же IBExpert содержат удобные средства создания, корректировки и выполнения скриптов. Программа isql также может выполнять скрипты.

В конце каждого скрипта следует разместить оператор `EXIT`.

2.3.2. Несколько слов о транзакциях

В реляционных базах данных есть такой замечательный механизм, как транзакции. Все операторы работы с базой данных выполняются в контексте (иными словами — под управлением) какой-либо транзакции. Вначале выполнения операторов вы можете явно запустить транзакцию с помощью SQL-оператора `SET TRANSACTION`. Если вы этого не сделаете, то сервер базы данных автоматически запустит транзакцию по умолчанию.

Самым полезным здесь является то, что все операторы, выполнявшиеся в контексте данной транзакции, можно либо подтвердить с помощью оператора `COMMIT` (тогда выполненные действия станут видны другим параллельным процессам), либо отменить посредством SQL-оператора `ROLLBACK`.

О транзакциях мы далее поговорим очень подробно.

2.3.3. Создание базы данных с использованием isql

Вызовите окно командной строки и перейдите в каталог \Bin корневого каталога инсталляции сервера базы данных, например:

```
c:
cd \Program Files\Firebird\Firebird_1_5\Bin
```

Запустите утилиту isql. Появится предложение использовать оператор подключения к базе данных или оператор создания базы данных и подсказка утилиты:

```
Use CONNECT or CREATE DATABASE to specify a database
SQL>
```

Вначале задайте диалект базы данных (точнее диалект клиента, который будет присвоен создаваемой базе данных). Любой оператор для этой утилиты должен заканчиваться точкой с запятой. Если вы введете часть оператора и нажмете клавишу <Enter>, то утилита выдаст подсказку продолжения:

```
CON>
```

Введите:

```
SQL> set sql dialect 3;
```

Теперь можно ввести оператор создания базы данных:

```
SQL> create database 'D:\BestDatabase\work.fdb'
CON> user 'wizard' password 'master'
CON> default character set WIN1251;
```

Имя файла базы данных может иметь любое расширение или вовсе не иметь оного. Для Firebird принято использовать fdb, для InterBase 6.0 и более ранних версий — gdb, для InterBase, начиная с версии 7, — ib.

Можно посмотреть характеристики созданной базы данных. Для этого у утилиты isql существует команда `SHOW DATABASE`. Наберите эту команду в верхнем или нижнем регистре. Не забудьте поставить в конце точку с запятой. Утилита спросит, подтвердить ли текущую транзакцию (имеется в виду транзакция, в контексте которой создавалась база данных):

```
Commit current transaction (y/n)?
```

Ответьте `y` (латинская буква). В результате выполненные изменения будут подтверждены, зафиксированы и станут доступными всем другим процессам.

> ***Замечание***
>
> В каких-то случаях такой запрос не появляется. Не пугайтесь, это не страшно.

После этого на экран будут выведены характеристики вновь созданной вами базы данных приблизительно следующего вида:

```
Database: D:\BestDatabase\work.fdb
        Owner: WIZARD
PAGE_SIZE 4096
Number of DB pages allocated = 146
Sweep interval = 20000
Forced Writes are ON
Transaction — oldest = 3
Transaction — oldest active = 4
Transaction — oldest snapshot = 4
Transaction — Next = 7
Default Character set: WIN1251
```

Здесь мы видим, что владельцем базы данных (`Owner`) является пользователь с именем `WIZARD`, размер страницы (`PAGE_SIZE`) равен 4096, набор символов по умолчанию (`Default Character set`) — `WIN1251`.

> ***Замечание о транзакции***
>
> Любопытным моментом здесь является то, что сразу после выполнения оператора `CREATE DATABASE` на диске создается файл базы данных, имеющий нулевую длину. Независимо от того, будет подтверждена или отменена данная транзакция, после выполнения оператора `COMMIT` или `ROLLBACK` файл базы данных будет создан в полном объеме со всеми его системными данными.

Используя isql, вы можете создавать базу данных и на удаленном компьютере в локальной сети. Для этого нужно задать путь к базе данных, включив и имя компьютера. Например, при использовании протокола Named Pipes, чтобы создать новую базу данных на компьютере, имеющем сетевое имя Computer02, нужно ввести:

```
SQL> create database '\\Computer02\D:\BestDatabase\work.fdb' . . .
```

В случае использования протокола TCP/IP вы должны использовать другой способ записи пути к компьютеру в сети:

```
SQL> create database 'Computer02:D:\BestDatabase\work.fdb' . . .
```

Базу данных можно создавать и на диске, к которому запрещен доступ с других компьютеров в сети.

Для выполнения хранящегося в файле скрипта используется команда `INPUT`.

Создайте в программе Блокнот файл с именем CreateDatabase.sql и поместите в него следующие операторы (на самом деле такой скрипт уже существует, однако вам ничто не мешает создать его самостоятельно):

```
set sql dialect 3;
```

```
create database 'D:\BestDatabase\work.fdb'
 user 'wizard' password 'master'
 default character set WIN1251;
exit;
```

В подсказке утилиты введите:

```
SQL> input CreateDatabase.sql;
```

Такую команду можно вводить, если вы поместили скрипт в тот же каталог, в котором находится утилита isql. В общем случае в команде нужно задать полный путь к файлу скрипта. В результате выполнения этого скрипта будет создана база данных.

Завершите работу с isql, выдав команду:

```
SQL> QUIT;
```

Не забывайте в конце каждого оператора этой утилиты ставить точку с запятой. Собственно, если вы забудете это сделать, то увидите подсказку продолжения. Вам нужно будет ввести только точку с запятой и нажать клавишу <Enter>.

Завершите работу с командной строкой.

2.3.4. Создание базы данных с использованием IBExpert

В программе IBExpert есть два способа создания базы данных: с использованием операторов SQL и с использованием графических инструментальных средств. Рассмотрим оба.

2.3.4.1. Создание базы данных с использованием операторов SQL

Если вы уже создали базу данных с помощью утилиты isql, то перед новым созданием базы данных следует удалить этот файл, используя любую программу Windows.

В программе IBExpert вызовите инструментальное средство, которое называется Script Executive (выполнение скриптов или, если угодно, диспетчер скриптов). Для этого в меню выберите **Tools | Script Executive** или нажмите клавиши <Ctrl>+<F12>. Появится окно **Script Executive** (рис. 2.8).

Для создания базы данных на вкладке **Script** введите те же самые операторы, что вы вводили в диалоге утилиты isql. Ключевые слова SQL программа IBExpert выводит на экран полужирным шрифтом. Любые константы выделяются фоном бирюзового цвета.

Следует напомнить, что существуют версии IBExpert, где для создания базы данных следует вводить только `CREATE DATABASE`, но не `CREATE SCHEMA`.

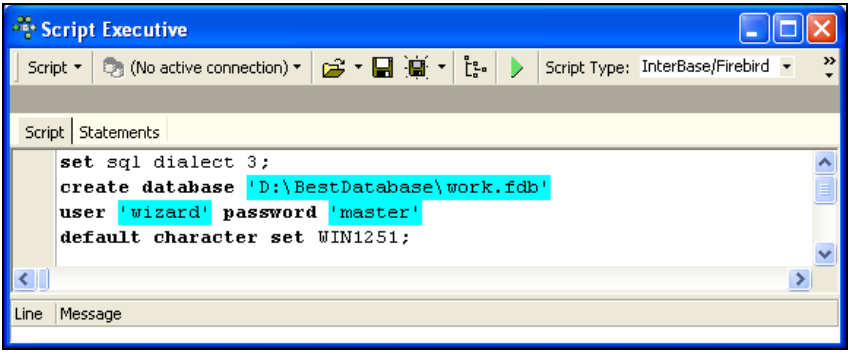


Рис. 2.8. Окно **Script Executive** программы IBExpert для создания, редактирования и выполнения скриптов

Для выполнения введенного оператора щелкните по кнопке **Run** (зеленый треугольник на инструментальной панели окна) или нажмите клавишу <F9> (рис. 2.9).

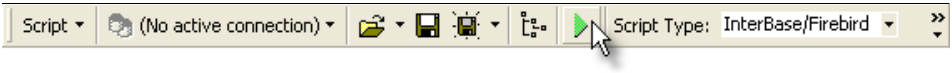


Рис. 2.9. Инструментальная панель окна работы со скриптами

Если вы правильно ввели операторы, и на диске в указанном каталоге не существует данного файла (перед выполнением оператора вы удалили этот файл, созданный утилитой isql), то будет создана новая база данных, и программа выдаст информационное сообщение об успешном выполнении скрипта (рис. 2.10).

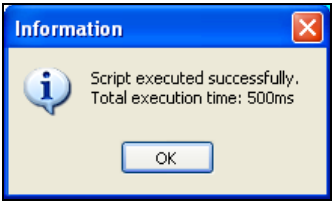


Рис. 2.10. Сообщение об успешном выполнении скрипта

Если же вы получили сообщение об ошибке, то выясните его причину, внесите в операторы изменения и заново выполните скрипт.

Вы можете создать базу данных и на любом компьютере в вашей локальной сети. Для этого в операторе `CREATE DATABASE` в пути к базе данных укажите и имя компьютера, например,

```
create database 'Computer02:D:\BestDatabase\work.fdb'
```

Вы можете создавать базу данных и на диске, к которому запрещен доступ с других компьютеров в сети.

В этом окне также можно не набирать все операторы вручную, а загрузить созданный вами скрипт и выполнить его.

Для особо ленивых

Если вам очень уж не хочется вводить все операторы вручную (хотя я очень рекомендую это сделать, чтобы действительно "потрогать все руками"), вы можете воспользоваться полученным на сайте издательства скриптом CreateDatabase.sql, выполнив его в isql или в IBExpert.

Есть еще скрипт RecreateDatabase.sql, который позволяет вначале удалить существующую базу данных (см. далее), а затем создать ее заново.

Снова о транзакции

IBExpert при создании базы данных автоматически запускает транзакцию и автоматически ее подтверждает, так что от вас никаких действий по управлению транзакцией не требуется. Если вы все же попытаетесь подтвердить транзакцию, вам ненавязчиво сообщат, что подтверждать-то нечего.

2.3.4.2. Создание базы данных с использованием инструмента Create Database

Удалите созданный файл базы данных. Вызовите инструмент Create Database: меню **Database | Create Database** (База данных | Создать базу данных). Появится окно **Create Database** (рис. 2.11).

Из выпадающего списка **Server** выберите **Local**. В поле **Database** введите полный путь к файлу и имя файла базы данных. В поле **Username** введите `WIZARD`, в **Password** — `master`.

Если в процессе создания/удаления учетных записей пользователя вы все-таки удалили пользователя `WIZARD`, вам для создания учебной базы данных необходимо опять создать пользователя `WIZARD` с паролем `master`. Размер страницы (**Page Size**) выберите **4096**.

Из выпадающего списка **Charset** выберите набор символов по умолчанию **WIN1251**. В списке **SQL Dialect** выберите значение **Dialect 3**. Снимите флажок в поле **Register Database After Creating** (не запрашивается регистрация базы данных после создания; регистрацию вы выполните позже).

Не обращайте внимания на поле **Client Library File** (файл клиентской библиотеки) — там уже задано нужное значение.

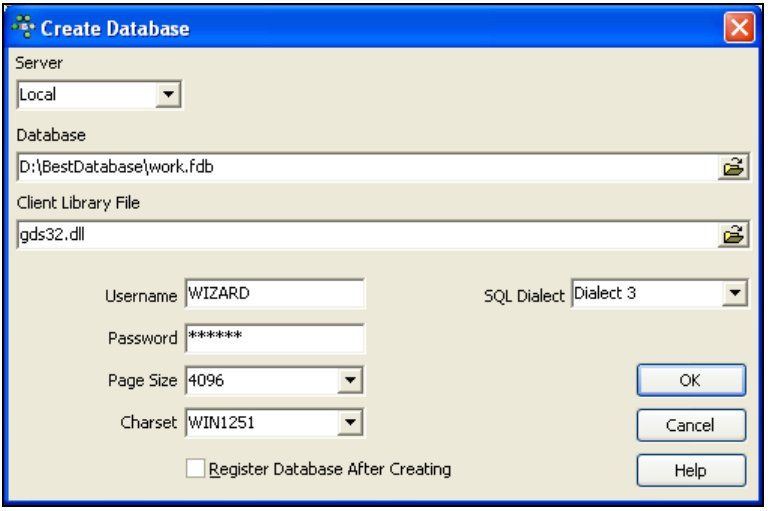


Рис. 2.11. Создание новой базы данных на локальном компьютере

Щелкните по кнопке **OK**. Будет создана база данных, появится информационное окно с соответствующим сообщением (рис. 2.12).

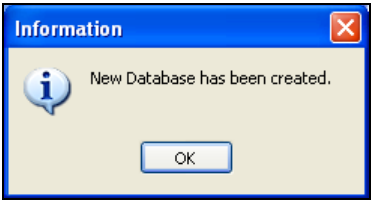


Рис. 2.12. Информационное окно создания новой базы данных

Вы можете создавать базу данных и на любом компьютере в вашей локальной сети. Для этого опять выберите в меню **Database | Create Database**.

В окне **Create Database** (рис. 2.13) в поле **Server** из выпадающего списка выберите **Remote** (удаленный), в поле **Server name** наберите вручную имя нужного компьютера (в используемой мною сети сервер называется незамысловато — Server, что показано на рисунке; вы можете набрать имя любого компьютера в сети, а не только сервера), оставьте протокол (поле **Protocol**) TCP/IP. В поле **Database** введите путь к файлу и имя файла создаваемой базы данных. Путь надо задавать, начиная с имени диска, как если бы файл располагался на локальном компьютере. Нужную строку пути к файлу программа

сформирует надлежащим образом. Базу данных можно создавать и на диске с запрещенным доступом.

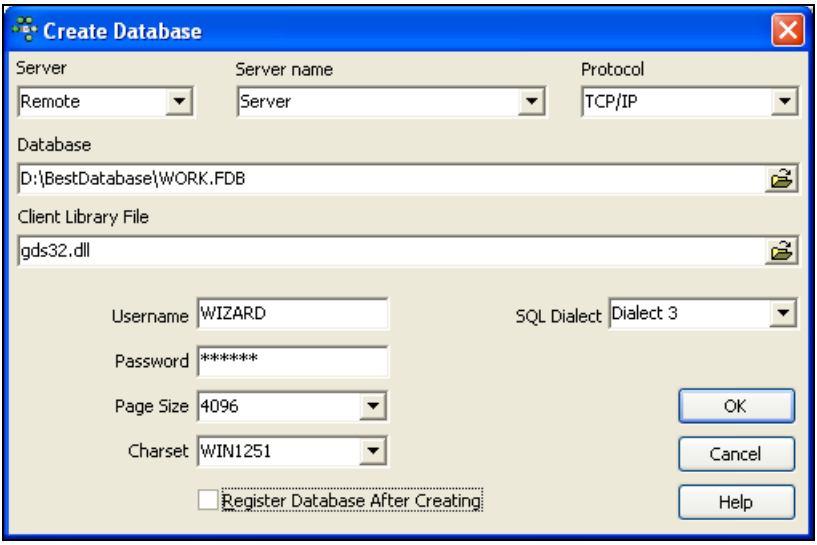


Рис. 2.13. Создание новой базы данных на компьютере в локальной сети

Как и в предыдущем примере, задайте остальные характеристики создаваемой базы данных и щелкните по кнопке **OK**. Будет создана нужная база данных.

> ***Замечание***
>
> Серверы InterBase и Firebird позволяют хранить базу данных в одном или нескольких файлах. Можно сразу создать многофайловую базу данных. А чтобы преобразовать однофайловую базу данных в многофайловую, используется оператор `ALTER DATABASE`. Мы рассмотрели только создание однофайловой базы данных. Если у вас появится потребность использовать многофайловую базу данных, вы легко решите эту задачу, посмотрев документацию.

2.3.5. Для особо одаренных. Собственная программа для создания базы данных

2.3.5.1. Использование компонентов FIBPlus

Для создания базы данных нужно поместить на вновь созданную форму нового приложения (рис. 2.14) шесть меток (`Label`), три поля редактирования (`Edit`) с именами `TDatabase`, `TUserName`, `TPassword`, три выпадающих списка `ComboBox` с именами `CBPageSize`, `CBCharSet`, `SQLDialect` с вкладки **Standard** и две кнопки (`Button`) `BCreateDatabase` и `BExit`, с вкладки **Additional** кнопку

SpeedButton с именем BSelectFile. С вкладки **FIBPlus** поместить компонент TpFIBDatabase, с вкладки **Dialogs** — компонент OpenDialog.

Выделите на форме кнопку BSelectFile, задайте размер 22×22 и загрузите рисунок, щелкнув по кнопке ... справа от имени свойства Glyph и выбрав из набора картинок иконку OpenM. Чуть позже, при написании более серьезных и полезных программ работы с базой данных мы чаще будем использовать не кнопки Button, а именно SpeedButton, т. к. они позволяют хранить иконки.

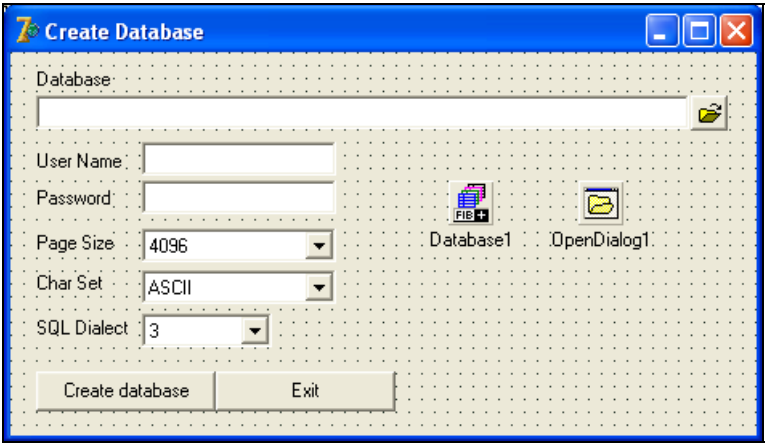


Рис. 2.14. Программа создания базы данных

Для компонента OpenDialog нужно установить значение свойства Filter, которое дает возможность пользователю выбирать отображаемые файлы с конкретным расширением в окне открытия файла.

Выделите на форме компонент OpenDialog, в Инспекторе объектов щелкните справа от свойства Filter по кнопке ... Появится окно **Filter Editor** (рис. 2.15), в котором нужно сформировать три строки, каждая из которых состоит из двух частей. В левой части, **Filter Name**, задается название фильтра, понятное пользователю (поскольку мы с вами с блеском владеем английским, тексты в нашей программе нам совершенно понятны). В правой части, **Filter**, задается значение фильтра, в котором используются обычные шаблонные символы.

Нужно задать файлы Firebird (значение фильтра *.fdb), файлы InterBase (значение фильтра *.gdb) и все файлы (фильтр *.*, т. е. отсутствие фильтра).

В поле TDatabase можно вводить вручную путь к базе данных и имя создаваемой базы данных. Кнопка справа от этого поля позволяет вызвать стандартное диалоговое окно открытия файла. Обработчик щелчка по этой кнопке очень прост (листинг 2.12).

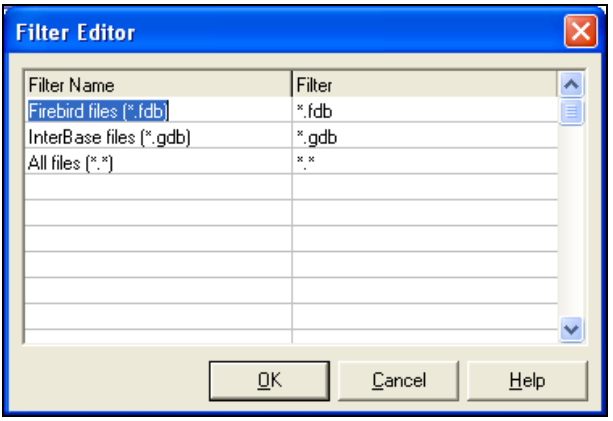


Рис. 2.15. Диалоговое окно задания фильтра файлов

Листинг 2.12. Процедура выбора файла базы данных

```
procedure TFormMain.BSelectFileClick(Sender: TObject);
begin
  if not OpenDialog1.Execute then exit;
  TDatabase.Text := OpenDialog1.FileName;
end;
```

В процедуре вызывается стандартное диалоговое окно Windows открытия файла. Если пользователь выбрал в этом окне нужный файл и щелкнул по кнопке **OK**, то полный путь к файлу и имя файла помещаются в поле редактирования `TDatabase`. Заметьте, однако, что создать файл базы данных, который уже присутствует на диске, нельзя. Если вы выберете существующий файл, то перед созданием базы данных этот файл нужно удалить с диска.

В поля редактирования `TUserName` и `TPassword` нужно при выполнении программы ввести имя пользователя и его пароль соответственно.

Выпадающий список `CBPageSize` содержит все допустимые размеры страницы базы данных — 1024, 2048, 4096, 8192 и 16 384. Чтобы задать эти значения, нужно в Инспекторе объектов щелкнуть по кнопке [...] справа от свойства `Items` (элементы). Появится простенькое окно, в которое нужно поместить эти числа — каждое в отдельной строке.

Выпадающий список `CBCharSet` содержит все наборы символов. Там, по крайней мере, должен существовать наш любимый набор символов `WIN1251`. Формируется таким же образом, как и список размеров страниц, в свойстве `Items`. Полный список наборов символов представлен в *приложении 2*.

Выпадающий список SQLDialect в своем списке Items содержит лишь два значения для диалекта — 1 и 3. Текущим является диалект 3. Это определяется заданием значения 1 для свойства ItemIndex (список нумеруется, начиная с нуля).

Само создание базы данных выполняется при щелчке по кнопке BCreateDatabase (листинг 2.13).

Листинг 2.13. Процедура создания базы данных

```
procedure TFormMain.BCreateDatabaseClick(Sender: TObject);
begin
  Database1.DBParams.Clear;
  Database1.DBParams.Add('USER ''' + TUserName.Text + '''');
  Database1.DBParams.Add('PASSWORD ''' + TPassword.Text + '''');
  Database1.DBParams.Add('PAGE_SIZE = ' + CBPageSize.Text);
  Database1.DBParams.Add('DEFAULT CHARACTER SET ' +
    CBCharSet.Text);
  Database1.DBName := TDatabase.Text;
  Database1.SQLDialect := StrToInt(SQLDialect.Text);
  try
    Database1.CreateDatabase;
  except
    Application.MessageBox('Ошибки при создании базы данных',
      'Ошибки', MB_OK + MB_ICONSTOP);
    exit;
  end;
  Application.MessageBox('База данных создана',
    'Создание', MB_OK + MB_ICONINFORMATION);
  Database1.Close;
end;
```

> ***Замечание***
>
> Если нам в строке символов нужно задать апостроф, то следует записать два подряд идущих апострофа.

В свойстве DBParams типа TStrings компонента Database формируются параметры для создания базы данных (имя пользователя, пароль, размер страницы и набор символов по умолчанию). Путь к базе данных и имя файла помещаются в свойство DBName, диалект базы данных — в свойство SQLDialect.

Создание базы данных осуществляется при вызове метода `CreateDatabase` компонента `Database`. Этот вызов помещается в блок **`try`**, и при возникновении ошибок выдается соответствующее сообщение.

Обратите внимание, что строки в свойстве `DBParams` компонента базы данных соответствуют синтаксису оператора `CREATE DATABASE`. Это допустимо только при создании базы данных. В других, обычных случаях использования этого компонента, когда он ссылается на существующую базу данных, синтаксис помещаемых в этот список строк совершенно другой. Этот синтаксис мы рассмотрим в одной из следующих глав.

Завершение работы происходит при щелчке по кнопке `BExit`. В обработчике этого события записан всего один оператор (листинг 2.14).

Листинг 2.14. Завершение работы программы

```
procedure TFormMain.BExitClick(Sender: TObject);
begin
  Application.Terminate;
end;
```

Текст программы находится в каталоге Chapter02\CreateDatabase\FIBPlus.

2.3.5.2. Использование компонентов IBX

При минимальных изменениях можно выполнить все те же действия при использовании компонентов IBX. Замените в программе компонент `TpFIBDatabase` из набора компонентов FIBPlus на компонент `IBDatabase` из компонентов IBX с вкладки **InterBase**. В тексте программы замените свойство `DBParams` на свойство `Params`, свойство `DBName` на `DatabaseName`. Больше изменений не потребуется. Программа будет выполнять те же действия.

На сайте **www.devrace.com** существует подробное описание процесса разработки программ на Delphi и C++Builder для создания и изменения баз данных при использовании компонентов FIBPlus.

Текст программы находится в каталоге Chapter02\CreateDatabase\IBX.

2.4. Соединение с базой данных

В дальнейшем при работе с созданной базой данных (изменение метаданных, изменение данных или поиск в базе данных) вам необходимо соединяться

с вашей базой данных. Для этого используется оператор CONNECT. Его несколько упрощенный синтаксис:

```
CONNECT '<спецификация файла>'
  USER '<имя пользователя>' PASSWORD '<пароль>';
```

Перед соединением с базой данных необходимо задать диалект клиента (в нашем случае 3), чтобы он соответствовал диалекту базы данных. Кроме этого, нужно задать и активный набор символов текущего соединения с базой данных. Для этого используется оператор SET NAMES, в котором следует задать набор символов WIN1251.

В isql или в программе IBExpert вы можете использовать для соединения с вашей базой данных следующие операторы:

```
set sql dialect 3;
set names WIN1251;
connect 'D:\BestDatabase\work.fdb'
user 'wizard' password 'master';
```

В программе при использовании компонентов FIBPlus для соединения с базой данных нужно либо вызвать метод Open компонента базы данных TpFIBDatabase, либо свойству Connected этого компонента присвоить значение True. Оба варианта равнозначны. Для аналогичного компонента IBX используются те же варианты соединения.

```
Database1.Open;
Database1.Connected := True;
```

2.5. Удаление базы данных

Созданную базу данных можно удалить несколькими способами.

Вы можете просто удалить файл базы данных средствами операционной системы.

> ***Замечание***
>
> В промышленно разработанных системах база данных часто состоит из нескольких файлов. Кроме того, в процессе работы могут создаваться так называемые оперативные (или теневые, shadow) копии базы данных. Все эти файлы могут находиться на различных носителях. При удалении таких сложных конструкций средствами операционной системы мы можем что-то забыть, что-то не учесть. В таком случае лучше удалять базу данных средствами сервера базы данных, учитывающими эту сложность, т. е. с использованием isql, IBExpert или написав собственную программу.

Вы можете удалить базу данных, используя в isql или в IBExpert SQL-оператор DROP DATABASE. Этот оператор удаляет базу данных, с которой вы в

настоящий момент соединены. Например, для удаления базы данных в isql или в программе IBExpert вы можете выполнить следующие операторы:

```
connect 'D:\BestDatabase\work.fdb'
user 'wizard' password 'master';
drop database;
```

Есть скрипт RecreateDatabase.sql, позволяющий удалить нашу базу данных, а затем снова ее создать. Выполнить его можно как в isql, так и в IBExpert. Текст скрипта с удаленными комментариями:

```
SET SQL DIALECT 3;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';
DROP DATABASE;

CREATE DATABASE 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master'
DEFAULT CHARACTER SET WIN1251;
EXIT;
```

Для удаления базы данных в собственной программе при использовании компонентов FIBPlus или IBX нужно при открытой базе данных выполнить один-единственный оператор (точнее метод компонента `Database`):

```
Database1.DropDatabase;
```

Здесь мы с вами не станем создавать программу удаления баз данных. Вы это прекрасно сможете сделать самостоятельно. В такой программе следует соединиться с выбираемой базой данных (например, при использовании стандартного диалога открытия файла Windows), задать имя пользователя и его пароль (пользователем может быть как владелец базы данных, так и пользователь `SYSDBA`) и после подтверждения пользователем (на этот раз тем, который выполняет эту программу) необходимости удаления — удалить базу данных.

Внешний вид формы такой программы будет несколько упрощенным вариантом программы создания базы данных. Потренируйтесь, создайте программу удаления.

Вообще говоря, будьте особенно осторожны с этим средством — как бы вам в порыве экспериментаторства не удалить базу, содержащую важные данные, с таким трудом собранные и записанные.

2.6. Регистрация базы данных в IBExpert

Для выполнения в дальнейшем комфортной работы с базой данных с использованием возможностей программы IBExpert следует зарегистрировать базу данных в этой программе. Если в дальнейшем вам не понадобится любая уже зарегистрированная база данных, вы всегда сможете отменить регистрацию.

Выберите в меню **Database | Register Database** (База данных | Регистрация базы данных). Появится окно выбора типа базы данных (рис. 2.16)

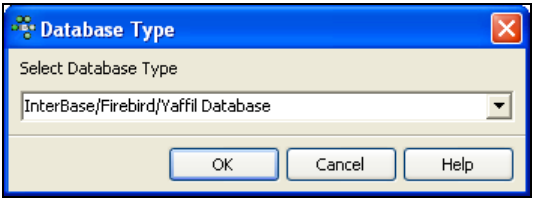


Рис. 2.16. Окно выбора типа базы данных в IBExpert

В этом окне из выпадающего списка **Select Database Type** нужно выбрать строку **InterBase/Firebird/Yaffil Database** (ее можно и не выбирать — она будет текущим значением этого списка) и щелкнуть по кнопке **OK**. Появится окно регистрации (рис. 2.17), в котором нужно будет ввести все необходимые характеристики регистрируемой базы данных.

В выпадающем списке **Server** нужно выбрать **Local** (или **Remote**, если вы регистрируете базу данных, находящуюся на сетевом компьютере), в списке **Server Version** — имя используемого сервера базы данных, в нашем случае здесь указано **Firebird 1.5**. Если вы регистрируете базу данных, располагающуюся на сетевом компьютере, то в поле **Server Name** вам нужно вручную набрать сетевое имя этого компьютера — в выпадающем списке имена доступных в сети компьютеров не появляются. В поле **Database File** нужно ввести полный путь к *существующему* файлу базы данных. Если этот файл находится на локальном компьютере или располагается в сети на диске с разрешенным доступом, вы можете воспользоваться кнопкой обзора, находящейся в правой части этого поля (рис. 2.18). Иначе придется вводить весь путь вручную.

Далее нужно ввести имя пользователя (**User Name**): `WIZARD`, пароль (**Password**): `master`, из выпадающего списка **Charset** выбрать набор символов **WIN1251** и из выпадающего списка **Font Characters Set** выбрать **RUSSIAN_CHARSET**.

Вы можете проверить правильность заданных характеристик базы данных, соединившись с базой данных. Для этого нужно щелкнуть по кнопке **Test**

Connect (проверка соединения). Если все было введено правильно, проверка будет успешной. Такая проверка в особенности имеет смысл в том случае, когда база данных располагается на каком-либо компьютере в сети.

Кстати, вы можете промоделировать сетевое подключение и на локальном компьютере. Для этого нужно установить **Server** в значение **Remote** (удаленный) и в поле **Server Name** набрать `localhost`.

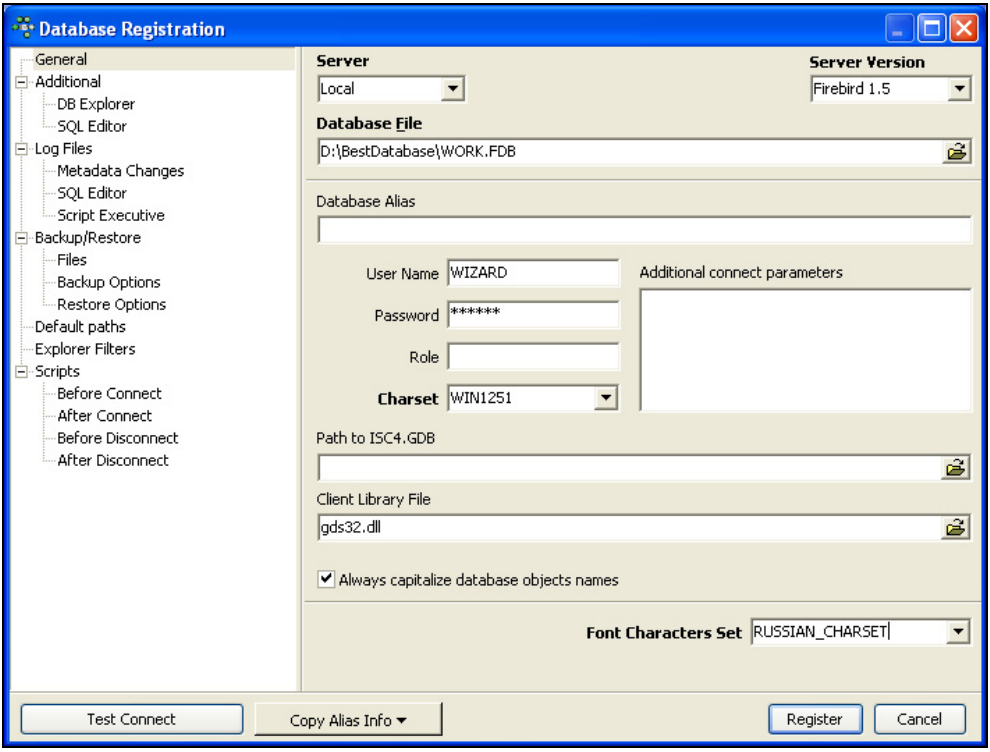


Рис. 2.17. Окно регистрации базы данных

Рис. 2.18. Кнопка обзора для выбора файла базы данных

После щелчка по кнопке **Register** (регистрировать) база данных будет зарегистрирована в программе. В дальнейшем вы сможете легко с ней соединиться и выполнять всевозможные операции изменения данных и метаданных, выборки, копирования и восстановления данных.

2.7. Копирование и восстановление базы данных

Базу данных рекомендуется регулярно копировать (back up) и восстанавливать (restore). Для этих целей может использоваться как утилита командной строки gbak, так и программа IBExpert. Разумеется, мы также напишем свои программы копирования и восстановления баз данных.

Базу данных может копировать только владелец базы данных или пользователь `SYSDBA`. Существуют ранние версии серверов баз данных, которые ошибочно позволяют выполнять копирование *любому* зарегистрированному пользователю.

Восстанавливать базу данных с резервной копии может любой пользователь, однако только владелец базы данных или пользователь `SYSDBA` могут восстанавливать копию поверх существующей базы данных.

2.7.1. Копирование и восстановление базы данных утилитой командной строки

Копирование и восстановление баз данных выполняет утилита командной строки gbak, входящая в комплект поставки серверов баз данных InterBase и Firebird.

При восстановлении базы данных с резервной копии можно поменять некоторые ее характеристики, в частности, размер страницы базы данных.

2.7.1.1. Создание резервной копии

Для выполнения копирования базы данных утилитой gbak нужно вызвать окно командной строки и перейти в каталог \Bin корневого каталога инсталляции сервера базы данных, например:

```
c:
cd \Program Files\Firebird\Firebird_1_5\Bin
```

Запустите утилиту gbak, введя в строке подсказки командной строки:

```
gbak -backup_database -verbose -user wizard -password master
d:\BestDatabase\work.fdb d:\BestDatabase\work.fbk
```

Переключатели утилиты для создания резервной копии представлены в табл. 2.3.

> ***ВНИМАНИЕ!***
>
> В документации по InterBase 6 неверно указаны сокращения для некоторых переключателей этой утилиты.

Таблица 2.3. *Переключатели утилиты gbak, используемые при создании резервной копии*

Переключатель	Назначение
`-b[ackup_database]`	Задает операцию резервного копирования базы данных
`-pas[sword]`	Пароль пользователя
`-user`	Имя пользователя
`-v[erbose]`	Задает отображение утилитой протокола выполняемых ею действий
`-se[rvice]`	Копирует базу данных на ту машину в локальной сети, где размещается база данных. При этом используется Service Manager

Последние два аргумента в нашем примере командной строки задают, соответственно, путь к файлу базы данных и путь к файлу резервной копии. Если файл копии уже существует, он будет заменен новым.

Нужно напомнить, что в именах файлов базы данных и копий не следует использовать буквы кириллицы.

2.7.1.2. Восстановление базы данных с резервной копии

Для восстановления базы данных запустите утилиту gbak:

```
gbak -create_database -verbose -user wizard -password master
d:\BestDatabase\work.fbk d:\BestDatabase\work2.fdb
```

Здесь мы восстанавливаем базу данных с резервной копии, задавая ей другое имя — WORK2.FDB. Чтобы перезаписать существующую базу данных, нужно первым переключателем указать `-replace_database`.

Переключатели утилиты для восстановления базы данных с резервной копии представлены в табл. 2.4.

Таблица 2.4. *Переключатели утилиты gbak, используемые при восстановлении базы данных с резервной копии*

Переключатель	Назначение
`-c[reate_database]`	Задает операцию восстановления базы данных во вновь создаваемый файл
`-r[eplace_database]`	Задает операцию восстановления базы данных во вновь создаваемый файл или при перезаписи существующего файла
`-pas[sword]`	Пароль пользователя
`-user`	Имя пользователя
`-v[erbose]`	Задает отображение утилитой протокола выполняемых ею действий
`-se[rvice]`	Создает восстанавливаемую базу данных на той машине в локальной сети, где размещается резервная копия. При этом используется Service Manager

Последние два аргумента в командной строке задают, соответственно, путь к файлу резервной копии и путь к файлу базы данных. Если выполняется восстановление базы данных на другом компьютере в сети, то файл резервной копии должен размещаться только на диске с разрешенным доступом.

Если копия базы данных находится на другой машине в локальной сети, то можно использовать переключатель `-service`, после которого указать имя Service Manager. При использовании протокола TCP/IP для восстановления копии на той же машине в сети можно задать следующее:

```
gbak -c -verbose -user wizard -password master -service
Server:service_mgr d:\BestDatabase\work.fbk d:\BestDatabase\work.fdb
```

Здесь `Server` — имя другой машины в сети, где располагается копия и куда нужно восстановить базу данных.

Использование Service Manager является единственным способом создания копии базы данных на диске с запрещенным доступом с других компьютеров в сети средствами isql. В IBExpert все гораздо проще.

2.7.2. Копирование и восстановление базы данных в IBExpert

2.7.2.1. Создание резервной копии

Для выполнения копирования базы данных запустите IBExpert. База данных должна быть зарегистрирована в программе. В окне **Database Explorer** щелкните мышью по строке вашей базы данных (рис. 2.19).

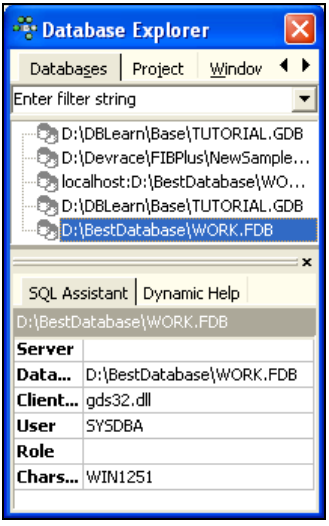


Рис. 2.19. Окно **Database Explorer**. Выбор базы данных для копирования

В меню **Services** выберите **Backup Database**. Появится окно **Database Backup**, в котором на вкладке **Backup Files** уже установлены некоторые необходимые значения (рис. 2.20).

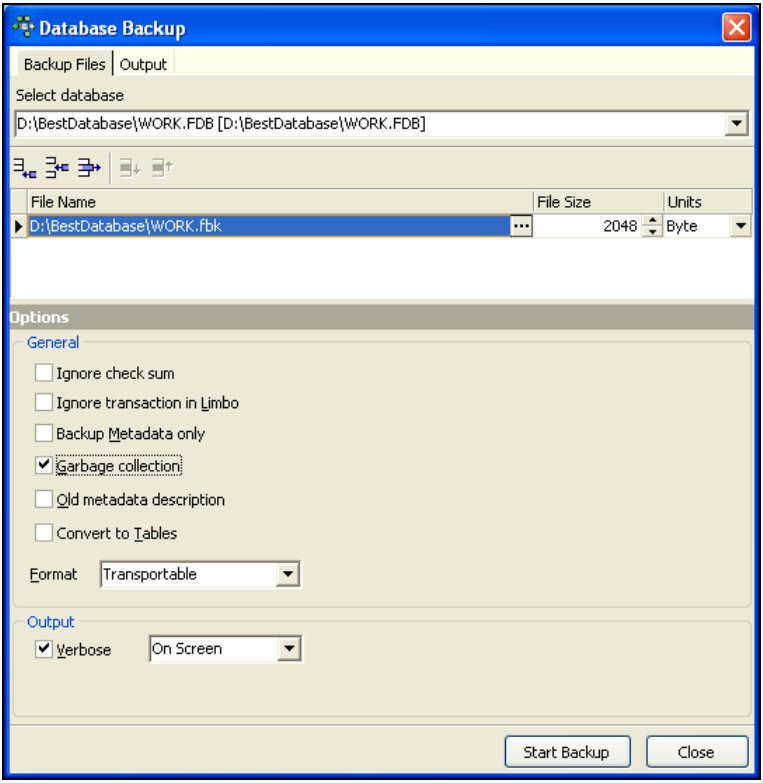


Рис. 2.20. Окно **Database Backup** программы IBExpert

Установите флажок **Garbage collection** (сборка мусора). В результате при выполнении копирования будут удаляться устаревшие записи, оставшиеся в базе данных при обновлении данных — удалении и изменении. Отметьте также флажок **Verbose**, чтобы на экран выводились информационные сообщения о процессе копирования. Из выпадающего списка выберите **On Screen**, чтобы сообщения выводились именно на экран, а не в файл.

Щелкните по кнопке **Start Backup**. Текущей станет другая вкладка — **Output**, на которой будут отображены все сообщения копирования.

2.7.2.2. Восстановление базы данных с резервной копии

Для восстановления базы данных в меню **Services** выберите **Restore Database**. Появится окно **Database Restore** (рис. 2.21). На вкладке **Files** уже установлены все необходимые значения. В поле **Restore into** (восстанавливать в)

указано **Existing database**, что означает, что копия будет заменять существующий файл базы данных.

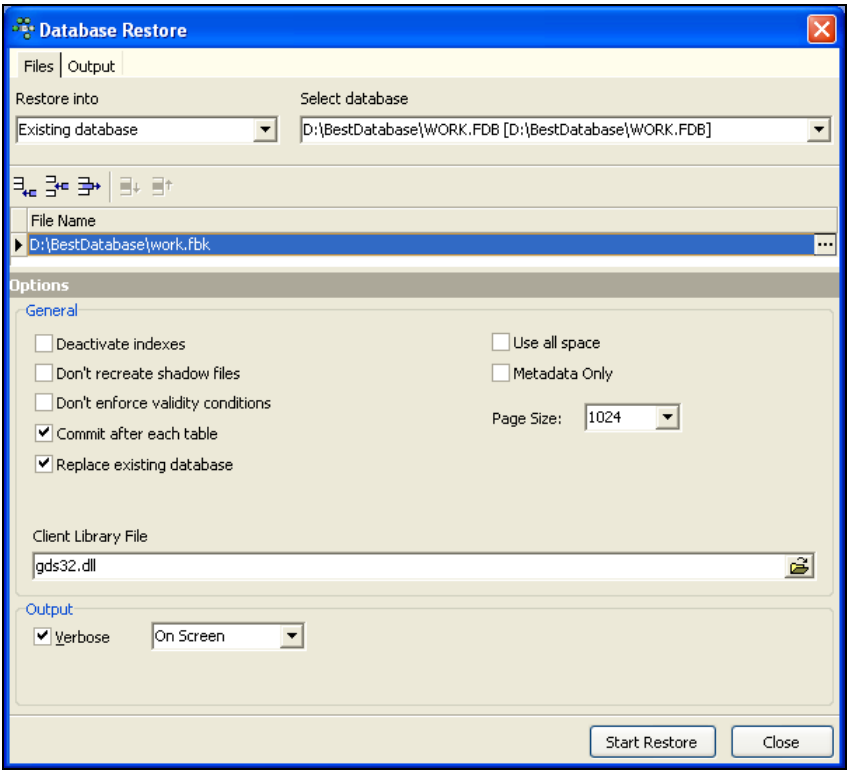


Рис. 2.21. Окно **Database Restore** программы IBExpert

Щелкните по строке **File Name**. В правой части поля появится кнопка с тремя точками. Щелкните по этой кнопке и в диалоговом окне открытия файла выберите файл копии. Отметьте флажком поле **Replace existing database** (заменять существующую базу данных). На всякий случай отметьте также поле **Commit after each table** (подтверждать транзакцию после восстановления каждой таблицы). Вы можете также изменить и размер страницы базы данных, выбрав его из выпадающего списка **Page Size**.

Как и в предыдущем случае, отметьте флажок **Verbose** и из выпадающего списка выберите **On Screen**, чтобы на экран выводились информационные сообщения о процессе восстановления.

Щелкните по кнопке **Start Restore**. Появится окно ввода имени и пароля пользователя. Вы можете выполнить восстановление в качестве пользователя SYSDBA, однако в этом случае именно пользователь SYSBDA станет владельцем

этой базы, что, откровенно говоря, нам не очень нужно. Восстанавливать базу данных в нашем случае нужно от имени пользователя `WIZARD`. База данных будет восстановлена с резервной копии.

Если вы хотите восстановить резервную копию в базу данных с другим именем, то в выпадающем списке **Restore into** нужно выбрать **New database**, в поле **Database File** ввести полный путь к новому (или существующему) файлу базы данных, в поле **File Name** ввести путь к файлу резервной копии или воспользоваться кнопкой обзора для выбора этого файла.

2.7.3. Для особо одаренных. Собственная программа копирования базы данных

2.7.3.1. Использование компонентов FIBPlus

В Delphi создайте новый проект. Поместите на форму одну панель, очистив поле `Caption` и выровняв ее по верху (`Align` = `alTop`). Измените соответствующим образом высоту панели, чтобы на ней могли с комфортом разместиться дочерние компоненты. На эту панель (именно на панель, а не на форму!) поместите несколько компонентов `Edit` с именами `TUserName`, `TPassword`, `TDatabase` и `TBackup`, несколько меток `Label` и две кнопки `SpeedButton`. Установите размеры кнопок 22×22, поместите на каждую из них иконку с именем OpenM из каталога картинок, используя свойство компонентов `Glyph`.

Поместите на форму еще одну панель, очистив свойство `Caption` и выровняв форму по нижнему краю (`Align` = `alBottom`). На эту панель поместите пару кнопок `Button` с текстами `Backup` и `Exit`.

На форму поместите компонент `Memo`, выровняв его по всей поверхности формы (`Align` = `alClient`). Здесь будут отображаться информационные сообщения, выдаваемые в процессе копирования. С вкладки **Dialog** поместите в любое место формы компонент `OpenDialog`, с вкладки **FIBPlus** — компонент `ErrorHandler`, а с вкладки **FIBPlusServices** — компонент `BackupService`.

Форма должна выглядеть приблизительно следующим образом — рис. 2.22.

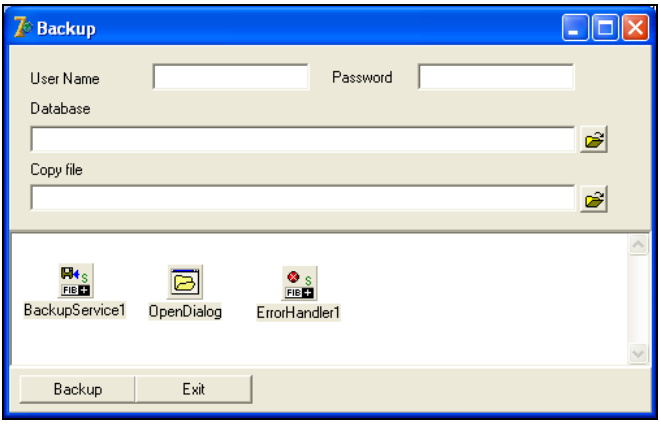


Рис. 2.22. Программа копирования базы данных

В поле `Database` пользователь должен ввести полный путь к базе данных и имя файла базы данных. Щелчок по кнопке `BOpenDatabase` справа от этого поля позволяет в стандартном диалоговом окне открытия файла выбрать файл базы данных. Обработчик щелчка по этой кнопке приведен в листинге 2.15.

Листинг 2.15. Открытие файла базы данных

```
procedure TFormMain.BOpenDatabaseClick(Sender: TObject);
begin
  if not (OpenDialog.Execute) then exit;
  TDatabase.Text := OpenDialog.FileName;
end;
```

Аналогичным образом формируется путь к файлу копии в поле `Copy file` как при ручном вводе пользователем пути и имени файла, так и в результате щелчка по кнопке `BOpenBackup`.

Здесь с целью экономии (времени, размера программы, сил), а может быть из лености для выбора и файла базы данных, и файла копии используется при обращении к стандартному диалогу открытия файла один компонент `OpenDialog` с единственным фильтром — все файлы (*.*). В хорошо спроектированной программе следовало бы использовать два таких компонента с различными вариантами фильтров.

Компонент `ErrorHandler` позволяет выполнять централизованную обработку ошибок базы данных. Для его использования необходимо в оператор **`uses`** вручную добавить модуль FIB. В нашем случае обработчик ошибок только лишь выдает характеристику возникшей ошибки — код и текст (естественно,

на английском языке). Дважды щелкните мышью по компоненту и создайте следующий простейший обработчик — листинг 2.16.

Листинг 2.16. Обработчик ошибок базы данных

```
procedure TFormMain.ErrorHandler1FIBErrorEvent(Sender: TObject;
  ErrorValue: EFIBError; KindIBError: TKindIBError; var DoRaise: Boo-
lean);
begin
  Application.MessageBox(PAnsiChar('Ошибки при копировании' + #10#13 +
    'IBErrorCode: ' + IntToStr(ErrorValue.IBErrorCode) + #10#13 +
    'IBMessage: ' + ErrorValue.IBMessage),
      'Ошибка', MB_OK + MB_ICONSTOP);
end;
```

Основная процедура выполнения копирования приведена в листинге 2.17.

Листинг 2.17. Процедура выполнения резервного копирования

```
procedure TFormMain.BBackupClick(Sender: TObject);
var OldCursor: TCursor;
begin
  OldCursor := Screen.Cursor;
  Screen.Cursor := crHourGlass;
  BackupService1.DatabaseName := TDatabase.Text;
  BackupService1.Params.Clear;
  BackupService1.Params.Add('user_name=' + TUserName.Text);
  BackupService1.Params.Add('password=' + TPassword.Text);
  BackupService1.BackupFile.Clear;
  BackupService1.BackupFile.Add(TBackup.Text);
  Memo1.Clear;
  try
    BackupService1.Active := True;
    BackupService1.ServiceStart;
  except
    BackupService1.Active := False;
    Screen.Cursor := OldCursor;
    exit;
  end;
  BExit.Enabled := False;
  BBackup.Enabled := False;
  Memo1.Lines.Add(
    '==================== Начало копирования ====================');
  while not (BackupService1.Eof) do
```

```
    Memo1.Lines.Add(BackupService1.GetNextLine);
  BackupService1.Active := False;
  BExit.Enabled := True;
  BBackup.Enabled := True;
  Memo1.Lines.Add(
    '=================== Завершение копирования ================');
  Screen.Cursor := OldCursor;
end;
```

Сюда мы добавили еще одну "украшалку". В локальной переменной `OldCursor` типа `TCursor` сохраняется текущее значение курсора — скорее всего, обычная стрелка. Затем курсору присваивается значение `crHourGlass`, он примет вид песочных часов. По окончании процесса копирования или при возникновении ошибок восстанавливается первоначальное значение курсора.

После этого формируются необходимые значения компонента `BackupService`. В свойство `DatabaseName` помещается имя базы данных, в свойстве `Params` создаются две строки, описывающие имя пользователя и его пароль; эти значения будут использованы при подключении к базе данных.

Поскольку сервис позволяет создавать многофайловые копии, его свойство `BackupFile` имеет тип `TStrings` и позволяет хранить произвольное количество строк, описывающих файлы копии. В нашем упрощенном варианте мы помещаем сюда лишь одну строку, описывающую единственный файл копии. Для этого нужно вначале очистить список, используя метод `Clear`, а затем добавить имя файла копии, используя метод `Add`.

Сервис переводится в активное состояние. Сам процесс копирования начинается при вызове метода `ServiceStart`. Если возникли ошибки, то наш обработчик ошибок выдаст сообщение, и процесс прекращается.

В поле `Memo` помещаются информационные строки о копировании. Очередную информационную строку мы получаем, используя метод `GetNextLine`. Процесс продолжается до тех пор, пока свойство `Eof` (конец данных) компонента `BackupService` не примет значение `True`.

Текст программы находится в каталоге Chapter02\Backup\FIBPlus.

2.7.3.2. Использование компонентов IBX

Программу довольно просто переделать под компоненты IBX. Вместо компонента `TpFIBBackupService` поместите на форму с вкладки **InterBase Admin**

компонент FIBBackupService. Вся работа по копированию выполняется точно так же, однако придется убрать обработчик ошибок и компонент ErrorHandler, поскольку в IBX нет соответствующего аналога. Добавьте в предложение **uses** модуль DB. Для выполнения обработки ошибок копирования блок **try...except** следует изменить следующим образом — листинг 2.18.

Листинг 2.18. Обработка ошибок соединения с базой данных в компонентах IBX

```
try
  BackupService1.Active := True;
  BackupService1.ServiceStart;
except
  on E: EDatabaseError do
  begin
    BackupService1.Active := False;
    Screen.Cursor := OldCursor;
    Application.MessageBox(PAnsiChar(
      'Ошибки при копировании базы данных'
      + #10#13 + E.Message), 'Ошибки', MB_OK + MB_ICONSTOP);
    exit;
  end;
end;
```

Здесь в случае ошибки (исключение типа EDatabaseError) с помощью объекта E выдается текст сообщения об этой ошибке.

Более полную версию программ копирования базы данных для C++Builder и Delphi при использовании компонентов FIBPlus см. на сайте **www.devrace.com**.

Текст программы находится в каталоге Chapter02\Backup\IBX.

2.7.4. Для особо одаренных. Собственная программа восстановления базы данных

2.7.4.1. Использование компонентов FIBPlus

Программа восстановления мало чем отличается от программы копирования — только выполнением собственно копирования. Создайте новое приложение. Разместите на форме компоненты, как показано на рис. 2.23. Они та-

кие же, как и в программе копирования, с теми же характеристиками, за тем исключением, что вместо компонента копирования `BackupService` используется компонент восстановления `RestoreService`.

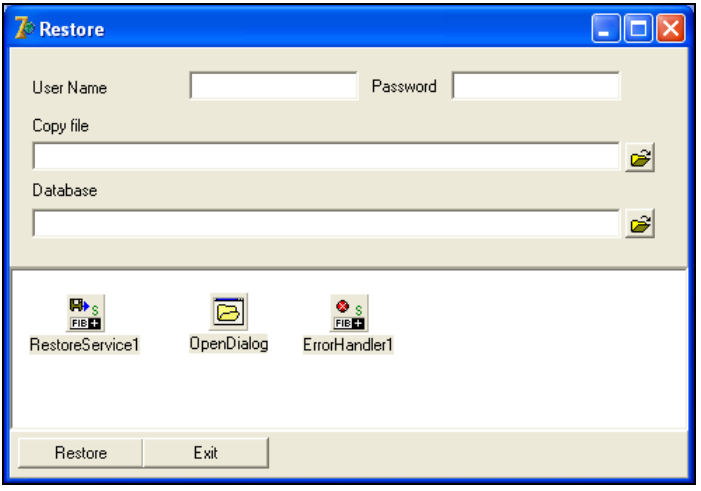


Рис. 2.23. Программа восстановления базы данных

Основная работа выполняется в обработчике события щелчка по кнопке `Restore` (листинг 2.19).

Листинг 2.19. Процедура восстановления базы данных с резервной копии

```
procedure TFormMain.BRestoreClick(Sender: TObject);
var OldCursor: TCursor;
begin
  OldCursor := Screen.Cursor;
  Screen.Cursor := crHourGlass;
  RestoreService1.DatabaseName.Clear;
  RestoreService1.DatabaseName.Add(TDatabase.Text);
  RestoreService1.Params.Clear;
  RestoreService1.Params.Add('user_name=' + TUserName.Text);
  RestoreService1.Params.Add('password=' + TPassword.Text);
  RestoreService1.BackupFile.Clear;
  RestoreService1.BackupFile.Add(TCopy.Text);
  Memo1.Clear;
  try
    RestoreService1.Active := True;
    RestoreService1.ServiceStart;
```

```
  except
    RestoreService1.Active := False;
    Screen.Cursor := OldCursor;
    exit;
  end;
  Memo1.Lines.Add(
    '============== Начало восстановления ================');
  while not (RestoreService1.Eof) do
    Memo1.Lines.Add(RestoreService1.GetNextLine);
  RestoreService1.Active := False;
  BExit.Enabled := True;
  BRestore.Enabled := True;
  Memo1.Lines.Add(
    '============== Завершение восстановления =============');
  Screen.Cursor := OldCursor;
end;
```

Поскольку восстановление возможно и в многофайловую базу данных, в компоненте `RestoreService` свойство `DatabaseName` также имеет тип `TStrings`. В остальном выполнение восстановления очень похоже на копирование.

В хорошей, "правильной" программе восстановления также следовало бы использовать не один, а два компонента `OpenDialog` — один для выбора файла базы данных, а другой для выбора файла копии.

Текст программы находится в каталоге Chapter02\Restore\FIBPlus.

2.7.4.2. Использование компонентов IBX

Здесь также довольно легко можно переделать программу для использования с компонентами IBX. Уберите с формы `RestoreService` и `ErrorHandler`, поместите компонент `IBRestoreService`. Добавьте в предложение **`uses`** модуль DB. Блок **`try...except`** измените следующим образом — листинг 2.20.

Листинг 2.20. Обработка ошибок восстановления базы данных в компонентах IBX

```
try
  RestoreService1.Active := True;
  RestoreService1.ServiceStart;
except
```

```
  RestoreService1.Active := False;
  Screen.Cursor := OldCursor;
  on E: EDatabaseError do
  begin
    RestoreService1.Active := False;
    Screen.Cursor := OldCursor;
    Application.MessageBox(PAnsiChar(
      'Ошибки при восстановлении базы данных'
      + #10#13 + E.Message), 'Ошибки', MB_OK + MB_ICONSTOP);
    exit;
  end;
end;
```

Текст программы находится в каталоге Chapter02\Restore\IBX.

Что там за перевалом?

Мы научились создавать и изменять учетные записи пользователя с помощью утилиты gsec и программы графического интерфейса IBExpert. Можем работать с базой данных безопасности локального компьютера или любого компьютера в сети.

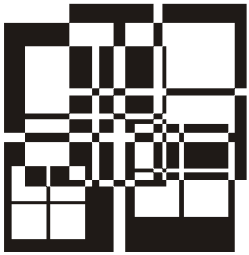
Мы также создали учебную базу данных тремя различными способами: с помощью утилиты isql, средствами IBExpert, написали собственные программы. Научились соединяться с базой данных и удалять базу данных, освоили средства создания резервной копии базы данных и восстановления базы данных с резервной копии. При этом оказалось возможным выполнять все эти действия как с базой данных на локальном компьютере, так и с базой данных, находящейся на любом компьютере в локальной сети. База данных может располагаться и на сетевом диске, для которого запрещен доступ с других компьютеров сети.

При создании базы данных необходимо задавать диалект клиента, который будет присвоен базе данных, и набор символов по умолчанию. При соединении с базой данных нужно опять же задать диалект клиента, а также используемый клиентом набор символов.

Для того чтобы в дальнейшем выполнять различные выборки данных в нашей базе данных, мы зарегистрировали нашу базу данных WORK.FDB в программе IBExpert.

Теперь самое время перейти к детальному рассмотрению объектов базы данных. Мы начнем с доменов; при этом разберемся с используемыми типами данных. Рассмотрим также литералы, применяемые в операторах SQL.

ГЛАВА 3



Работа с доменами

Домен — это один из объектов реляционной базы данных, при создании которого можно задать некоторые характеристики, а затем использовать ссылку на домен при определении столбцов таблиц. Один и тот же домен может использоваться в определении любого количества столбцов в таблицах базы данных.

> ***Замечание***
> Не знаю, в каких учебных заведениях и у каких преподавателей вы обучались, поэтому хочу напомнить, что в слове "домен" ударение падает на последний слог.

Важнейшей характеристикой домена является тип данных.

3.1. Типы данных

Тип данных определяет, во-первых, множество допустимых значений столбца и, во-вторых, множество допустимых операций.

Например, для числового типа данных множеством допустимых значений будет множество чисел, конкретный диапазон которых определяется формой представления типа данных (целое число, число с плавающей точкой или типы `NUMERIC` или `DECIMAL`) и количеством байтов, отводимых для хранения числа. Множеством допустимых операций для таких типов данных будут четыре арифметические операции — сложение, вычитание, умножение и деление.

Для строкового же типа данных множеством допустимых значений будет множество произвольных строк, а операция только одна — конкатенация (соединение двух строк в одну).

Типы данных для представления даты и времени позволяют хранить дату в определенном диапазоне и имеют свои нюансы при использовании арифметических операций.

Для полноты ощущений скажем несколько слов и о логическом (`BOOLEAN`) типе данных, который в Firebird, правда, не используется. Этот тип данных появился в InterBase 7.1. Множество допустимых значений составляют две константы — `TRUE` и `FALSE`, а множеством допустимых операций будут, по меньшей мере, операция отрицания, конъюнкция (логическое И) и дизъюнкция (логическое ИЛИ).

Мы рассмотрим некоторые типы данных из используемых в языке SQL серверов баз данных InterBase и Firebird, применяемые литералы, константы, предварительно определенные литералы, контекстные переменные даты и времени, преобразование типов, функции и операции над данными. Получается довольно обширная и, надо сказать, интересная программа действий. Приступим.

3.1.1. Числовые типы данных

Числовые типы данных представлены числами с фиксированной точкой (их также называют "точными" числами, exact numeric) и числами с плавающей точкой. Для всех числовых типов данных допустимы четыре арифметические операции: сложение, вычитание, умножение и деление.

3.1.1.1. Числа с фиксированной точкой

Числа с фиксированной точкой бывают *целыми* и *дробными*.

Целочисленные данные в SQL представлены тремя типами данных — `INTEGER`, `SMALLINT` и `BIGINT` (`BIGINT` не поддерживается в InterBase, вместо него можно использовать `DECIMAL(18)` — см. далее).

Тип данных `INTEGER` занимает 4 байта (или как любят писать в документации — 32 бита). Диапазон чисел, которые могут быть представлены в этом типе, составляет от –2,147,483,648 до +2,147,483,647. Заметьте, что здесь мы используем принятую в программистской документации систему обозначений, когда запятая применяется для разделения чисел (чтобы было удобно читать) на тысячи, миллионы и т. д. Для отделения целой части числа от дробной мы будем использовать, конечно, точку.

Вообще говоря, в литературе (обычно англоязычной) существует еще одна манера записи длинных чисел, когда в качестве такого разделителя используется не запятая, а апостроф (не хочу быть навязчивым в плане обучения русскому языку, но не могу удержаться и не сказать, что в слове "апостроф" уда-

рение опять же на последнем слоге). Тогда верхнюю границу для типа данных `INTEGER` следовало бы записать в виде +2'147'483'647.

Тип `SMALLINT` занимает 2 байта (16 битов). Диапазон чисел: от –32,768 до +32,767.

Тип данных `BIGINT` занимает 8 байтов (64 бита). Диапазон представляемых чисел совершенно фантастический, я даже не знаю, как произнести названия этих чисел: от –9,223,372,036,854,775,808 до +9,223,372,036,854,775,807[7].

Помимо всего прочего целочисленные типы данных используются для хранения значений искусственных первичных ключей. Об этом мы поговорим чуть позже.

Для представления **дробных чисел** с фиксированной точкой есть два похожих (если не сказать — одинаковых) числовых типа данных — `NUMERIC(n,m)` и `DECIMAL(n,m)`. Они позволяют хранить положительные и отрицательные числа с фиксированной десятичной точкой. При задании такого типа данных *n* (точность) определяет общее количество цифр в числе (включая десятичные знаки), максимальное значение 18; *m* (масштаб) — количество знаков после десятичной точки. Масштаб можно опустить, тогда принимается, что количество дробных знаков равно нулю. Разумеется, значение *m* не может превышать значения *n*. Эти типы удобно использовать, например, для представления денежных единиц.

Литералы для представления целых чисел имеют самый естественный вид — это нормальные обычные числа. В литерале дробного числа с фиксированной точкой дробная часть отделяется от целой точкой (а не запятой, как обычно у нас принято в обыденной жизни). Числовой литерал не должен заключаться в апострофы.

Сейчас я вас научу одному трюку, который очень любят знатоки InterBase и Firebird. Оператор SQL `SELECT` предназначен, как вы знаете, для выборки данных из таблиц базы данных. При этом в той части оператора, где перечисляются выбираемые столбцы таблицы, можно задавать литералы, константы и даже выражения. Этим мы сейчас и воспользуемся для исследования констант, преобразований и выражений.

Только что созданная, "пустая" база данных на самом деле содержит множество системных данных, в частности, системные таблицы. Есть таблица `RDB$DATABASE`, которая содержит ровно одну строку. Именно к этой таблице обычно и применяется оператор `SELECT` в данном трюке, хотя, конечно, можно использовать и любую другую таблицу, как системную, так и созданную

---

[7] На самом деле я научился произносить почти без запинки, однако получается слишком длинно: девять квинтиллионов, двести двадцать три квадриллиона, ...

пользователем. Правда, вместо одного набора значений, который вы всегда получите при использовании таблицы RDB$DATABASE, для других таблиц вы получите столько раз повторяющийся набор одних и тех же значений, сколько строк в этих таблицах (если не использовать предложение WHERE).

Запустите программу IBExpert. Любители командной строки могут использовать и утилиту isql, и там и тут работа выполняется похожим образом, хотя угадайте с трех раз — где удобнее работать?

Ранее вы зарегистрировали вашу базу данных WORK.FDB в программе IBExpert, поэтому после запуска программы в окне **Database Explorer** в левой части экрана дважды щелкните мышью по строке этой базы данных, это приведет к открытию базы данных. В меню **Tools** выберите элемент **SQL Editor** (редактор операторов SQL) или нажмите клавишу <F12>. Появится окно **SQL Editor**. Текущей будет вкладка **Edit**, где вы можете выполнять действия с базой данных WORK.FDB.

Одновременно можете запустить и утилиту isql. В подсказке утилиты введите оператор соединения с базой данных:

```
SQL> connect 'd:\BestDatabase\work.fdb'
CON> user 'wizard' password 'master';
```

В IBExpert введите оператор:

```
SELECT '18' FROM RDB$DATABASE
```

Нажмите клавишу <F9> или щелкните по кнопке **Run** (выполнить) для выполнения оператора. Текущей станет вкладка **Results** (результаты), где в единственной строке будет отображено число 18, прижатое влево. Это означает, что мы получили не число, а строку. Уберите в операторе SELECT апострофы и опять выполните оператор. Получим то же значение, но уже прижатое вправо, т. е. число.

Те же самые операторы выполните и в isql. На экран будет выведено следующее:

```
======
18
```

в первом случае и

```
============
          18
```

во втором.

Мы можем добавить заголовки в получаемый результат. Для этого после константы нужно указать AS "Желаемый заголовок". Например:

```
SELECT 18 AS "Целое" FROM RDB$DATABASE
```

Если в заголовке не требуются специальные символы, а присутствуют только буквы латинского алфавита, цифры, символы доллара и подчеркивания, то после ключевого слова AS можно задать текст без кавычек.

Посмотрите, как выполняются арифметические операции. Для сложения естественным образом используется символ +, для вычитания –, для умножения * и для деления /.

При работе с целыми числами все операции выполняются замечательно, кроме деления. Иногда пользователи ожидают других результатов. Введите и выполните:

```
SELECT 1 / 3 FROM RDB$DATABASE
```

Результатом будет 0.

На самом деле правила выполнения арифметических операций простые. При сложении и вычитании количество дробных знаков результата равно максимальному количеству дробных знаков у операндов. При умножении и делении результат имеет количество дробных знаков, равное сумме дробных знаков обоих операндов.

Чтобы в нашем случае получить дробное число нужной точности, необходимо делимое или делитель или оба представить в виде дробного числа, например:

```
SELECT 1.00 / 3 FROM RDB$DATABASE
```

Здесь результат 0.33. То же самое мы получим, если запишем операцию в следующем виде: 1.0 / 3.0. Сумма дробных знаков здесь так же равняется двум.

В языке SQL есть очень полезная и довольно часто используемая функция преобразования данных CAST. Ее формат:

```
CAST(<переменная или параметр или выражение> AS <тип данных>)
```

Чтобы получить предыдущий результат, введите и выполните:

```
SELECT CAST(1 AS DECIMAL(10,2)) / 3 FROM RDB$DATABASE
```

Здесь выполняется приведение целого числа (1) к типу DECIMAL с общим количеством знаков 10, из которых два знака отводятся под дробную часть.

Выполнение подобных преобразований для констант, конечно, не имеет почти никакого смысла, однако эта операция действительно очень полезна для столбцов таблиц и выражений.

3.1.1.2. Числа с плавающей точкой

Числа с плавающей точкой в основном используются в научных и инженерных расчетах, однако такое использование не всегда является оправданным. В SQL есть два типа данных чисел с плавающей точкой:

- `FLOAT` — 4 байта (32 бита), диапазон чисел от 1.175×10^{-38} до 3.402×10^{38}. Позволяет хранить до 7 значащих цифр;
- `DOUBLE PRECISION` — 8 байтов (64 бита), диапазон чисел от 2.225×10^{-308} до 1.179×10^{308}. Позволяет хранить до 15 значащих цифр.

Литералы для представления чисел с плавающей точкой по синтаксису соответствуют принятым нормам в языках программирования. Например, выполните оператор:

```
SELECT 1.84e-3 FROM RDB$DATABASE;
```

Результатом будет 0,00184. В IBExpert в результате округления вы получите 0,002. Чтобы получить более точное значение, выполните необходимое преобразование к десятичному числу с пятью десятичными знаками:

```
SELECT CAST(1.84e-3 AS DECIMAL(10, 5)) FROM RDB$DATABASE
```

Для числовых данных существует множество математических функций. Это так называемые функции, определенные пользователем — User Defined Function, UDF. О них мы поговорим несколько позже.

3.1.2. Типы данных даты и времени

Существует три типа данных для представления даты и времени — `DATE`, `TIME` и `TIMESTAMP`. Напомню, что мы с вами рассматриваем только диалект базы данных 3.

Тип `DATE` занимает 4 байта (32 бита), позволяет хранить даты в диапазоне от 1 января 1 года до 31 декабря 9999 года. Это я проверил лично в Firebird 1.5.3. Кстати, то же самое говорит и Хелен Борри в своей книге. В документации же по InterBase указан диапазон от 1 января 100 года до 29 февраля 32768 года.

Тип `TIME`, 4 байта (32 бита) — время в естественном диапазоне от 00:00:00.0000 до 23:59:59.9999 с точностью до 0.0001 сек.

Тип `TIMESTAMP`, 8 байтов (64 бита) позволяет хранить одновременно дату и время.

Для представления даты существует несколько вариантов литералов. Все они заключаются в апострофы, как и строковые данные. Это хорошее основание для того, чтобы их перепутать. По этой причине в большинстве случаев при работе с литералами даты следует выполнять их явное преобразование к типу данных `DATE`.

Литерал даты может быть представлен в одном из следующих вариантов:

```
'dd.mm.yyyy'
'mm-dd-yyyy'
'mm/dd/yyyy'
```

```
'yyyy-mm-dd'
'yyyy/mm/dd'
'yyyy.mm.dd'
'dd-MON-yyyy'
```

Здесь:

`dd` — номер дня в месяце: число от 1 до 31.

`mm` — номер месяца в году: число от 1 до 12.

`yyyy` — год: число от 1 до 9999.

`MON` — трехсимвольное сокращенное название месяца (английское). Может принимать значения (в любом регистре — можно писать строчными или прописными буквами или вперемешку) `jan`, `feb`, `mar`, `apr`, `may`, `jun`, `jul`, `aug`, `sep`, `oct`, `nov`, `dec`.

Например, возьмем 31 июля 2006 года. Эту дату можно записать следующим образом:

```
'31.07.2006'
'07-31-2006'
'07/31/2006'
'2006-07-31'
'2006/07/31'
'2006.07.31'
'31-jul-2006'
```

Для нас самым естественным будет, конечно, первый вариант: `'31.07.2006'`. Кстати, этот же формат используется и в Германии. Американцы предпочитают использовать с моей точки зрения не самый естественный формат `'mm/dd/yyyy'`.

Литерал времени задается в одном-единственном виде `'hh:mm:ss.nnnn'`.

Если нужно представить литерал даты и времени, то весь литерал заключается в апострофы, вначале задается дата и через один или более пробелов время, например: `'dd.mm.yyyy  hh:mm:ss.nnnn'`.

Существует четыре предварительно определенных литерала даты:

- `'NOW'` типа `TIMESTAMP` возвращает текущую дату и текущее время;
- `'TODAY'` типа `DATE` возвращает текущую дату;
- `'TOMORROW'` типа `DATE` возвращает завтрашнюю дату;
- `'YESTERDAY'` типа `DATE` возвращает вчерашнюю дату.

Эти литералы не чувствительны к регистру — вы можете вводить их как прописными (заглавными) буквами, так и строчными. При использовании этих литералов в операторе `SELECT` необходимо явно выполнить их преобразова-

ние к нужному типу данных, иначе они будут рассматриваться как строковые константы.

Выполните следующие операторы:

```
SELECT CAST('NOW' AS DATE) FROM RDB$DATABASE
SELECT CAST('NOW' AS TIMESTAMP) FROM RDB$DATABASE
```

Вы получите текущую дату, а во втором случае еще и время. В используемой мною версии IBExpert доли секунды при преобразовании времени и даты/времени функцией `CAST` отображаются неверно — вместо десятитысячных долей секунды после точки повторно почему-то выводится количество минут.

Существуют еще так называемые контекстные переменные даты и времени:

- `CURRENT_TIMESTAMP` типа `TIMESTAMP` возвращает текущую дату и текущее время;
- `CURRENT_DATE` типа `DATE` возвращает текущую дату;
- `CURRENT_TIME` типа `TIME` возвращает текущее время.

Интересно выполняются операции сложения и вычитания для дат и времени.

`DATE + <число>` дает указанную дату, увеличенную на заданное число дней. Здесь используется только целое число.

`DATE – <число>` дает указанную дату, уменьшенную на заданное число дней. Здесь также можно использовать только целое.

`DATE – DATE` — количество дней в интервале.

`TIME + <число>` дает указанное время, увеличенное на заданное число секунд, включая десятитысячные доли секунды. Здесь можно использовать дробное число. Например, выражение

```
'01:01:06.0001' + 1.0001
```

дает время `'01:01:07.0002'`.

`TIME – <число>` дает указанное время, уменьшенное на заданное число секунд, включая десятитысячные доли секунды. Выражение

```
'01:01:07.0002' – 1.0001
```

дает время `'01:01:06.0001'`.

`TIME – TIME` — дает количество секунд в интервале, включая десятитысячные доли секунды.

Можете проверить эти операции в IBExpert или isql. В частности, можете определить, сколько прошло дней с момента вашего рождения до сегодняшнего дня:

```
SELECT CAST('now' AS DATE) – CAST('ваша дата рождения' AS DATE) FROM
RDB$DATABASE
```

Найдите количество секунд в интервале:

```
SELECT CAST('12:48:59.0234' AS TIME) – CAST('12:48:59.0239' AS TIME) FROM
RDB$DATABASE
```

Вы получите результат: –0,0005.

Для типов даты и времени может использоваться замечательная функция `EXTRACT`, позволяющая выделять различные элементы. Ее вид:

```
EXTRACT (<выделяемый элемент> FROM <поле>)
```

Выделяемым элементом может быть:

- `YEAR` — год: функция вернет число от 1 до 9999;
- `MONTH` — месяц: функция вернет число от 1 до 12;
- `DAY` — день месяца: вернет число от 1 до 31;
- `HOUR` — часы;
- `MINUTE` — минуты;
- `SECOND` — секунды и десятитысячные доли секунды;
- `WEEKDAY` — номер дня в неделе; 0 — воскресенье;
- `YEARDAY` — номер дня в году: число от 0 до 365.

Вы также можете проверить выполнение этой функции. Например, оператор

```
SELECT EXTRACT(SECOND FROM CAST('13.12.2006 12:48:59.0234' AS TIMESTAMP))
FROM RDB$DATABASE
```

дает число 59,0234.

Оператор

```
SELECT EXTRACT(YEARDAY FROM CAST('31.12.2008' AS TIMESTAMP)) FROM
RDB$DATABASE
```

возвращает 365. Не забывайте, что дни года нумеруются, начиная с нуля, так что номер последнего дня в високосном году, которым является 2008 год, указанный в нашем примере, будет не 366, а 365. Соответственно, 31 декабря любого другого "обычного" года будет иметь номер 364.

Кстати, в этом примере можно видеть, что допустимо преобразование даты к типу `TIMESTAMP`. Просто в результате время будет нулевым. Однако преобразование даты и времени (`TIMESTAMP`) к типу времени (`TIME`) дает исключение по переполнению.

3.1.3. Строковые типы данных

Строковые данные представлены типами данных `CHAR(n)` и `VARCHAR(n)` Полные названия для этих типов данных, соответственно, `CHARACTER` и `CHARACTER VARYING`. Они позволяют хранить до 32 767 символов. Число в скобках (*n*) задает максимально допустимое для данного столбца количество символов. В базе данных эти типы данных хранятся одинаковым образом — конечные пробелы отбрасываются. Отличие их в том, что при отображении данные `CHAR` дополняются справа пробелами до общего количества символов *n*.

Строковые литералы заключаются в апострофы. Если сам литерал содержит апострофы, то каждый апостроф внутри литерала должен быть записан дважды. Например, чтобы представить в SQL литерал "Mr. and Mrs. Hunt's children", нужно написать:

```
'Mr. and Mrs. Hunt''s children'
```

Для строковых типов данных допустима только операция конкатенации — соединение двух строк в одну. Для этой операции используются два символа вертикальной черты (без пробела между ними). Например, конструкция

```
'Операция соединения ' || 'двух строк в одну'
```

даст результирующую строку

```
"Операция соединения двух строк в одну"
```

Обратите внимание — в конце первой подстроки стоит пробел. Часто отсутствие таких пробелов приводит к не очень красивым результатам.

Если вы работаете со строками ну очень больших размеров, то будьте осторожны с операцией конкатенации. Если результат операции превысит 32К символов, то осложнения в работе неизбежны.

Для строковых данных также существуют UDF (функции, определенные пользователем). Это интересный предмет для рассмотрения, но несколько позже.

3.1.4. Тип данных *BLOB*

Очень интересный и полезный тип данных — `BLOB`. Его называют большим двоичным объектом (Binary Large OBject). Он позволяет хранить любые данные — форматированные тексты, графику, звуки, видео.

В некоторых документах вы можете найти сведения, что максимальный размер данных 64 Кбайт. Это не совсем так. 64 Кбайт — это максимальный размер сегмента. Данные типа `BLOB` хранятся в базе данных в сегментах. Однако вопросами физической организации хранения мы заниматься не будем. Мак-

симальный же размер самих данных зависит от размера страницы базы данных.

При размере страницы 1024 байта размер BLOB не может превышать 64 Мбайт, 2048 — 512 Мбайт, 4096 — 4 Гбайт, 8192 — 32 Гбайт. Для размера страницы 16 384 байта я соответствующего значения не нашел, однако по логике вещей (при минимальных арифметических расчетах) можно предположить, что в этом случае размер BLOB не должен превышать 256 Гбайт.

При объявлении столбца или домена типа BLOB можно указать его подтип (предложение SUB_TYPE), а также размер сегмента, используемый при хранении данных (предложение SEGMENT SIZE).

Подтипы могут применяться в случае существования в базе данных так называемых BLOB-фильтров. Это программы, которые выполняют преобразования между данными BLOB разных подтипов. Преобразования связаны, как правило, с упаковкой данных.

Существуют определенные в системе неотрицательные подтипы: числа от 0 до 6, как минимум. Не используйте эти, да и другие положительные значения типов для своих особых данных. Вы можете использовать отрицательные числа для подтипов.

Размер сегмента задает размер полей в базе данных, которые будут использованы для хранения данных типа BLOB. По умолчанию принимается 80. Максимально возможное значение 64 Кбайта. Я обычно использую размер 400.

3.1.5. Тип данных *BOOLEAN*

Логический тип данных BOOLEAN существует в InterBase 2007. Он занимает по непонятным для меня причинам 2 байта. Может принимать значения TRUE, FALSE, NULL. Мне проще вместо логического типа данных использовать тип данных CHAR(1). Это решает все проблемы.

3.2. Синтаксис оператора создания домена

Для создания домена используется оператор CREATE DOMAIN, который относится к языку определения данных (DDL), который, в свою очередь, является подмножеством языка SQL. Синтаксис этого оператора:

```
CREATE DOMAIN <имя домена> [AS] <тип данных>
   [DEFAULT {<литерал> | NULL | USER}]
   [NOT NULL]
   [CHECK (<условие домена>)]
   [COLLATE <порядок сортировки>];
```

Имя домена должно быть уникальным среди имен доменов базы данных.

Несколько упрощенный синтаксис для типа данных:

```
<тип данных> ::=
{SMALLINT | INTEGER | BIGINT | FLOAT | DOUBLE PRECISION | BOOLEAN}
| {DATE | TIME | TIMESTAMP}
| {DECIMAL | NUMERIC} [(<n> [,<m>])]
| {CHAR | VARCHAR} [(<n>)] [CHARACTER SET <набор символов>]
| BLOB [SUB_TYPE <n>] [SEGMENT SIZE <n>] [CHARACTER SET <набор символов>]
```

> ***Напоминание***
>
> Тип данных `BIGINT` допустим только в Firebird, а `BOOLEAN` — в InterBase.

В описании домена обязательно должно присутствовать имя домена и тип данных. Типы данных мы с вами довольно подробно рассмотрели ранее в этой главе *(см. разд. 3.1)*. Остальные элементы (предложения, заключенные в квадратные скобки) могут отсутствовать.

Тип данных — один из типов данных SQL. Для всех типов данных, кроме `BLOB`, можно указать, что домен описывает массив. Для этого после типа данных нужно в квадратных скобках задать размерность массива (в описании синтаксиса не показано). Например, следующий оператор создает домен, являющийся массивом целых чисел размером 10 на 10:

```
CREATE DOMAIN ARRAYEXAMPLE AS INTEGER[10, 10];
```

Вообще говоря, использование массивов в реляционных базах данных не является хорошей идеей. Это нарушает требование первой нормальной формы, по которому значение любого столбца должно быть атомарным. Тем не менее использование массивов в InterBase/Firebird возможно и в некоторых случаях бывает действительно эффективно. Видимо для того и создаются правила, чтобы их нарушать (но только в случае острой необходимости).

При создании домена типа `CHAR`, `VARCHAR` и `BLOB` вы можете указать в предложении `CHARACTER SET` и набор символов, отличный от набора символов по умолчанию, заданному для всей базы в предложении `DEFAULT CHARACTER SET` оператора создания базы данных `CREATE DATABASE`. Если только вы действительно задавали набор символов по умолчанию для базы данных. В противном случае для каждого столбца, в том числе и основанного на домене, обязательно нужно задать набор символов, отличный от набора символов `NONE`. Иначе неприятностей не избежать с таким набором символов.

Если домен определяет строковый тип данных (`CHAR` или `VARCHAR`), то для него можно указать предложение `COLLATE`, в котором задается порядок сортировки. Списки наборов символов и допустимых для каждого из них порядков сортировки представлены в *приложении 2*. Для строковых столбцов, которые будут хранить буквы кириллицы, следует задать набор символов `WIN1251`. Порядок сортировки я использую `PXW_CYRL`. Он позволяет наиболее правиль-

но сортировать как тексты кириллицы, так и тексты, использующие латинские буквы *(см. приложение 2)*.

Фраза `DEFAULT` определяет значение по умолчанию.

Значение по умолчанию будет присвоено столбцу, основанному на данном домене, при добавлении в таблицу новой строки (оператор `INSERT`), если в операторе `INSERT` не указано значение этого столбца. Значение по умолчанию не используется при изменении существующей строки таблицы (оператор `UPDATE`).

Можно указать:

- конкретное значение (литерал или, иными словами, самоопределенное значение, например, числа `2`, `0.5`, `-2.7E+1` — если кто не знает, то это равно -2.7×10^1 или просто –27, — строковый литерал `'Как жить дальше?'`, дата `'14.11.2006'`, время `'23.05.18'` и др.);
- пустое значение (`NULL`);
- имя текущего пользователя (`USER`). Это имя пользователя, который в настоящее время (когда в базу данных помещается новая строка) подключен к этой базе данных с помощью оператора `CONNECT`.

Литерал должен соответствовать типу данных домена. Для числового типа данных можно указать только число, соответствующее типу данных этого домена. Для строкового данного можно указать строку символов, заключенную в апострофы. Если в такой строке должен быть представлен апостроф, то следует записать два подряд идущих апострофа.

Для даты и времени можно использовать либо обычные литералы, либо предварительно определенные литералы.

В значении по умолчанию задание `NULL` означает, что столбцу будет присвоено пустое значение. Кажется несколько странным использование этого варианта. Дело в том, что при отсутствии данного столбца в списке при добавлении новой строки (это мы рассмотрим несколько позже) ему будет присвоено пустое значение. Однако эта конструкция имеет смысл в случае изменения значения по умолчанию, когда на этом домене основывается столбец таблицы (см. далее).

Имя текущего пользователя (`USER`) позволяет поместить в строковый столбец имя пользователя, который соединен с базой данных в текущем сеансе. Это имя пользователя, которое было указано в предложении `USER` оператора `CONNECT`.

Понятно, что предложение `NOT NULL` запрещает помещение в столбец, основанный на данном домене, пустого значения. Следует быть аккуратным при описании домена и не допускать одновременного присутствия предложения `DEFAULT NULL` и предложения `NOT NULL`.

Очень интересное (и довольно сложное) предложение CHECK. Оно позволяет выполнить проверку значения, помещаемого в столбец (оператор INSERT) или изменяемое в столбце (оператор UPDATE), который основан на данном домене, и выдать исключение, если значение не соответствует заданным условиям. Предложение CHECK в описании домена относится к так называемым ограничениям (constraint) базы данных.

Синтаксис условия домена в предложении CHECK:

```
<условие домена> ::=
  VALUE <оператор> <значение>
  | VALUE [NOT] BETWEEN <значение> AND <значение>
  | VALUE [NOT] LIKE <значение>
  | VALUE [NOT] IN (<значение> [, <значение> ...])
  | VALUE IS [NOT] NULL
  | VALUE [NOT] CONTAINING <значение>
  | VALUE [NOT] STARTING [WITH] <значение>
  | (<условие домена>)
  | NOT <условие домена>
  | <условие домена> OR <условие домена>
  | <условие домена> AND <условие домена>
```

Ключевое слово VALUE замещает значение, присваиваемое столбцу, основанному на этом домене.

Рассмотрим этот синтаксис и определим его семантику (то есть смысл каждой из конструкций).

VALUE <оператор> <значение>

Здесь оператор — это один из операторов сравнения: =, <, >, <=, >=, !<, !> <>, !=, ^=, ^>, ^<. Восклицательный знак и символ ^ в этих операторах означают отрицание.

Пример. Чтобы указать, что значения числового столбца должно быть положительным, нужно записать:

```
CHECK (VALUE > 0)
```

VALUE [NOT] BETWEEN <значение> AND <значение>

Значение (числовое или строковое) должно находиться между двумя указанными значениями, включая и эти граничные значения. Если задано необязательное ключевое слово NOT, то значение *не должно* находиться в указанном диапазоне. Это верно и для всех последующих предложений.

Пример. Следующая конструкция указывает, что значение должно состоять из трех букв латинского алфавита в верхнем регистре:

```
CHECK (VALUE BETWEEN 'AAA' AND 'ZZZ')
```

VALUE [NOT] LIKE <значение>

Строковое значение должно содержать указанные символы. Здесь можно использовать шаблонные символы — знак процента (%) и подчеркивание (_). % означает произвольное количество (в том числе и нулевое) любых символов. Знак подчеркивания означает ровно один любой символ. Предложение LIKE является чувствительным к регистру.

Примеры. В следующем предложении проверяется значение столбца, основанного на домене. Значение должно оканчиваться на символы "ОВ", перед ними может быть любое количество любых символов.

```
CHECK (VALUE LIKE '%ОВ')
```

Подобную конструкцию мы в дальнейшем используем в операторе SELECT, когда будем отыскивать людей, чьи фамилии оканчиваются на буквы "ОВ", например, "ИВАНОВ", "ТУХМАНОВ". Чтобы не использовать дискриминационные методы по отношению к женщинам, следует изменить предыдущее условие:

```
CHECK (VALUE LIKE '%ОВ%')
```

Сюда уже подойдут и фамилии "ИВАНОВА", "ТУХМАНОВА". Однако станут допустимыми и другие: "СОВКОВ", "ЛОВЧАН", "ОВЕЧКИН" и др. Все они содержат текст "ОВ" в любом месте своей фамилии.

В следующем примере зададим шаблон для ввода строки символов, определяющей междугородный номер телефона. Вначале в круглых скобках указывается трехсимвольный код города, затем сам номер телефона в виде трех символов, знака тире, двух символов, тире и еще двух символов. (Понятно, что этот пример несколько надуманный.)

```
CHECK (VALUE LIKE '(___)___-__-__')
```

VALUE [NOT] IN (<значение> [, <значение> ...])

Вводимое в столбец значение должно находиться среди значений, представленных в списке. Это предложение может относиться к любому типу данных, кроме BLOB. Если задано ключевое слово NOT, то значение, конечно же, *не должно* находиться среди этих значений.

Пример. Следующее условие требует, чтобы числовое значение было нечетным числом, меньшим 10:

```
CHECK (VALUE IN (1, 3, 5, 7, 9))
```

VALUE IS [NOT] NULL

Вариант VALUE IS NOT NULL эквивалентен предложению в описании домена NOT NULL. Вариант же VALUE IS NULL вызывает серьезные сомнения — зачем нужен столбец, который имеет пустое значение? Отдельно взятое такое огра-

ничение действительно бессмысленно, однако в сочетании с другими условиями в логическом выражении этот вариант имеет все-таки конкретный смысл.

Пример. Следующее условие определяет проверку логического столбца (описанного с типом данных `CHAR(1)`), который может принимать значения 0 (ложь), 1 (истина) или `NULL` (неизвестное значение):

```
CHECK ((VALUE = '0') OR (VALUE = '1') OR (VALUE IS NULL))
```

Если в этом случае не указать допустимость пустого значения, то столбцу, основанному на этом домене, нельзя будет присвоить значение `NULL`, даже если в его описании не присутствует предложение `NOT NULL`.

Кстати, чтобы избежать неприятностей, связанных с порядком вычисления выражений, рекомендуется всегда использовать скобки для явного задания такого порядка.

Замечание

Если столбец включен в состав первичного ключа, то ему нужно явно присвоить атрибут `NOT NULL` в основной части описания при создании домена или столбца таблицы.

`VALUE [NOT] CONTAINING <значение>`

Выполняется проверка на присутствие в строковом значении указанной подстроки символов. Замечательной особенностью этого предложения является нечувствительность к регистру.

Пример. Когда мы строили варианты проверок в предложении `VALUE LIKE '%OB%'`, мы могли бы задать выражение:

```
VALUE CONTAINING 'OB'
```

что позволило бы нам выполнить ту же самую проверку и при этом не учитывать регистра букв.

`VALUE [NOT] STARTING [WITH] <значение>`

Значение должно начинаться с указанных символов.

Предложение `STARTING WITH` является чувствительным к регистру.

Пример. В одной организации, занимающейся торговлей, была установлена программная система, позволяющая отслеживать приход на склад и движение товара. Организационным недостатком использования этой системы было то, что не существовало службы централизованного ведения справочника товаров, и несколько человек вводили свои данные. Один из сотрудников по непонятной причине иногда название товара начинал с символа `*`. Никакие уговоры перестать так делать не помогали, а уволить его не могли, поскольку он был ценным во всех остальных отношениях

сотрудником. Тогда решили в домен, на котором был основан столбец, хранящий название товара, ввести следующее ограничение:

```
CHECK (VALUE NOT STARTING WITH '*')
```

Это помогло. Сотрудник не мог начинать название товара с полюбившегося ему символа *. Попытки вначале ввести пробел, а затем символ "звездочка", ни к чему не привели, поскольку в программе, с помощью которой вводилось название, использовалась функция Trim, отсекающая и начальные, и конечные пробелы.

После этого они жили счастливо.

Последние четыре строки в описании синтаксиса условия домена:

```
|(<условие домена>)
| NOT <условие домена>
| <условие домена> OR <условие домена>
| <условие домена> AND <условие домена>
```

позволяют комбинировать из этих условий любые логические конструкции. Вы можете любую часть условия заключить в скобки, выполнить отрицание условия (как простого, так и сложного, заключенного в скобки), включить в операцию дизъюнкцию (OR, логическое ИЛИ) или конъюнкцию (AND, логическое И).

Мы еще вернемся к условиям подобного вида, когда будем рассматривать условия (ограничения) таблицы и предложение WHERE операторов UPDATE, DELETE и SELECT. Там синтаксис является расширенным вариантом ограничения домена.

Когда вы будете описывать столбцы, основанные на существующем домене, вы сможете изменить и расширить ограничения домена, заданные предложением CHECK.

В необязательном предложении COLLATE задается порядок сортировки для строковых полей — столбцов, имеющих тип данных CHAR или VARCHAR. Если предложение не указать, то для столбцов, основанных на этом домене, будет использоваться порядок сортировки по умолчанию, что не всегда бывает полезным.

Если столбцы таблиц, которые будут основываться на создаваемом домене, могут содержать буквы кириллицы, и вы допускаете, что вашему пользователю может когда-либо прийти в голову мысль выполнять сортировку по таким столбцам, то следует явно указать порядок сортировки PXW_CYRL.

В нашей учебной базе данных при создании большинства строковых доменов указано предложение

```
COLLATE PXW_CYRL
```

Замечание

Хорошей практикой является явное задание порядка сортировки для любых строковых столбцов, если вас не устраивает порядок сортировки по умолчанию.

Заметьте, для домена, используемого при создании столбца кода страны (`D_CHAR3`), я не указал порядок сортировки, поскольку значения кода страны предопределены, в них не помещаются буквы кириллицы. В результате для домена стало использоваться значение по умолчанию — `WIN1251` (не путайте с набором символов). После этого с помощью надуманных примеров я провел эксперименты по проверке упорядочивания букв кириллицы в коде страны для этого порядка сортировки.

3.3. Создание доменов

Теперь создадим пару доменов для нашей учебной базы данных.

Первый домен предположительно будем использовать для столбца, в который будет помещаться код страны. Другой предназначен для хранения названия страны.

В isql или в IBExpert в окне Script Executive введите следующие операторы (не забывайте заканчивать каждый оператор символом точка с запятой) — листинг 3.1.

Листинг 3.1. Создание двух пробных доменов

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';

CREATE DOMAIN COUNTRYCOD CHAR(3)
  CHECK ((VALUE BETWEEN '000' AND '999') OR
        ((VALUE >= 'AAA') AND (VALUE <= 'ZZZ')));
CREATE DOMAIN COUNTRYNAME VARCHAR(30);

COMMIT;
```

Вы помните, что изменение метаданных, к которым относятся и домены, также выполняется в контексте какой-либо транзакции. В нашем случае это транзакция по умолчанию, которая стартует автоматически. После создания доменов мы подтверждаем транзакцию с помощью оператора `COMMIT`.

Для домена `COUNTRYCOD` задается условие проверки — значение, помещаемое в столбец или изменяемое в столбце, основанном на этом домене, должно содержать либо три десятичные цифры, либо три латинские буквы в верхнем регистре. Для разнообразия проверка на цифры осуществляется с помощью

конструкции BETWEEN, а для букв используются операции сравнения. Результат будет одинаковым.

Обратите внимание на обилие скобок в предложении CHECK. Здесь каждое условие мы заключили в скобки, явно задавая порядок выполнения проверок. Это избавит нас в дальнейшем от головной боли при выяснении причин, почему проверка осуществляется не в том виде, как мы предполагали.

Поскольку в предложении CHECK не было задано VALUE IS NULL, то столбцу, основанному на этом домене, нельзя будет присвоить пустого значения, если столбец не содержит дополнительного предложения CHECK, в котором допускается значение NULL. Несмотря на все сказанное, если вы используете такой столбец в качестве составной части первичного ключа, вы должны в описании этого столбца или домена в обязательном порядке явно задать NOT NULL.

Выполните введенные операторы. Если вы все набрали правильно (и не ошиблись с количеством открывающих и закрывающих скобок), то в базу данных будет помещено два домена.

Прямо сейчас я в IBExpert выполнил этот скрипт. Домены были успешно созданы.

На этом создание доменов вручную для нашей базы данных мы завершим. Чуть позже выполним некоторые изменения созданных доменов. В реальной (моей) жизни я создаю домен для *каждого* вида характеристик столбца. Если в моей базе данных есть десятки или сотни столбцов, имеющих тип данных CHAR(10), то я создам один домен:

```
CREATE DOMAIN D_CHAR10 AS CHAR(10);
```

и буду ссылаться на него при определении каждого соответствующего столбца. Дело в том, что сервер базы данных для каждого столбца с типом данных CHAR(10) автоматически создаст свой домен, присвоив странное имя. Таким образом в базе данных может оказаться сотня доменов с одинаковыми характеристиками, но с разными именами, что не очень хорошо как по причине перерасхода внешней памяти, так и по причине увеличения времени выполнения операций резервного копирования и восстановления базы данных.

Необходимые проверки условий, если это действительно нужно, я задаю не для домена, а для столбцов таблицы, основанных на доменах.

> **ВНИМАНИЕ!**
>
> Когда вы создаете столбцы таблицы, основываясь на домене, в котором присутствует ограничение CHECK, и для такого столбца таблицы также задается ограничение CHECK, то происходит объединение двух условий с помощью операции конъюнкции — логической операции И.
>
> Подробнее об этом смотрите в следующей главе.

3.4. Изменение домена

Практически все характеристики существующего домена можно изменить с помощью оператора ALTER DOMAIN. Его синтаксис:

```
ALTER DOMAIN {<имя> | <старое имя> TO <новое имя>}
  { SET DEFAULT {<литерал> | NULL | USER}
    | DROP DEFAULT
    | ADD [CONSTRAINT] CHECK (<условие домена>)
    | DROP CONSTRAINT
    | TYPE <тип данных>};
```

Мы можем переименовать домен (конструкция <старое имя> TO <новое имя>), установить новое значение по умолчанию (предложение SET DEFAULT), удалить значение по умолчанию (DROP DEFAULT), задать новое условие проверки вводимых данных (ADD [CONSTRAINT] CHECK), удалить существующее условие проверки (DROP CONSTRAINT) и даже изменить тип данных (предложение TYPE).

Нельзя только удалить условие NOT NULL.

Для того чтобы добавить новое ограничение CHECK, нужно вначале удалить существующее.

Значение по умолчанию (DEFAULT) можно изменять, не удаляя существующего значения. Можно также удалять значение по умолчанию, даже если оно и не установлено. Вы не получите сообщения об ошибке.

Разглядывая внимательно описание синтаксиса оператора изменения домена, можно заметить, что с его помощью мы не можем изменить порядок сортировки. Не знаю, с чем это связано.

Вы можете изменять домен и в том случае, когда на этом домене основаны столбцы в таблицах, и таблицы заполнены данными. Однако новые характеристики домена будут использованы только при создании *новых* таблиц, но не при добавлении новых строк существующих таблиц или при изменении значений существующих строк. Тогда описания таблиц, созданных до изменений доменов, будут не соответствовать реальному положению дел в базе данных.

Внесите изменения в созданный вами домен. В isql или в IBExpert выполните следующие операторы — листинг 3.2.

Листинг 3.2. Изменение пробных доменов

```
SET SQL DIALECT 3;
```

```
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';

ALTER DOMAIN COUNTRYCOD
 DROP CONSTRAINT;
ALTER DOMAIN COUNTRYCOD
 ADD CONSTRAINT CHECK (VALUE BETWEEN 'ААА' AND 'ЯЯЯ');
COMMIT;
```

Здесь вначале мы удаляем существующее ограничение `CHECK`, затем создаем новое. На этот раз мы решили допустить в значениях только прописные буквы кириллицы.

3.5. Удаление домена

Для удаления домена используется оператор `DROP DOMAIN`. Его синтаксис:

```
DROP DOMAIN <имя домена>;
```

Разумеется, для удаления домена вы предварительно должны соединиться с базой данных как ее владелец или пользователь `SYSDBA`.

Созданные нами два домена были нужны только для того, чтобы немного поэкспериментировать. На самом деле они нам в дальнейшем не потребуются. Удалите оба в isql или IBExpert (листинг 3.3).

Листинг 3.3. Удаление только что созданных доменов

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';
DROP DOMAIN COUNTRYCOD;
DROP DOMAIN COUNTRYNAME;
COMMIT;
```

3.6. Создание доменов для учебной базы данных

Теперь, после того как вы создавали, изменяли и, наконец, удалили все временные домены, пора создать в нашей базе данных домены, которые будут реально использоваться при создании таблиц.

В isql, используя команду INPUT, или в IBExpert, используя инструмент Script Executive, выполните скрипт CreateDomains.sql (листинг 3.4).

Листинг 3.4. Скрипт CreateDomains.sql

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';

CREATE DOMAIN D_DATE     AS DATE;
CREATE DOMAIN D_INTEGER  AS INTEGER;
CREATE DOMAIN D_CHAR1    AS CHAR(1);
CREATE DOMAIN D_CHAR2    AS CHAR(2);
CREATE DOMAIN D_CHAR3    AS CHAR(3);
CREATE DOMAIN D_CHAR4    AS CHAR(4);
CREATE DOMAIN D_CHAR6    AS CHAR(6);
CREATE DOMAIN D_CHAR8    AS CHAR(8);
CREATE DOMAIN D_DECIMAL  AS DECIMAL(8, 2);
CREATE DOMAIN D_CHAR15   AS VARCHAR(15) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR20   AS VARCHAR(20) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR25   AS VARCHAR(25) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR30   AS VARCHAR(30) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR35   AS VARCHAR(35) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR40   AS VARCHAR(40) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR50   AS VARCHAR(50) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR60   AS VARCHAR(60) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR110  AS VARCHAR(110) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR120  AS VARCHAR(120) COLLATE PXW_CYRL;
CREATE DOMAIN D_CHAR1000 AS VARCHAR(1000) COLLATE PXW_CYRL;
CREATE DOMAIN D_BLOB     AS BLOB SUB_TYPE 1 SEGMENT SIZE 400;

COMMIT;

EXIT;
```

У меня каждый скрипт заканчивается оператором EXIT. Рекомендую и вам заканчивать ваши скрипты этим оператором.

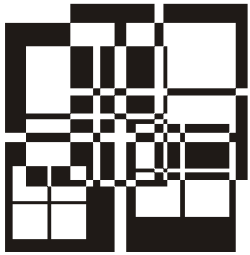
В результате будут созданы все необходимые для дальнейшей работы домены. На этих доменах будут основываться столбцы создаваемых таблиц базы данных.

Что там за перевалом?

В этой главе мы достаточно подробно ознакомились с типами данных, используемыми в SQL. Рассмотрели литералы, константы, операции и полезные функции. Познакомились с доменами, научились их создавать, изменять и удалять. Выполнили большой объем экспериментальной работы по исследованию типов данных.

Пора приступать к проектированию и созданию таблиц — важнейшего объекта реляционной базы данных.

ГЛАВА 4



Работа с таблицами

Самый важный и самый сложный объект реляционной базы данных, конечно, таблица. Именно в таблицах содержатся все обрабатываемые данные предметной области. Именно описание таблицы является в декларативной части языка SQL (в языке описания данных DDL) наиболее объемным и сложным, но, в то же время, и наиболее интересным, поскольку здесь можно увидеть всю мощь описательной части реляционных баз данных.

База данных может иметь произвольное количество таблиц[8]. Таблица может состоять из одного или большего количества столбцов. Для таблицы могут быть заданы некоторые *ограничения* (constraints). Ограничениями таблицы являются: первичный ключ (`PRIMARY KEY`), уникальный ключ (`UNIQUE`), внешний ключ (`FOREIGN KEY`) и ограничение на значения столбцов `CHECK`.

4.1. Первичные ключи

Сейчас мы рассмотрим ограничение первичного ключа. Другими ограничениями займемся несколько позже, когда будем исследовать синтаксис оператора создания таблиц — `CREATE TABLE`.

Хотя серверы базы данных InterBase и Firebird и не требуют обязательного наличия у таблиц первичного ключа, тем не менее очень желательно, чтобы *каждая* таблица имела первичный ключ. Кстати, это требование стандарта SQL-92. Первичный ключ может состоять из одного или нескольких столбцов. Основное требование к первичному ключу — его уникальность. Иными словами, в таблице не должно быть двух различных строк, имеющих одина-

[8] Для InterBase 2007 максимальное количество таблиц 32 640. Думаю, что для нас это ограничение практически означает отсутствие таковых.

ковое значение первичного ключа. Кроме того, столбец, включенный в состав первичного ключа, не может иметь пустого (`NULL`) значения.

Существует два варианта выбора первичного ключа у таблицы — так называемый "естественный" и искусственный ключ.

Можно выбрать в качестве первичного ключа столбец или группу "осмысленных" столбцов таблицы. Здесь следует исходить из того, что размер первичного ключа не должен быть слишком большим. Во-первых, для первичного ключа в базе данных автоматически создается индекс, размер которого зависит от размера первичного ключа и количества строк в таблице, и, во-вторых, часто на первичный ключ данной таблицы ссылаются внешние ключи других, подчиненных, таблиц. Подчиненные таблицы, как правило, имеют гораздо больше строк, чем родительская таблица. Кроме того, для внешних ключей также создаются индексы. При большом размере первичного ключа размер внешней памяти, выделяемый для базы данных, может быть очень большим. Это также приведет к увеличению времени реакции системы.

Существуют средства расчета максимального размера первичного ключа в байтах. Ориентировочно можно принять этот размер не более 200 байтов, хотя это уже сильно повлияет на требуемый объем внешней памяти для базы данных и резко ухудшит производительность системы.

Второй способ — добавление в таблицу числового столбца, которому при помещении в базу данных новой строки будет присваиваться уникальное значение, получаемое из генератора — особого объекта базы данных. Этот столбец будет искусственным ключом. Такой первичный ключ нужно использовать, когда в таблице невозможно найти подходящих кандидатов в первичные ключи или когда размер такого ключа слишком велик. Искусственный первичный ключ компактен — 4 (тип данных `INTEGER`) или 8 (`BIGINT`) байтов. В некоторых, довольно редких случаях, когда количество записей таблицы ну очень мало, можно также использовать тип данных `SMALLINT` (2 байта). Этот тип данных в качестве первичного ключа я, например, использую только для таблицы, содержащей локальные списки пользователей программной системы (некий аналог учетных записей пользователей).

Рассмотрим такой пример. Пусть вам нужно выбрать первичный ключ для таблицы, содержащей сведения о людях. Конечно, первым кандидатом в первичные ключи является фамилия. Это может пройти для очень небольшой группы людей, однако однофамильцы встречаются весьма часто. В первичный ключ можно добавить имя и отчество. Но при достаточно большом количестве людей в таблице могут встретиться и такие полные тезки. Добавляем еще и дату рождения. Казалось бы, есть основание для радости, однако ключ получился очень уж большим (около 60—70 байтов). И дело не только в том, что база данных становится слишком большой за счет создания объемных индексов. Операции выборки данных в такой базе будут выполняться

довольно долго. В данном случае явно нужен искусственный первичный ключ.

С другой стороны, если даже на очень большом предприятии сотрудникам присвоены табельные номера, то столбец табельного номера очень хорошо подойдет для использования в качестве первичного ключа, поскольку бухгалтерия и отдел кадров внимательно следят за тем, чтобы табельные номера не повторялись.

Можно сформулировать рекомендации по выбору первичного ключа.

- Для справочных данных — т. е. для таблиц, данные в которых изменяются довольно редко, — можно использовать естественный ключ: один-единственный столбец, некоторый код.

  Примером может служить справочник стран. Он имеет трехсимвольный код страны, который мы и используем в качестве первичного ключа. Справочник регионов (республик, краев, областей) страны должен содержать код страны и код региона. Оба эти столбца смело включаем в состав первичного ключа. Что мы, кстати, и сделаем в нашей учебной базе данных. А код страны в таблице регионов, разумеется, будет внешним ключом, ссылающимся на код страны в таблице стран. Аналогичным образом следует поступить, создавая справочную таблицу по районам регионов — добавить код района, включить его в состав первичного ключа. (Справедливости ради надо сказать, что вот этот пример не совсем точно описывает "естественный" ключ; в присваивании районам номеров чувствуется все-таки некоторая искусственность.)

- Для переменных данных лучше всего использовать искусственный первичный ключ, значение которого мы будем получать из генератора. Такой ключ особенно важно применять, если таблица содержит большое количество дочерних таблиц. Для дочерних, вспомогательных, таблиц уж точно нужно использовать искусственные первичные ключи и свои генераторы. Все это позволит сэкономить объем внешней памяти и улучшит производительность.

Разумеется, не всегда следует придерживаться этих рекомендаций, хотя лично мне они кажутся весьма разумными.

4.2. Проектирование таблиц

Сейчас мы займемся проектированием, а затем созданием таблиц нашей учебной базы данных.

Процесс проектирования предполагает составление списка таблиц, перечисление столбцов, входящих в состав таблиц, с заданием некоторых характеристик этих столбцов и установление взаимосвязей между таблицами.

Собственно создание таблицы происходит при выполнении оператора SQL `CREATE TABLE`, в котором описываются столбцы таблицы, их полные характеристики, задаются первичные и внешние ключи, устанавливаются другие ограничения отдельных столбцов и всей таблицы в целом.

4.2.1. Общая постановка задачи

Мы с вами собираемся создать фрагмент информационно-поисковой системы по организациям и людям. Наша база данных будет содержать справочные данные, которые изменяются достаточно редко, и оперативные данные — данные, необходимые пользователю для решения конкретных задач предметной области.

Основная цель разработки — так спроектировать базу данных, чтобы из нее можно было получать данные на основании разнообразных запросов, о которых прямо сейчас мы даже всего и не знаем. Напомню, что хорошо спроектированная реляционная база данных, где правильно выполнена нормализация таблиц, обладает таким замечательным свойством, как возможность получать по запросам различные данные в различных разрезах.

4.2.2. Справочные таблицы

В первую очередь для структуризации адресной части организации и человека мы введем таблицы: справочник стран, регионов страны и районов региона. Будем использовать справочник видов деятельности, чтобы можно было описывать основные виды деятельности организаций. Добавим также справочник организационно-правовых форм — для порядка. Введем справочник административных и уголовных правонарушений, чтобы учитывать недобросовестных граждан и вообще хулиганов.

Список полученных справочных таблиц и их названия, которые мы будем использовать в базе данных, представлены в табл. 4.1. Имена всех справочных таблиц мы начинаем с префикса `REF`.

Таблица 4.1. *Список справочных таблиц*

Имя	Название
`REFCTR`	Справочник стран
`REFREG`	Справочник регионов
`REFAREA`	Справочник районов региона
`REFACTIV`	Справочник видов деятельности
`REFFORMMORG`	Справочник организационно-правовых форм
`REFADMIN`	Справочник административных и уголовных правонарушений

В табл. 4.2—4.4 описывается структура справочных таблиц.

Таблица 4.2. *Объект REFCTR. Справочник стран*

Имя	Тип данных	Название
CODCTR	Строка	Код страны. Первичный ключ
NAME	Строка	Краткое название страны
FULLNAME	Строка	Полное название страны
CAPITAL	Строка	Название столицы
DESCR	BLOB	Дополнительное описание

Первичным ключом здесь будет код страны.

Таблица 4.3. *Объект REFREG. Справочник регионов*

Имя	Тип данных	Название
CODCTR	Строка	Код страны. Внешний ключ. Входит в состав первичного ключа
CODREG	Строка	Код региона. Входит в состав первичного ключа
NAMEREG	Строка	Название региона
CENTER	Строка	Центр региона
DESCR	BLOB	Дополнительное описание

Здесь первичный ключ — код страны и код региона. Внешний ключ, ссылающийся на таблицу стран, разумеется, код страны.

Код региона в этой таблице является достаточно естественным столбцом. Номера регионов России получаются последовательным присваиванием номеров субъектам Федерации, указанным в Конституции РФ. Для США каждый штат имеет свой стандартный двухбуквенный код.

Таблица 4.4. *Объект REFAREA. Справочник районов*

Имя	Тип данных	Название
CODCTR	Строка	Код страны. Входит в состав первичного ключа. Входит в состав внешнего ключа
CODREG	Строка	Код региона. Входит в состав первичного ключа. Входит в состав внешнего ключа
CODAREA	Строка	Код района. Входит в состав первичного ключа
NAMEAREA	Строка	Название района
CENTER	Строка	Центр района
DESCR	BLOB	Дополнительное описание

Первичным ключом здесь будет код страны, код региона и код района. Внешний ключ, ссылающийся на таблицу регионов, состоит из кода страны и кода региона.

Код района здесь имеет довольно искусственное происхождение. Районы каждого региона перечисляются в алфавитном порядке, а затем каждому району присваивается порядковый номер, начиная с 01.

Связь между этими тремя таблицами — чистая трехуровневая иерархия. К одной стране относятся несколько регионов, в каждом регионе — несколько районов. Иными словами, между страной и регионами существует отношение "один-ко-многим", такое же отношение имеется и между регионом и районами (рис. 4.1).

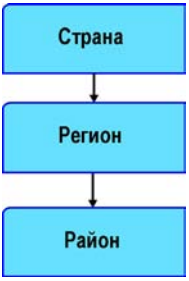


Рис. 4.1. Иерархическая связь между странами, регионами и районами

Остальные справочники (табл. 4.5—4.7) совсем простые; на данном уровне рассмотрения они имеют идентичную структуру. Между собой они не связаны, а отношения с оперативными данными мы рассмотрим чуть позже.

Таблица 4.5. *Объект* `REFACTIV`*. Справочник видов деятельности*

Имя	Тип данных	Название
`COD`	Строка	Код вида деятельности
`NAME`	Строка	Название вида деятельности

Таблица 4.6. *Объект* `REFFORMMORG`*. Справочник организационно-правовых форм*

Имя	Тип данных	Название
`COD`	Строка	Код организационно-правовой формы
`NAME`	Строка	Название организационно-правовой формы

Таблица 4.7. *Объект* `REFADMIN`*. Справочник административных и уголовных правонарушений*

Имя	Тип данных	Название
`COD`	Строка	Код правонарушения
`NAME`	Строка	Название правонарушения

В каждой из этих трех таблиц первичным ключом будет код — естественный первичный ключ. Его значения будут вводиться пользователем. Этим пользователем будем мы с вами, а точнее, на первое время я — скрипты содержат все необходимые значения для используемых нами фрагментов этих справочников.

4.2.3. Оперативные таблицы

В оперативных данных (табл. 4.8) нашей предметной области в первую очередь выделяются две так сказать системообразующие таблицы — список организаций и список людей. Остальные таблицы будут иметь связующий, вспомогательный характер.

Таблица 4.8. *Список оперативных таблиц*

Имя	Название
`organization`	Список организаций
`PEOPLE`	Список людей
`ORGACTIV`	Виды деятельности организации
`STAFF`	Сотрудники организации
`PEOPLEADMIN`	Правонарушения людей

Прежде чем описывать структуру этих таблиц, давайте рассмотрим, какие у них между собой, а также между ними и справочными таблицами должны существовать отношения.

Виды деятельности организации. Ясно, что у одной организации может быть множество видов деятельности. Например, организация может выпускать самолеты и при этом изготавливать товары народного потребления. Точно так же мы понимаем, что одним и тем же видом деятельности может заниматься множество различных организаций. То есть здесь мы имеем связь "многие-ко-многим", что опять же удобно и наглядно изобразить графически (рис. 4.2). Здесь нам и понадобилась третья, связующая, таблица `ORGACTIV`.

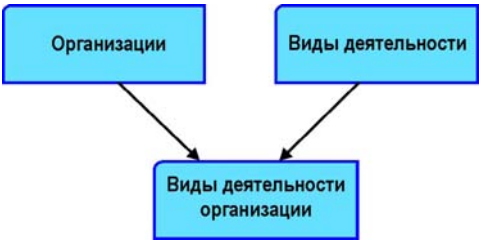


Рис. 4.2. Связь между организациями и видами деятельности

Точно такую же структуру имеют и отношения между людьми, справочником административных и уголовных правонарушений и хулиганствами людей — один человек может множество "хулиганств нарушать", одно и то же административное нарушение может применяться к множеству людей.

Между организациями, людьми и персоналом организации существует такая же связь — в организации может работать много людей, и один человек, в принципе, может работать в нескольких организациях.

В таблицу организации `ORGANIZATION` мы поместим только те столбцы, которые нам будут нужны для связей, а также для иллюстрации вариантов выборки строк на основании сложных условий поиска.

Первичный ключ в таблице организаций у нас будет, конечно, искусственным. Для него введем столбец, имеющий числовое значение. Чуть позже создадим генератор для получения уникальных значений этого первичного ключа. Пусть будет присутствовать краткое название организации. Для задания организационно-правовой формы добавим столбец, содержащий ссылку на код из справочника. Это будет внешний ключ. Довольно сложный момент — проектирование адресной части. В слабо структурированной базе данных адрес организации другие разработчики просто бы задали в виде одной достаточно длинной строки. Мы же, с учетом того, что у нас есть соответствующие справочники (и глубокие познания в области реляционных баз данных), выполним структурирование адреса.

Для адресной части нам, во-первых, нужен признак местоположения организации — областной (региональный) центр, районный центр или район (какая-нибудь деревушка в глубинке). Нужны код страны, код региона и код района, чтобы можно было получать соответствующие названия из справочников. И, наконец, строковый столбец, куда можно помещать название улицы, номер дома, номер офиса и др.

Здесь при проектировании возникает соблазн поместить в эту таблицу, например, название страны. Однако это будет нарушением третьей нормальной формы, которая требует, чтобы ни один неключевой столбец не зависел от другого неключевого столбца. Страну при выборке данных мы всегда при

необходимости сможем получить из справочника стран на основании значения кода страны в таблице организаций.

Более обоснован другой соблазн. При выборке данных из таблицы организаций нам нужно отображать либо название центра региона (областного центра), если организация располагается в центральном городе региона, либо название региона, если организация находится в районном центре или в той самой деревушке. Хотелось бы ввести в запись соответствующий столбец или даже два столбца, что также нарушает третье правило нормализации. Этого делать мы не будем.

> ***Замечание***
>
> Должен признаться, в одной из своих ранних разработок, в справочно-поисковой системе с красивым названием Business-Pilot® (лоцман в мире бизнеса), я так и поступил — поместил в запись организации и имя центра региона, и название региона. В результате база данных при большом количестве организаций становилась очень большой, однако, честно скажу, в некоторых фрагментах программы я получил выигрыш в плане функциональности. Использование хранимых процедур в этом случае (см. следующий абзац) решает все проблемы.

Забегая вперед, скажу, что, к сожалению, *декларативными* средствами SQL (с помощью только операторов создания таблиц и оператора выборки данных `SELECT`) мы задачу отображения разных названий не решим. Дальше мы с вами напишем хранимую процедуру, которая с блеском разрешит нашу проблему. В хранимой процедуре, как и в триггерах, мы можем уже использовать и *императивные* средства — т. е. средства проверки некоторых условий и выполнения соответствующих действий на основании результатов такой проверки.

В таблицу по людям `PEOPLE` мы поместим имя, фамилию, отчество и дату рождения. Кроме того, мы разместим там ссылки (внешние ключи) на родителей этого человека, а также на супруга: для женщины это будет ссылка на человека мужского пола, для мужчины — на человека женского пола. Необходимые проверки мы реализуем, забегая немного вперед, с помощью триггеров. Первичный ключ у нас также будет искусственным.

В табл. 4.9—4.13 описывается структура оперативных таблиц.

Таблица 4.9. *Объект* `ORGANIZATION`*. Список организаций*

Имя	Тип данных	Название
`COD`	Число	Код — искусственный первичный ключ
`CODCTR`	Строка	Код страны. Входит в состав внешнего ключа
`CODREG`	Строка	Код региона. Входит в состав внешнего ключа
`CODAREA`	Строка	Код района. Входит в состав внешнего ключа
`LOCATION`	Строка	Признак адреса: 0 — областной (региональный) центр; 1 — районный центр; 2 — район
`NAME`	Строка	Краткое название организации
`CODFORMORG`	Строка	Код организационно-правовой формы

Таблица 4.10. *Объект* `PEOPLE`*. Список людей*

Имя	Тип данных	Название
`COD`	Число	Код — искусственный первичный ключ
`NAME1`	Строка	Имя
`NAME2`	Строка	Отчество
`NAME3`	Строка	Фамилия
`BIRTHDAY`	Дата	Дата рождения
`SEX`	Строка	Пол: 0 — мужской; 1 — женский
`FULLNAME`	Строка	Фамилия, имя и отчество. Вычисляемый столбец
`CODMOTHER`	Число	Код матери. Внешний ключ
`CODFATHER`	Число	Код отца. Внешний ключ
`CODOTHERHALF`	Число	Код супруга (мужа или жены)

Обратите внимание на столбец `FULLNAME`. Его мы используем только для иллюстрации применения вычисляемых столбцов. Данные таких столбцов не хранятся в базе, а вычисляются при выборке строк. На самом деле такой столбец для наших задач обработки данных вовсе не нужен.

Кажется, я говорил вам, что в таблице по людям мы будем использовать структурированный адрес так же, как и в таблице по организациям. Похоже, я вас обманул. В реальной системе мы бы так и сделали. Здесь же в познавательных целях нам достаточно освоить работу с таким адресом только в организациях.

Три внешних ключа (код матери, код отца и код супруга) ссылаются на *ту же самую* таблицу. О внешних ключах мы подробно поговорим чуть позже. В этой таблице мы проиллюстрируем замечательную возможность внешнего ключа ссылаться на ту же самую таблицу. Факт не столько и не только странный (по крайней мере, на первый взгляд), сколько очень полезный в реальных системах.

Таблица 4.11. *Объект* `ORGACTIV`*. Виды деятельности организации*

Имя	Тип данных	Название
`COD`	Число	Код — искусственный первичный ключ
`CODACT`	Строка	Код вида деятельности из справочника. Внешний ключ
`CODORG`	Число	Код организации. Внешний ключ
`TEXT`	Строка	Некоторое описание вида деятельности

В табл. 4.11, казалось бы, можно было использовать код вида деятельности и код организации в качестве составного первичного ключа. Однако в этом случае каждый вид деятельности мог бы встречаться для одной организации не более одного раза, поскольку требуется уникальность значений первичного ключа. В нашем случае в организации один вид деятельности может появиться произвольное количество раз с различной описательной частью (столбец `TEXT`). И это правильно.

Таблица 4.12. *Объект* `STAFF`*. Сотрудники организации*

Имя	Тип данных	Название
`COD`	Число	Код — искусственный первичный ключ
`CODPEOPLE`	Число	Код человека из таблицы людей. Внешний ключ
`CODORG`	Число	Код организации. Внешний ключ
`DUTIES`	Строка	Должность
`SALARY`	Число	Должностной оклад

Здесь (табл. 4.12) менее очевидны преимущества использования искусственного первичного ключа — было бы несколько странно то, что один и тот же человек работает сразу в нескольких должностях в одной и той же организации. Тем не менее и здесь имеет смысл использовать такой ключ, например, в том случае, если в базе данных предполагается хранить историю приема на работу и увольнение сотрудников за длительный период. Человек мог поступать работать в организацию, увольняться, а затем опять приходить туда ра-

ботать. В этом случае в структуру следует ввести дату приема на работу и дату увольнения.

Таблица 4.13. *Объект* `PEOPLEADMIN`*. Правонарушения людей*

Имя	Тип данных	Название
`COD`	Число	Код — искусственный первичный ключ
`CODPEOPLE`	Число	Код человека из таблицы. Внешний ключ
`CODADMIN`	Число	Код правонарушения. Внешний ключ
`DATEADMIN`	Дата	Дата совершения правонарушения

В табл. 4.13 уж точно нужен искусственный первичный ключ, поскольку — такова природа человека — ясно, что он может дважды наступить на одни и те же грабли: несколько раз совершить нехорошие действия.

4.2.4. Промежуточные итоги

То, что мы сейчас проделали с нашей базой данных, в литературе чаще всего называется *концептуальным* проектированием. Мы представили на концептуальном (содержательном) уровне структуру каждой таблицы, указав только список столбцов с их основной характеристикой типа данных и отметив, что столбцы включены в состав первичного и/или внешнего ключа. На этом уровне тип данных задается в следующих вариантах — число, строка, дата или `BLOB`. Более детальные характеристики не указываются, такие как вид числовых данных, размер строки и т. д.

Сейчас мы выполним уточнение концептуальной структуры и получим *логическую* структуру наших данных. Если результаты концептуального проектирования мы представляли в виде обычных таблиц (что весьма удобно для человека), то для описания логической структуры мы используем сразу язык SQL, его подмножество DDL. Вообще говоря, и для описания логической структуры имеет смысл использовать вначале табличную форму, поскольку в операторах SQL для задания типа данных столбцов мы будем сразу ссылаться на созданные нами домены, что бывает не всегда достаточно наглядным. При разработке больших программных систем я так и поступаю. Логическая структура вначале представляется в табличной форме. В таком виде ее удобно показывать заказчику, выполнять необходимые согласования.

В любом случае создание таблиц, да и других объектов базы данных с использованием языка SQL, должно быть подробно документировано. Для каждого столбца таблицы, используя комментарии (как и в языке C, это `/*` и `*/`), нужно указать на человеческом языке название, а может быть и другие характеристики. Столбцы-признаки, которые могут иметь небольшое количество

значений, должны сопровождаться комментарием о смысле каждого значения. Например, для столбца, задающего пол человека, мы указываем, что 0 — это мужской пол, 1 — женский.

4.3. Контрольная работа

Предлагаю прямо сейчас, когда вы получили серьезные знания, умения и навыки в области концептуального проектирования реляционных баз данных, закрепить эти знания и выполнить три простенькие разработки для различных областей человеческой деятельности. Это мы можем рассматривать в качестве контрольной работы.

Если же вам совсем не хочется перейти от "чистых реляций" к реальной жизни, не выполняйте этих действий, но мои ответы все же посмотрите в конце этой главы.

4.3.1. Учебные дисциплины

В образовательных заведениях существуют различные учебные дисциплины. В каждой дисциплине можно выделить более мелкие части — разделы. Каждый раздел, в свою очередь, состоит из некоторого количества тем. Если вам когда-либо в своей жизни приходилось разрабатывать рабочую программу к учебному курсу и календарно-тематический план проведения занятий, вы увидите здесь что-то родное.

Предлагаю немного подумать и разработать концептуальную структуру базы данных, частью которой будут являться и данные по дисциплинам, разделам, темам. Вам нужно решить, какие должны существовать таблицы, какие поля должны входить в состав каждой таблицы. После этого следует принять решения по первичным и внешним ключам, если такие нужны, и описать структуру каждой таблицы по той форме, которую мы с вами только что использовали. Приступайте. Я подожду ваших результатов.

4.3.2. Сотрудники организации

Поскольку, как я понимаю, вам такая деятельность понравилась, предлагаю выполнить еще одну разработку. Пусть имеются организации, в которых существуют структурные подразделения (например, отделы), а в них работают сотрудники. Разработайте концептуальную структуру такой базы данных. Здесь мы не рассматриваем вариант, когда один и тот же человек работает в разных организациях.

4.3.3. Движение товаров на складах

Для кого-то подобная задача может показаться неинтересной. Однако, на то мы и специалисты, чтобы уметь применять свои обширные знания по реляционным базам данных для решения задач в различных областях человеческой деятельности.

Каждое производственное предприятие для выпуска своей продукции использует какой-то фиксированный круг исходных материалов, комплектующих изделий. Материалы от поставщиков приходят на склады предприятия, откуда они отпускаются в производство.

Предлагаю создать таблицы для хранения соответствующих данных, чтобы потом можно было выполнять необходимый учет, формировать ведомости движения материалов, учитывать остатки на складах.

Руководящие идеи здесь такие. Должна существовать таблица, являющаяся справочником материалов. Отдельная таблица должна представлять список складов предприятия, нужна таблица, описывающая материал на каждом складе, и таблица движения материалов по каждому складу.

Замечание

Эти задачи проектирования я предлагаю решать моим студентам. Свои решения не показывайте им — пусть немного подумают и поработают самостоятельно.

Мои версии подобных разработок представлены в конце этой главы. Сравните ваши результаты с моими, поставьте оценки — себе и мне.

4.4. Синтаксические конструкции

Нам нужно рассмотреть синтаксис операторов DDL, используемых для создания и изменения таблиц и генераторов. Вначале обсудим работу с одним из самых простых объектов базы данных — генератором.

4.4.1. Работа с генераторами

Для создания генератора используется оператор `CREATE GENERATOR`:

```
CREATE GENERATOR <имя генератора>;
```

Имя генератора должно быть уникальным среди имен всех генераторов базы данных. Оператор создает генератор с этим именем и устанавливает его значение в ноль.

Если вам вдруг понадобится изменить значение генератора (но я вам этого делать не советую), вы можете использовать следующий оператор:

```
SET GENERATOR <имя генератора> TO <целое>;
```

Здесь указанное целое помещается в текущее значение генератора.

Для удаления существующего генератора из базы данных используется оператор:

```
DROP GENERATOR <имя генератора>;
```

> ***Замечание***
>
> Странно, но факт — в ранних версиях InterBase такого оператора не существовало. Нет его и в документации по InterBase 2007. В "Справочнике по языкам" (Language Reference) так и написано (разумеется, в переводе на русский): "Не существует оператора "drop generator". Для удаления генератора удалите его из системной таблицы. Например:
>
> ```
> DELETE FROM RDB$GENERATOR WHERE RDB$GENERATOR_NAME = 'EMPNO_GEN';"
> ```
>
> Удаление генератора напрямую из системных таблиц никак нельзя одобрить. В Firebird 1.5.3 этот оператор нормально работает. Я проверял.

Чтобы получить значение генератора и увеличить это значение на какое-то число, используется функция `GEN_ID`. Ее синтаксис:

```
GEN_ID(<имя генератора>, <шаг>);
```

Здесь указывается имя созданного ранее генератора, а шаг — целое число со знаком, на которое изменяется значение генератора. При обращении к этой функции сначала изменяется значение генератора, а затем функция возвращает новое, измененное, значение. Конечно, самым естественным вариантом является задание шага, равного единице, хотя есть некоторые изощренные способы использовать и другие значения, например, отрицательные, когда значение генератора уменьшается с каждым новым обращением к нему.

В Firebird 1.5.3 генератор может принимать значения от -2^{63} до $2^{63}-1$. Если перевести эти значения на человеческий язык, то получается необычайно большой диапазон: от –9,223,372,036,854,775,808 до +9,223,372,036,854,775,807. Несколько ранее мы с вами уже пытались произнести названия подобных чисел, используя такие слова, как "квинтиллион" и "квадриллион".

Генераторы используются для получения последовательности уникальных целых чисел, т. е. в последовательности одно и то же число не может повториться дважды — если только в процессе работы системы вы не переустанавливали вручную это значение, используя оператор `SET GENERATOR` (чего я вам делать, как вы помните, не советовал). Это свойство уникальности значений генераторов используется для получения значений искусственных первичных ключей.

4.4.2. Синтаксис создания таблицы

Таблицы создаются с помощью оператора CREATE TABLE. Его синтаксис довольно сложный. Рассмотрим несколько упрощенный вариант.

```
CREATE TABLE <имя таблицы>
(<определение столбца>
  [, <определение столбца> | <ограничение таблицы>] ...);
```

В этом описании мы видим, что таблица может содержать как минимум один столбец и произвольное количество ограничений таблицы. Что такое ограничение (constraint), мы скоро разберем подробно. Список столбцов и ограничений таблицы заключен в круглые скобки. Элементы в списке, как обычно в различных языках программирования, отделяются друг от друга запятыми. Сейчас мы рассмотрим синтаксис определения столбца таблицы и определения ограничения таблицы. Чисто количественно эти синтаксические конструкции довольно объемны, однако по смыслу очень понятны и естественны. Не пугайтесь, пожалуйста, кажущейся сложности синтаксиса, а не спеша рассмотрите приводимые описания и примеры использования.

4.4.2.1. Определение столбца таблицы

В определении столбца таблицы частично повторяется синтаксис, используемый при создании домена. Это понятно, поскольку оба описывают один и тот же объект. Синтаксис определения столбца:

```
<имя столбца> {<тип данных> | COMPUTED [BY] (<выражение>) | <имя домена>}
  [DEFAULT {<литерал> | NULL | USER}]
  [NOT NULL]
  [<ограничение столбца>]
  [COLLATE <порядок сортировки>]
```

Имя столбца должно быть уникальным среди имен данной (только данной) таблицы. Длина имени не должна превышать 31 символа.

Ограничение на количество столбцов в одной таблице зависит от типа данных существующих столбцов. Общее правило — размер строки таблицы не должен превышать 64 Кбайт. При этом столбец BLOB (точнее, указатель на сегмент BLOB) в строке таблицы занимает всего восемь байтов.

Для столбца мы можем задать либо тип данных, либо указать, что он является вычисляемым (COMPUTED BY), либо сослаться на ранее созданный домен, который уже содержит все нужные характеристики столбца или только некоторую часть таких характеристик.

Синтаксис задания типа данных в точности соответствует синтаксису, который мы описали для создания домена.

```
<тип данных> ::=
{SMALLINT | INTEGER | BIGINT | FLOAT | DOUBLE PRECISION | BOOLEAN}
```

```
| {DATE | TIME | TIMESTAMP}
| {DECIMAL | NUMERIC} [(<n> [,<m>])]
| {CHAR | VARCHAR} [(<n>)] [CHARACTER SET <набор символов>]
| BLOB [SUB_TYPE <n>] [SEGMENT SIZE <n>] [CHARACTER SET <набор символов>]
```

Предложение `COMPUTED BY` указывает, что столбец является вычисляемым. Значение такого столбца не хранится в базе данных, а вычисляется при выборке данных. Это может быть арифметическое, строковое или другое выражение. Мы чуть позже используем для вычисляемого столбца операцию конкатенации (объединения) трех строк в таблице людей, а в таблице персонала создадим вычисляемый столбец, для которого будут использованы арифметические вычисления.

Вместо указания типа данных мы можем задать имя домена, чьи характеристики будут присвоены данному столбцу.

Предложение `DEFAULT`, как и в домене, задает значение по умолчанию для столбца, которое будет присвоено этому столбцу, если при добавлении новой строки в данную таблицу в операторе `INSERT` не будет указано значение для этого столбца. Значением по умолчанию может быть литерал (константа соответствующего вида), пустое значение (`NULL`) или имя пользователя, соединенного в момент добавления строки в таблицу с этой базой данных (`USER`). Еще раз напомню, что значение по умолчанию применяется только при создании новой строки, но не при внесении изменений в существующую строку оператором `UPDATE`.

Предложение `NOT NULL` задает запрет на присваивание пустого значения этому столбцу. Для столбцов, входящих в состав первичного ключа, обязательно должно быть указано это предложение либо в определении столбца, либо в определении домена, на который ссылается этот столбец. Это необходимо явно задать даже в том случае, когда столбцу запрещено иметь пустое значение на основании, например, ограничения `CHECK`.

Приведем полный синтаксис ограничения столбца, хотя на самом деле в наших примерах мы будем использовать не ограничения столбца, а ограничения таблицы (см. замечание в конце этого раздела). По этой причине я не хочу давать подробных объяснений по каждой конструкции, кроме ограничения `CHECK`. Посмотрев на ограничения таблицы чуть ниже по тексту, вы всегда при необходимости разберетесь и с этими простыми элементами синтаксиса.

```
<ограничение столбца> ::=
  { PRIMARY KEY
  | UNIQUE
  | REFERENCES <имя таблицы> (<столбец>)
     [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
     [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<условие столбца>)}
```

Условие столбца в ограничении CHECK очень похоже на ограничение домена в описании домена. Вот несколько упрощенный синтаксис:

```
<условие столбца> ::=
  <значение> <оператор> <значение>
  | <значение> [NOT] BETWEEN <значение> AND <значение>
  | <значение> [NOT] LIKE <значение>
  | <значение> [NOT] IN (<значение> [, <значение> ...])
  | <значение> IS [NOT] NULL
  | <значение> [NOT] CONTAINING <значение>
  | <значение> [NOT] STARTING [WITH] <значение>
  | (<условие столбца>)
  | NOT <условие столбца>
  | <условие столбца> OR <условие столбца>
  | <условие столбца> AND <условие столбца>
```

Здесь "значением" может быть имя столбца таблицы (по-хорошему, только имя данного столбца, к которому относится предложение CHECK, хотя допустимы и имена других столбцов, *предшествующих в описании таблицы данному столбцу*), константа (числовая, строковая, дата, время), выражение или USER — имя текущего пользователя, подключенного к базе данных.

Из описания синтаксиса я убрал слишком сложные конструкции, которые не только запутывают, но и очень вредны с точки зрения производительности создаваемой программной системы.

Последним предложением в определении столбца является необязательное предложение COLLATE, которое задает порядок сортировки значений данного столбца. Может использоваться только для строковых типов данных (CHAR и VARCHAR).

4.4.2.2. Определение ограничения таблицы

Рассмотрим теперь ограничение таблицы.

```
<ограничение таблицы> ::=
[CONSTRAINT <имя ограничения>]
  {{PRIMARY KEY | UNIQUE} (<столбец> [, <столбец>] ...)
  | FOREIGN KEY (<столбец> [, <столбец> ...])
     REFERENCES <имя таблицы> (<столбец> [, <столбец>] ...)
     [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
     [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
  | CHECK (<условие таблицы>)}
```

Мы можем задать имя ограничения (предложение CONSTRAINT), что очень рекомендуется сделать, чтобы при отладке наших программ видеть, какие именно нарушаются ограничения, и чтобы иметь возможность вносить изме-

нения в существующие ограничения. (Честно говоря, лично я не всегда придерживаюсь этой мудрой рекомендации.)

Есть четыре ограничения таблицы — первичный ключ (предложение `PRIMARY KEY`), уникальный ключ (предложение `UNIQUE`), внешний ключ (предложение `FOREIGN KEY`) и ограничение на значение столбца/столбцов данной таблицы (предложение `CHECK`).

Имена столбцов таблицы, входящих в состав первичного ключа, заключаются в круглые скобки и отделяются друг от друга запятыми, как и в любом нормальном списке. Даже если первичный ключ состоит только из одного столбца, его имя также заключается в круглые скобки.

Точно так же задается уникальный ключ. Для каждого первичного и уникального ключа автоматически создается индекс, который позволяет ускорить выборку необходимых строк в таблицах.

Внешний ключ задается предложением `FOREIGN KEY`, где опять же в круглых скобках перечисляются имена столбцов данной таблицы, которые входят в состав этого внешнего ключа. После ключевого слова `REFERENCES` (что в переводе означает "ссылается") задается имя таблицы, на первичный (`PRIMARY`) или уникальный (`UNIQUE`) ключ которой ссылается данный внешний ключ, и в круглых скобках — список столбцов, входящих в состав первичного или уникального ключа той таблицы, на которую ссылается наш внешний ключ.

Таблица, на которую ссылается внешний ключ, часто называется *родительской* (parent) или главной таблицей, а таблица, ссылающаяся на родительскую таблицу — *дочерней* (child) или подчиненной.

Общее правило использования связки "внешний ключ/первичный (уникальный) ключ". Либо в родительской таблице должна присутствовать строка, содержащая значение первичного (уникального) ключа, равное значению внешнего ключа дочерней таблицы, либо значение внешнего ключа (всех столбцов, входящих в состав внешнего ключа), должно быть пустым — т. е. иметь значение `NULL`.

Еще раз обратите внимание на тот важный факт, что внешний ключ может ссылаться на первичный или уникальный ключ *той же самой* таблицы. В нашем примере в таблице, содержащей сведения о людях, будет аж три внешних ключа, ссылающихся на первичный ключ этой же таблицы.

Важный момент — таблица, на которую ссылается внешний ключ, уже должна быть описана в базе данных.

Для внешнего ключа система также создает индекс.

Необязательная фраза `ON DELETE` задает, что должно выполняться, если удаляется соответствующая строка родительской таблицы. Можно указать один из следующих вариантов:

- `NO ACTION` — ничего не происходит. Это вариант по умолчанию, т. е., если вы для внешнего ключа не укажете предложение `ON DELETE`, никаких действий выполняться не будет. Это хорошее основание получать исключения базы данных практически на пустом месте. Вам самим придется как-то приводить в соответствие состояние строк дочерней таблицы после удаления (а точнее, перед удалением) строки родительской таблицы;
- `CASCADE` — все строки дочерней таблицы, содержащие внешние ключи, имеющие те же значения, что и первичный (уникальный) ключ удаленной строки родительской таблицы, будут автоматически удалены из базы данных. Чаще всего при проектировании баз данных используется именно этот вариант;
- `SET DEFAULT` — значения столбцов внешнего ключа в соответствующих строках дочерней таблицы устанавливаются в заданные для них значения по умолчанию;
- `SET NULL` — значения столбцов внешнего ключа в соответствующих строках дочерней таблицы устанавливаются в `NULL`. Тоже не самое плохое решение, хотя, скорее всего потом нужно будет как-то наводить порядок в базе данных.

Необязательная фраза `ON UPDATE` задает, что должно произойти, когда изменяется значение первичного (уникального) ключа родительской таблицы. Здесь варианты такие же, как и в случае удаления строки родительской таблицы (`NO ACTION`, `CASCADE`, `SET DEFAULT`, `SET NULL`). Самым естественным, разумеется, является вариант `CASCADE`, при котором все соответствующие внешние ключи в строках дочерней таблицы получают новое значение, равное измененному значению первичного ключа родительской таблицы.

Все эти изменения в дочерних таблицах выполняются с помощью системных триггеров, которые автоматически создаются сервером базы данных по результатам создания таблиц с подобными внешними ключами. До триггеров мы с вами еще доберемся несколько позже.

Предложение `CHECK` в описании таблицы сильно напоминает то же предложение в описании домена и полностью соответствует по форме такому предложению в описании ограничения столбца. При этом есть и некоторые отличия.

Далее представлен синтаксис предложения `CHECK` в описании таблицы. Опять же мы его довольно сильно упрощаем, и это упрощение только пойдет нам на пользу — мы сейчас не будем рассматривать очень сложные, "заумные" варианты, которые очень редко используются и расходуют слишком много ре-

сурсов вычислительной системы — как внешней и оперативной памяти, так и процессорного времени. В основном я убрал из описания синтаксиса конструкции, связанные с явным или неявным обращением через оператор SELECT к другим строкам этой же или другой таблицы.

```
<условие таблицы> ::=
  <значение> <оператор> <значение>
  | <значение> [NOT] BETWEEN <значение> AND <значение>
  | <значение> [NOT] LIKE <значение>
  | <значение> [NOT] IN (<значение> [, <значение> ...])
  | <значение> IS [NOT] NULL
  | <значение> [NOT] CONTAINING <значение>
  | <значение> [NOT] STARTING [WITH] <значение>
  | (<условие таблицы>)
  | NOT <условие таблицы>
  | <условие таблицы> OR <условие таблицы>
  | <условие таблицы> AND <условие таблицы>
```

Здесь <значение> — литерал, константа, выражение или имя одного из столбцов данной таблицы. В остальном все работает так же, как описано в доменах.

Если столбец основывается на домене, в котором уже присутствует ограничение CHECK, и для этого столбца также задается ограничение CHECK, то в результате для столбца таблицы устанавливается ограничение CHECK, являющееся конъюнкцией (логическая операция И) обоих логических выражений. Следует в подобных случаях быть аккуратными (равно как и в любых других случаях) и не задавать бессмысленных выражений.

Например, пусть был создан целочисленный домен:

```
CREATE DOMAIN D_TEST_CHECK AS INTEGER
  CHECK (VALUE IN (0, 1, 2, 3));
```

То есть задается ограничение на значение в виде целого числа между 0 и 3 включительно. Теперь пусть создается таблица, в которой присутствует столбец, основанный на этом домене, и для столбца также задается ограничение CHECK:

```
CREATE TABLE TABLE_TEST_CHECK
( ...
  COL_CHECK D_TEST_CHECK,
  ...
  CONSTRAINT TEST_CHECK
    CHECK (COL_CHECK IN (4, 5, 6, 7, 8));
```

В результате для значения столбца COL_CHECK будет с помощью логической операции конъюнкции сформировано следующее условие: это значение

должно находиться в пределах от 0 до 3 и *в то же время* в пределах от 4 до 8. Понятно, что не существует такого значения, которое удовлетворяло бы этому условию. В такую таблицу невозможно будет записать ни одной строки. Здесь даже не пройдет вариант, при котором этот столбец будет иметь пустое значение — `NULL`.

Замечание по синтаксису создания таблицы

Как мы с вами видели, ограничения можно задавать не только на уровне таблицы, но и на уровне одного столбца. Например, если в состав первичного ключа таблицы входит только один столбец, то при описании этого столбца можно было бы сразу и указать, что он является первичным ключом. В частности, при создании таблицы, являющейся справочником стран (см. в следующем разделе несколькими строками ниже), при описании столбца `CODCTR` можно было бы сразу записать:

```
( CODCTR   D_CHAR3 NOT NULL PRIMARY KEY,   /* Код страны */
  ...
```

Однако я этого не делаю, а выношу соответствующее именованное ограничение в конец описания таблицы:

```
( CODCTR   D_CHAR3 NOT NULL,   /* Код страны */
  ...
  CONSTRAINT PK_REFCTR PRIMARY KEY (CODCTR)
  ...);
```

Поверьте, это гораздо удобнее и нагляднее. Все ограничения таблицы, даже если некоторые из них относятся только к одному столбцу, у меня поименованы и собраны в одном месте. Много сложнее отыскивать нужные ограничения по всем строкам в операторе создания таблицы.

Существует еще один неплохой вариант задания ограничений. Вначале с помощью операторов `CREATE TABLE` создаются все таблицы. При этом ни для одной из них не описываются ограничения. После создания таблиц с помощью операторов `ALTER TABLE` в таблицы добавляются необходимые ограничения. Это хороший вариант, особенно в том случае, если у вас множество таблиц ссылаются друг на друга. Таблица, на которую осуществляется ссылка, уже должна существовать в базе данных. Заранее созданные таблицы без указания ограничений решают проблему задания правильного порядка создания таблиц. То есть в этом случае порядок совершенно произвольный.

4.5. Создание таблиц и генераторов

Напишем операторы SQL (а именно DDL) для создания таблицы стран (листинг 4.1). Прежде чем создавать таблицу, нужно установить диалект клиента (3), задать набор символов (`WIN1251`) и соединиться с базой данных.

Листинг 4.1. Создание таблицы стран

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
```

```
CONNECT 'D:\BestDatabase\Work.fdb'
USER 'WIZARD' PASSWORD 'master';
     /*** Справочник стран ***/
CREATE TABLE REFCTR
( CODCTR   D_CHAR3 NOT NULL,   /* Код страны */
  NAME     D_CHAR30,           /* Краткое название страны */
  FULLNAME D_CHAR60,           /* Полное название страны */
  CAPITAL  D_CHAR15,           /* Название столицы */
  DESCR    D_BLOB,             /* Дополнительное описание */
  CONSTRAINT PK_REFCTR PRIMARY KEY (CODCTR)
);
COMMIT;
```

Примечания, или комментарии, в SQL помещаются между символами `/*` и `*/`. Хочу напомнить — если у вас до сих пор не выработалась привычка документировать программные тексты, то либо срочно учитесь это делать, либо бросайте заниматься разработкой программных продуктов.

Здесь в описании столбцов мы не задаем тип данных, а ссылаемся на домены, которые мы с вами заблаговременно создали. Получилось не слишком наглядно, но привыкнуть можно. Например, при описании кода страны (`CODCTR`) мы ссылаемся на домен `D_CHAR3`. Следуя нашим соглашениям по именованию доменов видно, что этот домен описывает тип данных `CHAR(3)`. Поскольку этот столбец будет первичным ключом, мы задаем для него `NOT NULL`.

Сам факт, что этот столбец является первичным ключом, задается в ограничении таблицы, а не столбца, как мы с вами и договорились:

```
CONSTRAINT PK_REFCTR PRIMARY KEY (CODCTR)
```

Обратите внимание, что мы не забыли присвоить этому ограничению имя (по правде сказать, когда я первоначально написал этот фрагмент скрипта, то забыл это сделать — не говорите об этом никому!). Обычная практика присваивания имен ограничениям такая. Используется префикс, знак подчеркивания и имя таблицы. Префиксами могут быть:

- `PK` — первичный ключ;
- `FK` — внешний ключ (если в таблице несколько внешних ключей, то просто используем индекс: `FK1`, `FK2` и т. д.);
- `UK` — уникальный ключ (тоже при необходимости могут быть использованы индексы);
- `CH` — ограничение `CHECK` (при необходимости с индексами).

С этой таблицей все просто. Более интересны для нас таблицы регионов и районов (листинг 4.2).

Листинг 4.2. Создание таблицы регионов

```
    /*** Справочник регионов ***/
CREATE TABLE REFREG
( CODCTR   D_CHAR3 NOT NULL,   /* Код страны */
  CODREG   D_CHAR2 NOT NULL,   /* Код региона */
  NAMEREG  D_CHAR40,           /* Название региона */
  CENTER   D_CHAR25,           /* Название центра региона */
  DESCR    D_BLOB,             /* Дополнительное описание */
  CONSTRAINT PK_REFREG PRIMARY KEY (CODCTR, CODREG),
  CONSTRAINT FK_REFREG
    FOREIGN KEY (CODCTR) REFERENCES REFCTR (CODCTR)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
COMMIT;
```

Первичный ключ в этой таблице, как мы с вами и решили, состоит из двух столбцов — кода страны и кода региона. Это обеспечивает нам уникальность значения. Оба описываются с предложением `NOT NULL`.

Внешний ключ мы определили следующим образом:

```
CONSTRAINT FK_REFREG
  FOREIGN KEY (CODCTR) REFERENCES REFCTR (CODCTR)
    ON DELETE CASCADE
    ON UPDATE CASCADE
```

Присвоили имя этому ограничению (молодцы — не забыли). Внешний ключ состоит из одного столбца, кода страны. Он ссылается на код страны в справочнике стран. Мы также указали, что при удалении страны должны быть удалены все ее регионы. При изменении кода страны в таблице стран во всех строках регионов этой страны также должно измениться значение кода страны. Все разумно и просто.

В таблице районов все выполняется аналогичным образом с учетом того, что эта таблица находится на третьем уровне иерархии. В результате количество столбцов для первичного и внешнего ключей увеличивается на единицу (листинг 4.3).

Листинг 4.3. Создание таблицы районов

```
    /*** Справочник районов ***/
CREATE TABLE REFAREA
( CODCTR   D_CHAR3 NOT NULL,   /* Код страны */
  CODREG   D_CHAR2 NOT NULL,   /* Код региона */
  CODAREA  D_CHAR2 NOT NULL,   /* Код района */
  NAMEAREA D_CHAR35,           /* Название района */
```

```
  CENTER   D_CHAR25,           /* Название центра района */
  DESCR    D_BLOB,             /* Дополнительное описание */
  CONSTRAINT PK_REFAREA PRIMARY KEY (CODCTR, CODREG, CODAREA),
  CONSTRAINT FK_REFAREA
    FOREIGN KEY (CODCTR, CODREG) REFERENCES REFREG (CODCTR, CODREG)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
COMMIT;
```

При создании справочника районов возникает вопрос, а не следует ли сюда ввести внешний ключ (код страны), ссылающийся на таблицу стран. На самом деле это было бы излишним. Наличие внешних ключей в таблице регионов и в таблице районов решает все проблемы.

Если происходит изменение первичного ключа (кода страны) в таблице стран, то при наличии варианта `ON UPDATE CASCADE` автоматически (с использованием созданных системой триггеров) изменяются значения кода этой страны во всех соответствующих строках в таблице регионов. Однако, как только изменился код страны в первом регионе, тут же начинается изменение кодов страны в соответствующих строках таблицы районов. Здесь происходят именно *каскадные* изменения. В результате будут выполнены все необходимые изменения во всех подчиненных таблицах.

Запустите IBExpert или isql, соединитесь с базой данных и создайте вручную эти три таблицы. Если вы что-то введете неправильно, то получите сообщение об ошибке. Внесите исправления, удалите те таблицы, которые все-таки были созданы (см. следующий раздел), и вновь выполните ваши операторы.

Остальные справочные таблицы простые. Вы можете посмотреть их описание в скрипте CreateTables.sql.

Сейчас рассмотрим некоторые оперативные таблицы и интересные связи (листинг 4.4).

Листинг 4.4. Создание таблицы для хранения данных по людям

```
    /*** Список людей ***/
CREATE TABLE PEOPLE
( COD       D_INTEGER NOT NULL,  /* Код человека */
  NAME1     D_CHAR15,            /* Имя */
  NAME2     D_CHAR15,            /* Отчество */
  NAME3     D_CHAR20,            /* Фамилия */
  BIRTHDAY  D_DATE,              /* Дата рождения */
  SEX       D_CHAR1 DEFAULT '0', /* Пол: */
                                 /* 0 – мужской, */
```

```
                                  /* 1 – женский. */
  FULLNAME  COMPUTED BY           /* Вычисляемый столбец */
    (NAME3 || ' ' || NAME1 || ' ' || NAME2),
  CODMOTHER    D_INTEGER,         /* Ссылка на мать */
  CODFATHER    D_INTEGER,         /* Ссылка на отца */
  CODOTHERHALF D_INTEGER,         /* Ссылка на супруга */
  CONSTRAINT PK_PEOPLE PRIMARY KEY (COD),
  CONSTRAINT CH_PEOPLE CHECK (SEX IN ('0', '1')),
  CONSTRAINT FK1_PEOPLE
    FOREIGN KEY (CODMOTHER) REFERENCES PEOPLE (COD)
      ON DELETE SET NULL,
  CONSTRAINT FK2_PEOPLE
    FOREIGN KEY (CODFATHER) REFERENCES PEOPLE (COD)
      ON DELETE SET NULL,
  CONSTRAINT FK3_PEOPLE
    FOREIGN KEY (CODOTHERHALF) REFERENCES PEOPLE (COD)
      ON DELETE SET NULL
);
CREATE GENERATOR GEN_PEOPLE;
COMMIT;
```

Посмотрим на вычисляемый столбец:

```
FULLNAME COMPUTED BY          /* Вычисляемый столбец */
  (NAME3 || ' ' || NAME1 || ' ' || NAME2),
```

Столбец `FULLNAME` не хранится в базе данных, а вычисляется при выборке данных. Его значением становится строка, содержащая фамилию, имя и отчество, разделенные пробелами. Здесь мы использовали операцию конкатенации строк.

Для пола человека устанавливается ограничение `CHECK`, позволяющее поместить в столбец `SEX` только символы 0 или 1:

```
CONSTRAINT CH_PEOPLE CHECK (SEX IN ('0', '1')),
```

Для этого столбца мы также задали значение по умолчанию с помощью предложения `DEFAULT '0'`.

Столбцы `CODMOTHER`, `CODFATHER` и `CODOTHERHALF` являются внешними ключами, ссылающимися на ту же самую таблицу. (В английском языке нет эквивалента русскому слову "супруг". По этой причине название третьего внешнего ключа в таблице можно рассматривать как "код другой половины".) Обратите внимание на два момента в описании этих внешних ключей.

- Мы не задаем ключевых слов `ON UPDATE` по той простой причине, что для строки человека не может изменяться первичный ключ. Его мы автомати-

чески получаем из генератора при помещении новой строки в базу данных, и это значение не меняется, пока строка существует в базе данных.

- При удалении (`ON DELETE`) соответствующей строки родительской таблицы (это та же самая таблица, что и дочерняя, но строки *разные*) значение внешнего ключа устанавливается в `NULL`. То есть в этом случае мы просто разрываем связь ребенок/родитель или связь между супругами, но не удаляем нашу запись, что было бы совершенно неправильным решением.

Здесь мы также сразу создали и генератор для получения значения первичного ключа этой таблицы. Генератору задали незамысловатое имя `GEN_PEOPLE`. Это также одно из соглашений по именованию программных объектов, которое применяется *правильными* программистами (то есть вами и мною).

Давайте посмотрим на таблицу организаций (листинг 4.5).

Листинг 4.5. Создание таблицы для хранения данных по организациям

```
    /*** Список организаций ***/
CREATE TABLE ORGANIZATION
( COD         D_INTEGER NOT NULL,    /* Код организации */
  CODCTR      D_CHAR3,               /* Код страны */
  CODREG      D_CHAR2,               /* Код региона */
  CODAREA     D_CHAR2,               /* Код района */
  LOCATION    D_CHAR1 DEFAULT '0',   /* Признак адреса: */
                                     /* 0 – областной центр, */
                                     /* 1 – районный центр, */
                                     /* 2 – район. */
  NAME        D_CHAR60,              /* Название организации */
  CODFORMORG D_CHAR2 DEFAULT '00', /* Организационно-правовая форма */
  CONSTRAINT PK_ORGANIZATION PRIMARY KEY (COD),
  CONSTRAINT CH_ORGANIZATION CHECK (LOCATION IN ('0', '1', '2')),
  CONSTRAINT FK1_ORGANIZATION
    FOREIGN KEY (CODCTR, CODREG) REFERENCES REFREG (CODCTR, CODREG)
      ON DELETE SET NULL
      ON UPDATE CASCADE,
  CONSTRAINT FK2_ORGANIZATION
    FOREIGN KEY (CODFORMORG) REFERENCES REFFORMORG (COD)
      ON DELETE SET NULL
      ON UPDATE CASCADE
);
CREATE GENERATOR GEN_ORGANIZATION;
COMMIT;
```

Для признака адреса мы задаем ограничение `CHECK`, позволяющее поместить в столбец `LOCATION` только символы 0, 1 или 2:

```
CONSTRAINT CH_ORGANIZATION CHECK (LOCATION IN ('0', '1', '2'))
```

Довольно сложный на первый взгляд вопрос создания внешних ключей для этой таблицы. С одной стороны, мы должны создать два внешних ключа для связки адресной части со справочником регионов и со справочником районов. С другой стороны, если местоположением организации является областной центр, то код района должен иметь пустое значение (NULL), а это разрушает всю нашу замечательную картину мира — *все* столбцы, входящие в состав внешнего ключа, должны иметь какое-то непустое значение либо *все* эти столбцы должны содержать значение NULL.

В результате мы оставляем только один внешний ключ, ссылающийся на таблицу регионов, поскольку организация располагается в каком-то конкретном регионе:

```
CONSTRAINT FK1_ORGANIZATION
  FOREIGN KEY (CODCTR, CODREG) REFERENCES REFREG (CODCTR, CODREG)
    ON DELETE SET NULL
    ON UPDATE CASCADE
```

При изменении кода страны и/или кода региона в справочнике стран или в справочнике регионов эти изменения будут автоматически внесены в соответствующие строки таблицы организаций. А вот при удалении страны или региона я решил устанавливать столбцы внешнего ключа в пустое значение — что-то не хочется мне удалять такие организации и множество их дочерних таблиц. При этом необходимо будет в дальнейшем вносить, скорее всего вручную, изменения в адресную часть таких организаций.

У нас осталась одна проблема — как поступать в том случае, если организация находится в районном центре или в районе и происходит изменение кода района в таблице района или район вообще удаляется. Эта проблема просто решается написанием триггеров, реагирующих на эти события. Такие триггеры мы с вами напишем, но несколько позже.

Еще один внешний ключ у нас ссылается на справочник организационно-правовых форм. Интересно, сможете ли вы сейчас вот так просто навскидку, не задумываясь сказать, какое отношение существует между организациями и таблицей организационно-правовых форм? У каждой организации может быть только одна организационно-правовая форма. Почему-то очень многие, именно не задумываясь, отвечают, что тут существует отношение "один-к-одному". Это, конечно же, неверно. Отношение "один-ко-многим". Одна организационно-правовая форма и много организаций, имеющих эту форму.

Рассмотрим еще одну таблицу — список персонала организации (листинг 4.6).

Листинг 4.6. Создание таблицы сотрудников организации

```
    /*** Сотрудники организации ***/
CREATE TABLE STAFF
```

```
( COD       D_INTEGER NOT NULL,  /* Код сотрудника – первичный ключ */
  CODPEOPLE D_INTEGER,           /* Код человека из списка людей */
  CODORG    D_INTEGER,           /* Код организации */
  DUTIES    D_CHAR40,            /* Должность */
  SALARY    D_DECIMAL,           /* Оклад */
  NET_SALARY    COMPUTED BY (SALARY * .87),
  CONSTRAINT PK_STAFF PRIMARY KEY (COD),
  CONSTRAINT FK1_STAFF
    FOREIGN KEY (CODPEOPLE) REFERENCES PEOPLE (COD)
      ON DELETE CASCADE,
  CONSTRAINT FK2_STAFF
    FOREIGN KEY (CODORG) REFERENCES ORGANIZATION (COD)
      ON DELETE CASCADE
);
CREATE GENERATOR GEN_STAFF;
COMMIT;
```

Собственно, ничего нового здесь для нас с вами нет. Эта таблица иллюстрирует реализацию отношения "многие-ко-многим" (в одной организации работает много людей, один человек может в принципе работать в нескольких организациях). Мы добавили эту промежуточную таблицу и задали в ней два внешних ключа, ссылающихся как на таблицу организаций, так и на таблицу людей.

Для задания столбца, описывающего должностной оклад (столбец `SALARY`), использован домен `D_DECIMAL`, который был описан с типом данных `DECIMAL(8, 2)`.

В таблице людей мы использовали вычисляемый столбец (`COMPUTED BY`), который представлял собой конкатенацию столбцов фамилии, имени, отчества и разделяющих их пробелов. Приведем еще один пример вычисляемого столбца, который можно было бы использовать в таблице сотрудников организации. Это будет действительно *вычисляемый* столбец, потому что возвращаемым значением будет число.

В таблице столбец `SALARY` задает должностной оклад сотрудника. Мы можем ввести вычисляемый столбец `NET_SALARY`, который будет содержать сумму оклада, из которой вычтены налоги. Поскольку налог составляет 13%, нам нужно умножить оклад на число 0.87:

```
NET_SALARY    COMPUTED BY (SALARY * .87),
```

Напомню, что по правилам SQL, как и в большинстве языков программирования, в литерале числа можно опускать указание целой части числа, если это незначащий ноль.

Таблица, описывающая виды деятельности организаций, имеет следующую структуру — листинг 4.7.

Листинг 4.7. Таблица видов деятельности организаций

```
    /*** Виды деятельности организации ***/
CREATE TABLE ORGACTIV
( COD         D_INTEGER NOT NULL,
  CODACT      D_CHAR4,          /* Код вида деятельности */
  CODORG      D_INTEGER,        /* Код организации */
  TEXT        D_CHAR110,        /* Описание вида деятельности */
  CONSTRAINT PK_ORGACTIV
    PRIMARY KEY (COD),          /* Первичный ключ */
  CONSTRAINT FK1_ORGACTIV
    FOREIGN KEY (CODACT) REFERENCES REFACTIV (COD)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT FK2_ORGACTIV
    FOREIGN KEY (CODORG) REFERENCES ORGANIZATION (COD)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
CREATE GENERATOR GEN_ORGACTIV;
COMMIT;
```

Таблица также содержит искусственный ключ и два внешних ключа, ссылающихся на справочную таблицу видов деятельности и на таблицу организаций.

Структура таблицы правонарушений людей показана в листинге 4.8.

Листинг 4.8. Создание таблицы правонарушений людей

```
    /*** Правонарушения людей ***/
CREATE TABLE PEOPLEADMIN
( COD       D_INTEGER NOT NULL,  /* Код — первичный ключ */
  CODPEOPLE D_INTEGER,           /* Код человека из списка людей */
  CODADMIN  D_CHAR8,             /* Код правонарушения из справочника */
  DATEADMIN D_DATE,              /* Дата правонарушения */
  CONSTRAINT PK_PEOPLEADMIN PRIMARY KEY (COD),
  CONSTRAINT FK1_PEOPLEADMIN
    FOREIGN KEY (CODPEOPLE) REFERENCES PEOPLE (COD)
     ON DELETE CASCADE,
  CONSTRAINT FK2_PEOPLEADMIN
```

```
    FOREIGN KEY (CODADMIN) REFERENCES REFADMIN (COD)
      ON UPDATE CASCADE
      ON DELETE CASCADE
);
CREATE GENERATOR GEN_PEOPLEADMIN;
```

Эта таблица по своей идее такая же, как и таблица сотрудников.

Весь скрипт завершается двумя операторами:

```
COMMIT;
EXIT;
```

Первый подтверждает транзакцию, в контексте которой выполнялось создание таблиц, второй завершает скрипт. Мне попадались программы работы со скриптами, которые выполнялись неправильно при отсутствии оператора `EXIT`. По этой причине рекомендую использовать его всегда.

***

Тяга к совершенству заставляет меня поместить в нашу базу данных еще несколько таблиц, которые мы в программах использовать не будем, но которые придадут законченность нашей работе по проектированию базы данных.

Посмотрите на рис. 4.1. Там показана иерархическая связь между странами, регионами и районами.

Давайте добавим еще таблицу, описывающую районы города. В нашей стране районы существуют только у региональных (республиканских, областных и краевых) центров. По этой причине такая таблица должна находиться на том же уровне иерархии, что и районы региона. Кроме того, добавим еще таблицу, в которую можно будет помещать список населенных пунктов региона. Естественным образом это будет следующий уровень в нашей иерархии, сразу после районов. Окончательно структура этой части базы данных будет выглядеть, как показано на рис. 4.3.

Фрагмент скрипта создания этих двух дополнительных таблиц представлен в листинге 4.9.

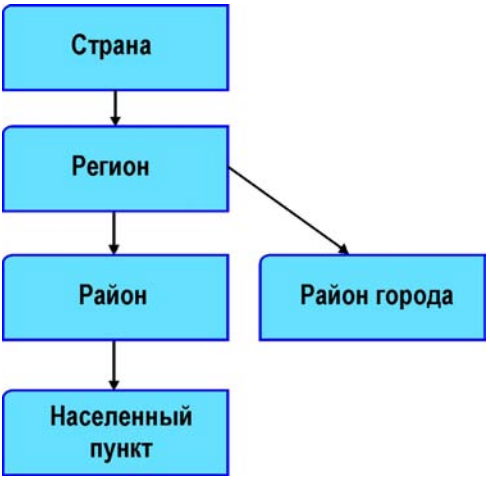


Рис. 4.3. Связь между таблицами административно-территориального деления страны

Листинг 4.9. Создание таблиц районов города и населенных пунктов

```
     /*** Справочник районов города ***/
CREATE TABLE REFDISTRTOWN
( CODCTR   D_CHAR3 NOT NULL,    /* Код страны */
  CODREG   D_CHAR2 NOT NULL,    /* Код региона */
  CODDISTR D_CHAR2 NOT NULL,    /* Код района города */
  AREA     D_CHAR25,            /* Название района города */
  DESCR    D_BLOB,              /* Дополнительное описание */
  CONSTRAINT PK_REFDISTRTOWN
    PRIMARY KEY (CODCTR, CODREG, CODDISTR),
  CONSTRAINT FK_REFDISTRTOWN
    FOREIGN KEY (CODCTR, CODREG) REFERENCES REFREG (CODCTR, CODREG)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
COMMIT;
     /*** Справочник населенных пунктов ***/
CREATE TABLE REFVILLAGE
( CODCTR   D_CHAR3 NOT NULL,    /* Код страны */
  CODREG   D_CHAR2 NOT NULL,    /* Код региона */
  CODAREA  D_CHAR2 NOT NULL,    /* Код района региона */
  CODVILL  D_CHAR3 NOT NULL,    /* Код населенного пункта */
  NAME     D_CHAR25,            /* Название населенного пункта */
  DESCR    D_BLOB,              /* Дополнительное описание */
  CONSTRAINT PK_REFVILLAGE
    PRIMARY KEY (CODCTR, CODREG, CODAREA, CODVILL),
```

```
  CONSTRAINT FK_REFVILLAGE
    FOREIGN KEY (CODCTR, CODREG, CODAREA)
      REFERENCES REFAREA (CODCTR, CODREG, CODAREA)
        ON DELETE CASCADE
        ON UPDATE CASCADE
);
COMMIT;
```

Надо полагать, тут все понятно и объяснения не нужны.

Здесь можно было бы пойти еще дальше и добавить таблицу, описывающую федеральные округа. Отношение между федеральным округом и регионом — "один-ко-многим". К одному федеральному округу относится несколько регионов. Первичным ключом в таблице федеральных округов будет код страны и код федерального округа. В этом случае в таблицу регионов нужно добавить код федерального округа и описать еще один внешний ключ, ссылающийся на первичный ключ соответствующей строки федерального округа — на код страны и код федерального округа в этой стране. Мы этого здесь делать не будем, однако, если вашему заказчику такое понадобится, вы это сделаете легко и непринужденно.

Для большой справочно-поисковой системы, имеющей достаточно богатые возможности по выборке данных, можно добавить в базу данных еще массу справочных таблиц, например, справочник товаров, позволяющий описывать продукцию и услуги, справочник форм собственности, справочник валют и ряд других.

Соответствующий фрагмент скрипта показан в листинге 4.10.

Листинг 4.10. Создание дополнительных справочных таблиц

```
    /*** Справочник товаров ***/
CREATE TABLE REFGOODS
( COD         D_CHAR6 NOT NULL,      /* Код товара */
  NAME        D_CHAR110,             /* Название товара */
  CONSTRAINT PK_REFGOODS
    PRIMARY KEY (COD)                /* Первичный ключ */
);
COMMIT;

    /*** Справочник валют ***/
CREATE TABLE REFCURRENCY
( CODCUR D_CHAR3 NOT NULL,           /* Код валюты */
  Name   D_CHAR35,                   /* Название валюты */
```

```
  CONSTRAINT PK_REFCURRENCY
    PRIMARY KEY (CODCUR)              /* Первичный ключ */
);
COMMIT;

     /*** Справочник форм собственности ***/
CREATE TABLE REFFORMPROP
( COD          D_CHAR2 NOT NULL,      /* Код формы собственности */
  NAME         D_CHAR120,             /* Название формы собственности */
  CONSTRAINT PK_REFFORMPROP
    PRIMARY KEY (COD)                 /* Первичный ключ */
);
COMMIT;

     /*** Справочник специальностей ***/
CREATE TABLE REFSPEC
( COD          D_CHAR6 NOT NULL,      /* Код специальности */
  NAME         D_CHAR120,             /* Название специальности */
  CONSTRAINT PK_REFSPEC
    PRIMARY KEY (COD)                 /* Первичный ключ */
);
COMMIT;

     /*** Справочник наград ***/
CREATE TABLE REFREWARD
( COD          D_CHAR6 NOT NULL,      /* Код награды */
  NAME         D_CHAR120,             /* Название награды */
  CONSTRAINT PK_REFREWARD
  PRIMARY KEY (COD)                   /* Первичный ключ */
);
COMMIT;
```

Эти таблицы я также поместил в базу данных. Чтобы вы не подумали, что у нас в базе данных находятся одни хулиганы, я добавил и справочник наград, которые могут получать достойные люди.

Для использования по назначению справочника форм собственности в структуру таблицы организаций нужно ввести столбец, внешний ключ, ссылающийся на код формы собственности таблицы `REFFORMPROP`.

Наличие справочника товаров позволит нам описывать в организациях покупаемые и продаваемые товары. Для этого необходимо добавить в базу данных еще две таблицы (листинг 4.11).

Листинг 4.11. Таблицы покупаемых и продаваемых товаров организации

```
     /*** Продаваемые товары организации ***/
CREATE TABLE SALEGOODS
( COD          D_INTEGER NOT NULL,
  CODGOODS     D_CHAR6,          /* Код товара */
  CODORG       D_INTEGER,        /* Код организации */
  TEXT         D_CHAR110,        /* Описание товара */
  CONSTRAINT PK_SALEGOODS
    PRIMARY KEY (COD),          /* Первичный ключ */
  CONSTRAINT FK1_SALEGOODS
    FOREIGN KEY (CODGOODS) REFERENCES REFGOODS (COD)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT FK2_SALEGOODS
    FOREIGN KEY (CODORG) REFERENCES ORGANIZATION (COD)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
CREATE GENERATOR GEN_SALEGOODS;
COMMIT;

     /*** Покупаемые товары организации ***/
CREATE TABLE BUYGOODS
( COD          D_INTEGER NOT NULL,
  CODGOODS     D_CHAR6,          /* Код товара */
  CODORG       D_INTEGER,        /* Код организации */
  TEXT         D_CHAR110,        /* Описание товара */
  CONSTRAINT PK_BUYGOODS
    PRIMARY KEY (COD),          /* Первичный ключ */
  CONSTRAINT FK1_BUYGOODS
    FOREIGN KEY (CODGOODS) REFERENCES REFGOODS (COD)
      ON DELETE CASCADE
      ON UPDATE CASCADE,
  CONSTRAINT FK2_BUYGOODS
    FOREIGN KEY (CODORG) REFERENCES ORGANIZATION (COD)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
CREATE GENERATOR GEN_BUYGOODS;
COMMIT;
```

Эти две таблицы имеют совершенно одинаковую структуру. Каждая содержит искусственный первичный ключ (значения которого получаются из соответствующего генератора), внешний ключ, код товара (ссылающийся на таблицу товаров), внешний ключ, код организации (ссылающийся на таблицу

организаций). Кроме того, в таблицы мы поместили текстовое поле TEXT, позволяющее задавать некоторые характеристики товара.

Поскольку у нас существуют справочные таблицы по специальностям и наградам людей, имеет смысл добавить и таблицы приобретенных специальностей и полученных наград (листинг 4.12).

Листинг 4.12. Таблицы специальностей и наград людей

```
     /*** Специальности людей ***/
CREATE TABLE PEOPLESPEC
( COD       D_INTEGER NOT NULL,  /* Код – первичный ключ */
  CODPEOPLE D_INTEGER,           /* Код человека из списка людей */
  CODSPEC   D_CHAR6,             /* Код специальности из справочника */
  DATESPEC  D_DATE,              /* Дата получения */
  CONSTRAINT PK_PEOPLESPEC PRIMARY KEY (COD),
  CONSTRAINT FK1_PEOPLESPEC
    FOREIGN KEY (CODPEOPLE) REFERENCES PEOPLE (COD)
      ON DELETE CASCADE,
  CONSTRAINT FK2_PEOPLESPEC
    FOREIGN KEY (CODSPEC) REFERENCES REFSPEC (COD)
      ON UPDATE CASCADE
      ON DELETE CASCADE
);
CREATE GENERATOR GEN_PEOPLESPEC;
COMMIT;

     /*** Награды людей ***/
CREATE TABLE PEOPLEREW
( COD       D_INTEGER NOT NULL,  /* Код – первичный ключ */
  CODPEOPLE D_INTEGER,           /* Код человека из списка людей */
  CODREW    D_CHAR6,             /* Код награды из справочника */
  DATESPEC  D_DATE,              /* Дата получения */
  CONSTRAINT PK_PEOPLEREW PRIMARY KEY (COD),
  CONSTRAINT FK1_PEOPLEREW
    FOREIGN KEY (CODPEOPLE) REFERENCES PEOPLE (COD)
      ON DELETE CASCADE,
  CONSTRAINT FK2_PEOPLEREW
    FOREIGN KEY (CODREW) REFERENCES REFREWARD (COD)
      ON UPDATE CASCADE
      ON DELETE CASCADE
);
CREATE GENERATOR GEN_PEOPLEREW;
COMMIT;
```

Здесь также отношение между справочниками и людьми "многие-ко-многим".

Добавим еще одну таблицу, которая будет хранить историю изменения окладов сотрудников (листинг 4.13). Эта таблица нам пригодится для иллюстрации действий одного из триггеров (см. *главу 9*).

Листинг 4.13. Таблица истории окладов сотрудников

```
    /*** История окладов сотрудников ***/
CREATE TABLE STAFFHISTORY
( COD          D_INTEGER NOT NULL,  /* Код — первичный ключ */
  CODSTAFF     D_INTEGER,           /* Код сотрудника */
  SALARY       D_DECIMAL,           /* Старое значение оклада */
  DATE SALARY D_DATE,               /* Дата изменения оклада */
  CONSTRAINT PK_STAFFHISTORY PRIMARY KEY (COD),
  CONSTRAINT FK1_STAFFHISTORY
    FOREIGN KEY (CODSTAFF) REFERENCES STAFF (COD)
      ON DELETE CASCADE
);
CREATE GENERATOR GEN_STAFFHISTORY;
COMMIT;
```

Таблица содержит искусственный первичный ключ, значение которого мы будем получать из генератора. Внешний ключ ссылается на родительскую таблицу персонала (`STAFF`). Здесь также присутствует старое значение оклада и дата, до которой сотрудник получал этот оклад.

Имея все эти таблицы в вашей базе данных, вы можете самостоятельно поэкспериментировать с соответствующими данными.

4.6. Удаление и изменение таблиц

Для удаления таблицы нужно соединиться с базой данных и выполнить оператор `DROP TABLE`:

```
DROP TABLE <имя таблицы>;
```

Будет удалена указанная таблица и все существующие в ней данные. Удалять таблицу может владелец базы данных или пользователь с именем `SYSDBA`.

Применяйте этот оператор с осторожностью, чтобы не уничтожить нужные вам данные. Кроме того, вы не сможете удалить из базы данных таблицу, являющуюся родительской для других таблиц, первичные ключи которых ссылаются на данную таблицу. Вначале нужно удалить все дочерние таблицы. Прямо сейчас я попытался из нашей базы данных удалить таблицу стран и,

разумеется, получил сообщение об ошибке: нарушение ссылочной целостности данных.

Для изменения существующей таблицы используется оператор ALTER TABLE:

```
ALTER TABLE <имя таблицы> <операция> [, <операция>] ...;

<операция> ::=
    ADD <определение столбца>
  | ADD <ограничение таблицы>
  | ALTER [COLUMN] <имя столбца> <предложение изменения столбца>
  | DROP <имя столбца>
  | DROP CONSTRAINT <имя ограничения>
```

Предложения ADD <определение столбца> и ADD <ограничение таблицы> добавляют в существующую таблицу новый столбец и новое ограничение соответственно. Их синтаксис совпадает с синтаксисом таких предложений, используемых при создании таблицы.

Как мы уже говорили, существует вариант создания всех таблиц базы данных без задания их ограничений. После этого в таблицы добавляются нужные ограничения с помощью оператора ALTER TABLE.

Предложение DROP <имя столбца> удаляет столбец. Опять же, если вы попытаетесь удалить столбец, который входит в состав первичного, уникального или внешнего ключа, в состав вычисляемого столбца или в состав ограничения таблицы CHECK, то у вас ничего не выйдет. Вначале нужно удалить соответствующее ограничение и/или вычисляемый столбец.

Предложение DROP CONSTRAINT <имя ограничения> удаляет ограничение. Здесь мы и можем получить удовольствие (интеллектуальное) от того, что именовали все наши ограничения. В противном случае система присвоила бы ограничению сложное и непонятное имя, которое следовало бы вначале отыскать в базе данных, а затем уж пытаться удалить.

Предложение ALTER позволяет изменить имя, тип данных и местоположение столбца в списке столбцов таблицы:

```
<предложение изменения столбца> ::=
    TO <новое имя столбца>
  | TYPE <новый тип данных столбца>
  | POSITION <новая позиция столбца>
```

Вы можете изменять таблицы и в том случае, когда эти таблицы заполнены данными. Можно изменять любые характеристики, в том числе и типы данных любых столбцов. Однако занятие это очень нездоровое.

Во-первых, весьма хлопотное, поскольку требует выполнения вручную множества изменений в таблицах базы данных как до изменения описания таблицы, так и после, во-вторых, очень велика вероятность потери данных в табли-

цах в тех столбцах, характеристики которых вы изменяете. Потери возможны даже в том простом случае, если вы изменяете только размер столбца.

Лучший вариант:

4. "Вытащить" все данные изменяемой таблицы базы данных, сохранив их в виде скрипта. В IBExpert для этого есть средство Extract.
5. Внести требуемые изменения в скрипты создания таблицы базы данных.
6. Создать таблицу с новыми характеристиками.
7. Загрузить в базу данных сохраненные данные.

В любом случае порядок действий сильно зависит от конкретных изменений в базе данных, которые вам нужно выполнить.

Если же вам все-таки очень хочется выполнить изменения в существующих таблицах, не копируя базу данных, то посмотрите в документации по InterBase технологию таких действий. По-моему такой документ назывался Getting Started.

Изменять таблицу также может владелец базы данных или пользователь SYSDBA.

4.7. Создание таблиц для учебной базы данных

Если вы много экспериментировали с вашей базой данных, создавая, изменяя и удаляя домены, таблицы, генераторы, то следует пересоздать базу данных.

Для этого последовательно выполните следующие скрипты:

- RecreateDatabase.sql (соединится с существующей базой данных, удалит ее и заново создаст);
- CreateDomains.sql (создаст домены);
- CreateTables.sql (создаст все необходимые таблицы — справочные и оперативные).

4.8. Использование индексов

Индекс — механизм, который используется для ускорения поиска записей, соответствующих некоторому условию поиска, и для поддержания ограничений уникальности столбцов. Поиск данных выполняется быстро, потому что значения в индексе упорядочены, а сам индекс относительно мал.

В отличие от простых, настольных (desktop) СУБД типа dBase и FoxPro, где без индексов просто нельзя было обойтись, индексы в настоящих реляционных базах данных не играют столь большой роли. Однако во многих случаях грамотно созданные индексы могут резко улучшить временные характери-

стики созданной системы — ускорить выборку и упорядочение данных. Иногда бывает наоборот — плохо созданные индексы сильно ухудшают производительность системы.

Напомню, что для таких ограничений таблицы, как первичный ключ, уникальный ключ и внешний ключ, система автоматически создает необходимые индексы.

Оператор `CREATE INDEX` создает индекс для одного или более столбцов таблицы. Его синтаксис:

```
CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]
INDEX <имя индекса> ON <таблица> (<столбец> [, <столбец>] ...);
```

Задание сортировки индекса производится с помощью ключевых слов `ASCENDING` (возрастающий) или `DESCENDING` (убывающий). По умолчанию, если упорядоченность не указывается, принимается возрастающий порядок. Для определения индекса, запрещающего дублирующие записи, надо включить в оператор `CREATE INDEX` ключевое слово `UNIQUE`.

Оператор `ALTER INDEX` деактивирует и активирует индекс. Это бывает полезным при выполнении так называемых "пакетных" обновлений данных, когда в базу данных загружается большое количество строк таблицы. Перед загрузкой данных соответствующий индекс (или индексы) деактивируется, а после завершения загрузки активируется. При этом произойдет пересоздание индекса. Для всей операции это займет меньше времени, а пересозданный индекс будет хорошего качества.

Синтаксис оператора `ALTER INDEX`:

```
ALTER INDEX <имя индекса> {ACTIVE | INACTIVE};
```

`DROP INDEX` удаляет созданный пользователем индекс из базы данных. Системные индексы, которые создаются для столбцов с ограничениями `UNIQUE`, `PRIMARY KEY` и `FOREIGN KEY`, не могут быть удалены. Синтаксис оператора:

```
DROP INDEX <имя индекса>;
```

Создавать, изменять и удалять индексы может владелец базы данных или пользователь `SYSDBA`.

Есть одно **важное правило по использованию индексов** — ни в коем случае нельзя создавать индексы, имеющие ту же структуру (состав столбцов в индексе, а также его упорядоченность), что и автоматически создаваемые системой индексы для первичных, уникальных и внешних ключей. В противном случае можно столкнуться с полным заклиниванием сервера базы данных при выполнении некоторых запросов на выборку данных.

В нашей учебной базе данных мы вообще не будем создавать никаких индексов.

Вопросы оптимизации работы системы при использовании индексов достаточно подробно рассматривает Хелен Борри в своей книге по Firebird.

4.9. Правильные ответы

Здесь мы рассмотрим правильные, с моей точки зрения, решения заданий предложенной вам контрольной работы (см. *разд. 4.3*).

4.9.1. Дисциплины, разделы, темы

При проектировании дисциплин, разделов дисциплины и тем разделов следует создать три соответствующие таблицы. Между ними существует естественная иерархическая связь — дисциплина состоит из нескольких разделов, раздел состоит из нескольких тем. Здесь структура базы данных похожа на рассмотренную нами структуру стран, регионов, районов.

В табл. 4.14—4.16 представлена приблизительная структура таблиц базы данных, где указаны только самые необходимые столбцы.

Таблица 4.14. *Список дисциплин* `SUBJECT`

Имя	Тип данных	Название
`CODSUBJ`	Строка	Код дисциплины. Первичный ключ
`NAME`	Строка	Название дисциплины

Первичным ключом здесь будет код дисциплины.

Таблица 4.15. *Разделы дисциплины* `SECTION`

Имя	Тип данных	Название
`CODSUBJ`	Строка	Код дисциплины. Внешний ключ. Входит в состав первичного ключа
`CODSECT`	Строка	Код раздела. Входит в состав первичного ключа
`NAME`	Строка	Название раздела

Здесь первичный ключ — код дисциплины и код раздела. Внешний ключ, ссылающийся на таблицу дисциплин, код дисциплины.

Таблица 4.16. *Темы разделов* `TOPIC`

Имя	Тип данных	Название
`CODSUBJ`	Строка	Код дисциплины. Входит в состав первичного ключа. Входит в состав внешнего ключа
`CODSECT`	Строка	Код раздела. Входит в состав первичного ключа. Входит в состав внешнего ключа
`CODTOPIC`	Строка	Код темы. Входит в состав первичного ключа
`NAME`	Строка	Название темы

Первичным ключом здесь будет код дисциплины, код раздела и код темы. Внешний ключ, ссылающийся на таблицу разделов, состоит из кода раздела и кода темы.

А теперь напишите операторы SQL для создания этих таблиц с указанием первичных и внешних ключей.

Я тоже сейчас это сделаю. Сравните свой вариант с моим (листинг 4.14).

Листинг 4.14. Создание таблиц для дисциплин, разделов, тем

```
/***  Таблица дисциплин  ***/
CREATE TABLE SUBJECT
( CODSUBJ  CHAR(4) NOT NULL,  /* Код дисциплины */
  NAME     VARCHAR(100),      /* Название дисциплины */
  CONSTRAINT PK_SUBJECT PRIMARY KEY (CODSUBJ)
);

/***  Таблица разделов  ***/
CREATE TABLE SECTION
( CODSUBJ  CHAR(4) NOT NULL,  /* Код дисциплины */
  CODSECT  CHAR(4) NOT NULL,  /* Код раздела */
  NAME     VARCHAR(100),      /* Название раздела */
  CONSTRAINT PK_SECTION PRIMARY KEY (CODSUBJ, CODSECT),
  CONSTRAINT FK_SECTION
    FOREIGN KEY (CODSUBJ) REFERENCES SUBJECT (CODSUBJ)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);

/***  Таблица тем  ***/
CREATE TABLE TOPIC
```

```
( CODSUBJ  CHAR(4) NOT NULL,  /* Код дисциплины */
  CODSECT  CHAR(4) NOT NULL,  /* Код раздела */
  CODTOPIC CHAR(4) NOT NULL,  /* Код темы */
  NAME     VARCHAR(100),      /* Название темы */
  CONSTRAINT PK_TOPIC PRIMARY KEY (CODSUBJ, CODSECT, CODTOPIC),
  CONSTRAINT FK_TOPIC
    FOREIGN KEY (CODSUBJ, CODSECT) REFERENCES SECTION (CODSUBJ, CODSECT)
      ON DELETE CASCADE
      ON UPDATE CASCADE
);
COMMIT;
```

Ну, и у кого лучше? Конкретные количественные характеристики (размеры столбцов) не имеют сейчас никакого значения. Предполагаю, что вы не писали комментарии к столбцам таблиц. Я прав?

Проверьте ваши операторы, соединившись через isql или IBExpert с нашей учебной базой данных и добавив три новые таблицы. Мы их потом, конечно, удалим.

4.9.2. Организации, отделы, сотрудники

В этом задании есть как минимум два нормальных варианта решений. В любом случае нужно создать три таблицы — по организациям, отделам и сотрудникам, между которыми устанавливается иерархическая связь с помощью иерархических ключей (табл. 4.17—4.19).

Разница между двумя вариантами только в том, что в одном из вариантов для таблицы сотрудников первичный ключ будет состоять из кода организации, кода отдела и номера сотрудника в отделе, а в другом варианте первичным ключом будет код организации и табельный номер сотрудника, который является уникальным для организации. Поскольку табельный номер уникален в рамках только *одной* организации, нам необходимо в состав первичного ключа сотрудника добавить и код организации.

Рассмотрим первый вариант.

Список организаций содержит, по меньшей мере, код и название (табл. 4.17).

Таблица 4.17. *Список организаций* `ORGANIZATION`

Имя	Тип данных	Название
`CODORG`	Строка	Код организации. Первичный ключ
`NAME`	Строка	Наименование организации

В состав первичного ключа таблицы отделов входит код организации и код (или номер) отдела. Внешний ключ — код организации, ссылается на таблицу организаций.

Таблица 4.18. *Список отделов организации* `DEPARTMENT`

Имя	Тип данных	Название
`CODORG`	Строка	Код организации. Внешний ключ. Входит в состав первичного ключа
`CODDEPART`	Строка	Код отдела. Входит в состав первичного ключа
`NAME`	Строка	Наименование отдела

Таблица 4.19. *Список сотрудников отделов организации* `STAFF`

Имя	Тип данных	Название
`CODORG`	Строка	Код организации. Внешний ключ. Входит в состав первичного ключа
`CODDEPART`	Строка	Код отдела. Входит в состав первичного ключа.
`CODSTAFF`	Строка	Код (табельный номер) сотрудника. Входит в состав первичного ключа
`NAME`	Строка	Фамилия сотрудника, может быть имя, отчество и другие реквизиты

Первичный ключ — код организации, код отдела, код сотрудника. Внешний ключ (код организации, код отдела) ссылается на таблицу отделов организации.

Напишите скрипт для создания этих таблиц. Проверьте правильность скрипта на нашей учебной базе данных. А я, по правде сказать, устал и не буду сегодня этого делать. Ваша очередь активно трудиться.

4.9.3. Движение материалов на складах

Здесь четыре таблицы. Связь между ними представлена на рис. 4.4.

Справочник материалов совсем прост (табл. 4.20).

Таблица 4.20. *Справочник материалов* `MATIRIAL`

Имя	Тип данных	Название
`CODMATERIAL`	Строка	Код материала. Первичный ключ
`NAME`	Строка	Наименование материала

Разумеется, на конкретном предприятии в структуру записи этого справочника могут входить и различные дополнительные характеристики.

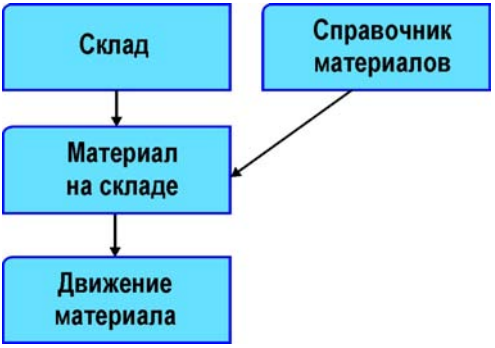


Рис. 4.4. Связь между таблицами для решения задачи движения материалов

Список складов также не отличается большой сложностью (табл. 4.21).

Таблица 4.21. *Список складов* `WAREHOUSE`

Имя	Тип данных	Название
`CODWAREHOUSE`	Строка	Код склада. Первичный ключ
`NAME`	Строка	Наименование склада

Для таблицы материалов на складе следует ввести искусственный первичный ключ, поскольку лично я не вижу других кандидатов в первичные ключи (табл. 4.22). Нужны два внешних ключа, ссылающихся на таблицу складов и на справочник материалов. Еще я для примера добавил величину остатка материалов на складе.

Таблица 4.22. *Материалы на складе* `MATERIAL_ON_WAREHOUSE`

Имя	Тип данных	Название
`COD`	Целое	Искусственный первичный ключ
`CODWAREHOUSE`	Строка	Код склада. Внешний ключ. Ссылается на таблицу складов
`CODMATERIAL`	Строка	Код материала. Внешний ключ. Ссылается на справочник материалов
`REST`	Число	Остатки материала на складе

Для таблицы движения материалов опять же нужен искусственный первичный ключ, внешний ключ, связывающий движение с конкретным материалом

на конкретном складе, количество материала, пришедшего на склад или выданного со склада, возможно, дата операции (табл. 4.23).

По поводу указания прихода или расхода материала есть два варианта. В первом варианте нужно ввести признак, который определит, является ли операция приходом или расходом. Во втором вид операции можно определить по знаку количества — если число положительное, то это операция прихода, если отрицательное, то операция расхода. Мне больше нравится первый, поскольку в нем явно называются вещи своими именами: приход так приход, расход так расход. В этом случае следовало бы для столбца количества прихода/расхода задать ограничение, что количество должно быть больше нуля. Правда, в бухгалтерском учете, насколько я помню, существует понятие сторнирующей операции, которая позволяет внести исправления в неверно введенные ранее данные. В этом случае следует для прихода/расхода допустить и отрицательные значения.

Существует еще и третий вариант. Приход и расход представлять в отдельных таблицах. Лично мне эта идея не очень нравится. Думается, что выбор между этими вариантами — дело вкуса.

Таблица 4.23. *Движение материала на складе* `MATERIALMOVE`

Имя	Тип данных	Название
`COD`	Целое	Искусственный первичный ключ
`CODWAREHOUSE`	Строка	Код склада. Входит в состав внешнего ключа
`CODMATERIAL`	Строка	Код материала. Входит в состав внешнего ключа
`SIGN`	Число	Признак операции — приход или расход
`NUMBER`	Число	Количество материала в операции

Внешний ключ состоит из двух столбцов — кода склада и кода материала. Он ссылается на таблицу материалов на складе.

Сейчас опять ваша очередь писать и проверять на учебной базе данных скрипт для создания этих таблиц. Я же приступаю к добавлению и изменению данных в таблицах нашей учебной базы данных. Когда закончите писать скрипты, возвращайтесь сюда, будем опять работать вместе.

***

Как мы с вами говорили, все метаданные, в том числе и описания таблиц, хранятся в системных таблицах. В *приложении 4* описаны некоторые системные таблицы и программы, позволяющие просматривать взаимосвязанные таблицы. Там вы можете посмотреть, как хранятся описания таблиц, с помо-

щью написанных программ можно увидеть, в каком виде представлены таблицы нашей учебной базы данных.

Что там за перевалом?

Мы с вами научились проектировать и создавать с помощью операторов языка SQL таблицы, использовать для них первичные ключи. Знаем, как использовать генераторы для получения уникальных значений искусственных первичных ключей. Можем устанавливать необходимые связи между таблицами, используя внешние и первичные (уникальные) ключи. Легко используем вычисляемые столбцы и столбцы, основанные на доменах.

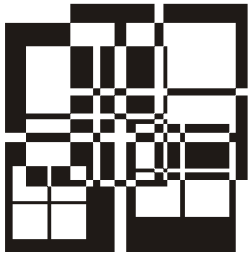
Успели также сказать несколько слов об индексах.

Если вы внимательно читали эту главу и честно пытались решить предложенные задачи, то, значит, вы уже стали классным специалистом в области проектирования и создания реляционных баз данных. Примите мои поздравления.

Пора теперь заполнять нашу базу данных справочными и оперативными данными, изменять отдельные строки и при необходимости удалять неверные или просто устаревшие данные.

Этим мы займемся в следующей главе.

ГЛАВА 5



Добавление, изменение, удаление и выборка данных

До этого момента мы с вами использовали в основном операторы DDL — языка определения данных. Сейчас мы приступаем к рассмотрению операторов DML, языка манипулирования данными (надеюсь, вы помните, что это на самом деле не самостоятельные языки, а подмножества SQL).

Мы рассмотрим средства добавления (оператор `INSERT`), удаления (`DELETE`) и изменения (`UPDATE`) данных в базе данных. Не сможем мы обойтись и без оператора выборки данных — `SELECT`. В этой главе мы пока опишем лишь наиболее простые варианты использования этого оператора. Детальному исследованию возможностей `SELECT` мы посвятим всю *главу 7*.

Здесь же, забегая немного вперед, мы рассмотрим несколько необходимых нам для работы триггеров.

В следующей главе мы напишем небольшое количество достаточно простых программ для работы с некоторыми таблицами нашей учебной базы данных. Написание этих учебных программ проиллюстрирует основные методы создания реальных программ для большого количества вариантов их применения, существующих в реальной программистской жизни. Программы будут создаваться как с применением компонентов FIBPlus, так и IBX.

Напомню, в *главе 1* мы говорили о четырех уровнях взаимодействия с базой данных. Мы будем использовать как средства языка SQL для обращения к данным (и метаданным) базы данных, так и подходящие компоненты, позволяющие из создаваемой программы работать с базой данных. Иными словами, будем активно трудиться для освоения способов работы на втором и третьем уровнях.

5.1. Добавление данных в базу данных

Оператор INSERT — это средство, с помощью которого запоминается одна или более строк данных в существующей таблице базы данных. Существует два варианта оператора. Один позволяет вводить значения в одну строку. Другой дает возможность помещать в таблицу сразу группу строк с помощью оператора SELECT, который выбирает множество строк из другой или той же самой таблицы.

5.1.1. Добавление в таблицу одной строки

Синтаксис оператора INSERT, добавляющего в указанную таблицу одну строку:

```
INSERT INTO <имя таблицы> [(<столбцы>)] VALUES (<значения>);
```

Здесь <столбцы> — список имен столбцов, разделенных запятыми, <значения> — список значений, присваиваемых столбцам таблицы. Также разделяются запятыми. Значением может быть литерал (чаще всего) или выражение, сколь угодно сложное, в том числе это может быть оператор SELECT, возвращающий ровно одно значение — так называемый единичный (singleton) SELECT. Этот оператор должен заключаться в скобки. Здесь также можно применять некоторые встроенные и внешние (UDF) функции, существующие в языках SQL InterBase и Firebird. Чуть позже мы рассмотрим примеры, где используются функции GEN_ID и UPPER.

Если не указываются имена столбцов, то значения присваиваются столбцам в том порядке, в котором они были определены при создании таблицы, или в том порядке, в котором располагаются столбцы после изменения их позиций при выполнении оператора ALTER TABLE. В списке должно быть такое же количество значений, что и столбцов в таблице. Если были заданы имена столбцов, то значения задаются в том же порядке, что и имена столбцов. Столбцы, которые не указаны, получают значения по умолчанию или NULL в зависимости от определения этих столбцов.

Предыдущий абзац оказался довольно тяжеловесным и нудным. Однако не поленитесь вчитаться в него. Тут все сказано правильно относительно задания порядка значений в операторе добавления. И все же лучшим вариантом является явное перечисление имен столбцов в этом операторе, что я и делаю всегда (или почти всегда).

Рекомендую выполнить все следующие примеры. Для этого нужно запустить на выполнение IBExpert и вызвать инструмент **Script Executive**. В этом окне вводим группу операторов и выполняем их, щелкнув по кнопке **Run** или нажав клавишу <F9>. Для каждой группы операторов добавления не забывайте предварительно записать операторы установки диалекта, набора символов по умолчанию и соединения с базой данных (листинг 5.1).

Листинг 5.1. Соединение с учебной базой данных

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';
```

Каждую группу операторов, кроме соединения с базой данных, завершайте оператором подтверждения транзакции `COMMIT`.

Пример добавления строк в справочник стран — листинг 5.2.

Листинг 5.2. Добавление строк в справочник стран

```
INSERT INTO REFCTR (CODCTR, NAME, FULLNAME, CAPITAL)
 VALUES ('558', 'РОССИЯ', 'Российская Федерация', 'Москва');
INSERT INTO REFCTR (NAME, FULLNAME, CODCTR, CAPITAL)
 VALUES ('USA', 'United States of America', 'USA', 'Washington');
INSERT INTO REFCTR (NAME, FULLNAME, CODCTR, CAPITAL)
 VALUES ('ENGLAND', 'United Kingdom', 'ENG', 'London');
```

Обратите внимание, что порядок столбцов и, соответственно, их значений в первом операторе отличается от порядка в других операторах.

После добавления соответствующей страны мы можем добавлять ее регионы, а после добавления региона можно добавлять его районы — прежде чем добавлять данные в дочернюю таблицу, мы должны поместить в базу данных соответствующую строку родительской таблицы, на первичный ключ которой будет ссылаться внешний ключ дочерней таблицы. Иначе мы получим исключение, и новая строка не будет записана.

В некоторых случаях добавления новых строк мы можем в дочерней таблице задать значение `NULL` для внешнего ключа. Тогда такая строка будет записана, даже если в базе данных не существует ни одной записи родительской таблицы. Но нам потом придется вносить в нее изменения, устанавливая нужное значение внешнего ключа. Такой трюк мы с вами сегодня проделаем для таблицы людей. Для регионов же этот номер не пройдет, поскольку здесь у нас чистая иерархия — внешний ключ в дочерней таблице (таблице регионов `REFREG`) входит в состав первичного ключа этой таблицы и по этой причине не может принимать пустого значения.

Аналогично, для того, чтобы поместить в базу данных любую строку района (таблица `REFAREA`), необходимо предварительно туда добавить соответствующую запись страны и региона.

Возможности оператора INSERT достаточно большие. Мы можем, например, для получения значения любого столбца использовать оператор SELECT, который возвращает только одно значение. Такой оператор может обращаться к любым данным, хранящимся в нашей базе.

```
INSERT INTO REFREG (CODCTR, CODREG, CENTER, NAMEREG)
 VALUES ((SELECT CODCTR FROM REFCTR WHERE NAME = 'РОССИЯ'),
 '50', 'МОСКВА', 'Московская область');
```

Здесь мы добавляем новую строку в таблицу регионов. Для получения значения кода страны мы использовали оператор SELECT, который выбирает код страны для России. Еще раз обратите внимание — оператор SELECT заключен в скобки.

Чтобы получить значение первичного ключа из генератора, мы используем функцию GEN_ID. Следующий оператор добавляет запись в таблицу людей:

```
INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, BIRTHDAY, CODMOTHER)
 VALUES (GEN_ID(GEN_PEOPLE, 1), 'АЛЕКСАНДР', 'АЛЕКСАНДРОВИЧ', 'ИВАНОВ',
 CURRENT_DATE – 30, NULL);
```

Для получения кода (это искусственный первичный ключ — столбец COD) мы применяем функцию GEN_ID, используя ранее созданный генератор GEN_PEOPLE.

Обратите внимание на то, как здесь задается дата. Используется *выражение* — из текущей даты (контекстная переменная CURRENT_DATE типа DATE) вычитается 30 дней[9]. В этом случае можно было бы использовать и предварительно определенный литерал 'NOW', однако его форма записи несколько двусмысленная — уж очень этот литерал смахивает на строку символов. Поэтому при использовании его в выражении нужно выполнить явное приведение типа:

```
CAST('NOW' AS DATE)
```

В этой таблице есть три внешних ключа, ссылающихся на ту же самую таблицу — CODMOTHER, CODFATHER и CODOTHERHALF. В нашем операторе добавления мы указали в списке столбцов CODMOTHER и в списке значений задали значение NULL. Столбцы же CODFATHER и CODOTHERHALF нигде не указаны. Им также будет присвоено значение NULL, как значение по умолчанию. Точнее здесь следовало бы сказать — т. к. столбцы не указаны в операторе добавления новой строки и для них не задано значений по умолчанию, то им будет присвоено значение NULL.

---

[9] Ясно, что в реальной жизни подобного типа выражение применять не следует — не нужно, чтобы дата рождения человека зависела от текущей даты.

Еще один момент. Имя, отчество и фамилия введены прописными буквами. Это хорошая практика для полей, по которым может выполняться поиск и упорядочение выбранных строк. Чтобы принудительно переводить буквы в прописные при добавлении и изменении строк таблицы, можно написать простенький триггер, что мы и сделаем в соответствующее время. Сейчас же можно в операторе `INSERT` использовать функцию `UPPER`, которая переводит все буквы строки в прописные. Выполните следующий оператор:

```
INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, SEX)
 VALUES (GEN_ID(GEN_PEOPLE, 1), UPPER('виктория'), UPPER('ивановна'),
 UPPER('шалтенбергер'), '1');
```

В результате в базу данных будет помещена новая запись, содержащая в полях "Имя", "Фамилия" и "Отчество" только прописные буквы. Проверьте это, выполнив оператор `SELECT` для таблицы `PEOPLE`:

```
SELECT * FROM PEOPLE;
```

5.1.2. Добавление в таблицу нескольких строк

Второй вариант оператора `INSERT` позволяет добавить в таблицу сразу несколько записей из другой или той же самой таблицы. Для этого вместо списка значений используется оператор `SELECT`, возвращающий несколько (а может быть и ноль) строк. Синтаксис:

```
INSERT INTO <имя таблицы> [(<столбцы>)] <выражение SELECT>;
```

В таблицу будет помещено столько строк, сколько вернет оператор `SELECT`, а количество записей, как вы понимаете, зависит от предложения `WHERE` в этом операторе.

Следует быть особенно осторожными при добавлении в таблицу строк, получаемых из той же самой таблицы, т. к. в подобных случаях легко получить бесконечно выполняемый оператор — это когда вновь добавленные строки возвращаются в оператор добавления в результате выполнения оператора `SELECT`.

Выполните следующие действия. Создайте новую страну с кодом XXX. Добавьте в таблицу регионов для этой страны все регионы России (листинг 5.3).

Листинг 5.3. Пример использования оператора `SELECT` в операторе добавления

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';
```

```
INSERT INTO REFCTR (CODCTR, NAME) VALUES ('XXX', 'TEST INSERT SELECT');

INSERT INTO REFREG (CODCTR, CODREG, NAMEREG, CENTER)
  SELECT 'XXX', CODREG, NAMEREG, CENTER FROM REFREG
    WHERE CODCTR = '558';

COMMIT;
```

Посмотрите внимательно на оператор добавления списка регионов. В списке столбцов указаны четыре столбца (`CODCTR`, `CODREG`, `NAMEREG`, `CENTER`). В операторе `SELECT` также указано четыре возвращаемых в одной строке значения, однако первым значением указан не столбец, а литерал `'XXX'`. Это позволяет нам добавлять регионы именно к вновь созданной стране.

Проверьте результаты добавления, вызвав в IBExpert инструмент SQL Editor и выполнив оператор:

```
SELECT * FROM REFREG WHERE CODCTR = 'XXX';
```

Удалите теперь эту пробную страну, выполнив оператор:

```
DELETE FROM REFCTR WHERE CODCTR = 'XXX';
```

При этом, разумеется, будут удалены и все соответствующие строки подчиненной таблицы `REFREG`, поскольку в описании внешнего ключа этой таблицы мы задали каскадные изменения как при изменении, так и при удалении строк родительской таблицы: `ON UPDATE CASCADE` и `ON DELETE CASCADE`.

Теперь давайте создадим бесконечный цикл помещения строк в таблицу. Посмотрите на следующие операторы, только посмотрите, но не выполняйте — листинг 5.4.

Листинг 5.4. Пример бесконечного цикла добавления. Не делайте этого!

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';


INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, SEX)
 SELECT (GEN_ID(GEN_PEOPLE, 1), NAME1, NAME2, NAME3, SEX FROM PEOPLE);

COMMIT;
```

Чисто по-дружески не советую вам выполнять эти операторы. Здесь бесконечный цикл. Снять такую задачу довольно сложно — нужно не только за-

вершить работу программы IBExpert или утилиты isql, но и в Диспетчере задач завершить процесс InterBase или Firebird (сначала программы guardian, затем самого сервера базы данных). Сама задача может аварийно завершиться только по исчерпании дискового пространства. Вспомнив, какие объемы внешней памяти сейчас находятся на наших компьютерах, понимаешь, что для аварийного завершения потребуется слишком много времени.

Здесь как раз и происходит тот самый случай, когда вновь добавленные строки опять включаются в процесс добавления с помощью оператора `SELECT`.

На этом примере можно увидеть, каков порядок выполнения действий с базой данных в используемом варианте оператора `INSERT`. Видно, что все выполняется последовательно. Вначале из базы данных выбирается одна строка, соответствующая условиям поиска. Потом она добавляется в базу данных. Далее опять выполняется оператор `SELECT`, который может подхватить ту же самую строку. То есть не происходит первоначального полного выбора всех строк, соответствующих условиям.

Еще раз обратите внимание на то, что в списке в операторе `SELECT` мы можем указывать не только имена столбцов, не только литералы, но и выражения. В данном случае мы опять обращаемся к функции `GEN_ID` для получения уникального значения первичного ключа, и это значение будет частью набора значений, возвращаемых оператором `SELECT`.

5.2. Удаление данных

Ежели мы добавили что-то неправильное, мы можем смело удалить это, используя оператор `DELETE`. Слегка упрощенный синтаксис:

```
DELETE FROM <имя таблицы> [WHERE <условие выбора>];
```

Здесь самая сложная часть — предложение `WHERE`. Мы его будем рассматривать в десяти тысячах вариантов при изучении оператора `SELECT`. Сейчас же отметим следующие моменты оператора `DELETE`.

Первое, и самое важное, используйте оператор с величайшей осторожностью. Если вы не зададите предложение `WHERE`, то будут удалены *все* записи указанной таблицы! Впрочем, пока вы не подтвердите транзакцию (оператор `COMMIT`), ошибочно удаленные данные можно восстановить, выдав оператор `ROLLBACK`.

Предложение `WHERE` позволяет определить круг записей, которые должны быть действительно удалены. Это может быть одна запись (чаще всего так и бывает) или группа записей, имеющих определенный смысл. Рассмотрим пример удаления из таблицы `PEOPLE` всех детей человека, у которого код

(столбец COD) равен единице. В IBExpert или в isql введите и выполните следующий оператор:

```
DELETE FROM PEOPLE WHERE COD IN
  (SELECT COD FROM PEOPLE WHERE ((CODMOTHER = 1) OR (CODFATHER = 1)));
```

В предложении WHERE мы используем вариант IN. То есть должны удаляться те записи, коды которых равняются какому-либо значению в списке. Список же формируется оператором SELECT, который возвращает произвольное количество значений (может быть и ни одного). В этот список попадут только те коды людей, у которых код матери или код отца равен нашему заданному человеку (мы не знаем пол этого человека — или делаем вид, что не знаем, — поэтому проверяем оба внешних ключа).

Поскольку никаких строк в этой таблице пока не существует, не будет никакого удаления. Если же вы добавили перед этим в базу данных гражданку ШАЛТЕНБЕРГЕР, то она все равно не будет удалена, потому что не удовлетворяет условию для удаляемых записей.

5.3. Изменение данных

Изменение данных осуществляется оператором UPDATE. Несколько упрощенный синтаксис:

```
UPDATE <имя таблицы> SET <имя столбца> = <значение>
  [, <имя столбца> = <значение>] ...
  [WHERE <условие выбора>];
```

Особых сложностей при использовании этого оператора нет, если не считать сложностей предложения WHERE.

Рассмотрим маленький пример. Будучи по природе своей людьми добрыми и отзывчивыми, мы хотим увеличить на 12% оклады всем сотрудникам всех организаций (таблица STAFF). Для этого нужно выполнить следующий оператор:

```
UPDATE STAFF SET SALARY = SALARY * 1.12;
```

Другие люди (не столь добрые и не слишком отзывчивые) хотят увеличить всем мужчинам оклады на 15%, а женщинам только на 9%. Им придется использовать два оператора и достаточно сложное предложение WHERE:

```
UPDATE STAFF SET SALARY = SALARY * 1.15
  WHERE (SELECT SEX FROM PEOPLE
    WHERE PEOPLE.COD = STAFF.CODPEOPLE) = '0';
UPDATE STAFF SET SALARY = SALARY * 1.09
  WHERE (SELECT SEX FROM PEOPLE
    WHERE PEOPLE.COD = STAFF.CODPEOPLE) = '1';
```

Первый оператор увеличивает оклад только мужчинам, второй — женщинам. Для выполнения таких несправедливых дискриминационных действий в предложении WHERE используется оператор SELECT, который для каждого сотрудника (таблица STAFF) выбирает его пол из таблицы людей (PEOPLE) и сравнивает с соответствующим литералом.

> ***Замечание***
>
> Я не перестаю удивляться необыкновенным возможностям языка SQL в реляционных базах данных, которые позволяют выполнить практически любые необходимые нам действия над данными, хранящимися в базе данных — будь то добавление, изменение, удаление или выборка данных. Если же декларативными средствами нам не удается сделать что-то нужное, на помощь приходит вся императивная мощь хранимых процедур и триггеров. При этом использование собственных программ работы с базами данных позволит не только создать комфортный для пользователя интерфейс, но и обеспечит дополнительную функциональность, возможность заблаговременно обнаружить ошибки пользователя, просто не позволить ему допустить ошибки.

Обратите внимание на использование в операторах, включающих несколько таблиц, уточняющих имен: PEOPLE.COD и STAFF.CODPEOPLE. Поскольку имена столбцов в разных таблицах могут совпадать, то, чтобы избежать двусмысленности, перед именем столбца помещается имя таблицы и точка.

Позже мы рассмотрим вариант присваивания именам таблиц псевдонимов, позволяющих несколько сократить объем ручного ввода. Псевдонимы иногда также называют не очень привычным для русскоязычного программиста словом алиас (alias) — даже Word, в котором я набираю сейчас этот текст, подчеркивает это слово красной волнистой линией. Впрочем, он иногда подчеркивает и вполне приличные слова.

5.4. Выборка данных

Практически в любом операторе SQL для получения одного значения или набора значений из базы данных можно использовать средства выборки данных.

Для выборки данных применяется оператор SELECT. Вот предварительный упрощенный синтаксис этого оператора:

```
SELECT <список>
   FROM <имя таблицы>
   [WHERE <условия поиска>]
   [ORDER BY <список упорядочения>];
```

Здесь <список> — список выбираемых столбцов. Может содержать имена столбцов, разделенные запятыми, или символ *, который означает, что выбираются все столбцы таблицы, в том числе и вычисляемые. Кроме того, как мы

уже видели, в списке могут присутствовать литералы и выражения, в которых можно использовать функции и всевозможные допустимые в SQL операции. Такую замечательную особенность мы с вами регулярно будем использовать в дальнейшем.

Предложение FROM задает имя таблицы, из которой производится выборка данных. На самом деле выборка может производиться из нескольких таблиц. Это мы рассмотрим позже.

Предложение WHERE определяет, какие именно строки из таблицы (таблиц) попадут в данную выборку. Если предложение отсутствует, то в выборку попадают все строки таблицы.

В некоторых синтаксических конструкциях требуется так называемый единичный (singleton) оператор SELECT — оператор, возвращающий ровно одно значение.

Предложение ORDER BY задает упорядоченность полученного списка строк путем задания списка имен или порядковых номеров столбцов, по значениям которых сортируется список, а также необязательного направления упорядоченности — по возрастанию или по убыванию значений. Чуть позже при написании программ мы будем упорядочивать список стран по краткому названию страны и при желании изменять упорядоченность, благо такую возможность нам предоставляют используемые нами компоненты работы с базой данных. Синтаксис такого списка можно описать следующим образом:

```
<список упорядочения> ::= {<имя столбца> | <номер столбца>}
  [ASC[ENDING] | DESC[ENDING]] [, <список упорядочения>] ...
```

Здесь ASCENDING означает упорядочение по возрастанию значений. Принимается по умолчанию. DESCENDING — по убыванию.

Есть еще одна особенность у оператора SELECT. Существует возможность соединения нескольких таблиц на основании некоторых условий соединения. Это операция соединения (JOIN), которую мы подробно со временем рассмотрим.

Пока нам достаточно описанных сведений об этом операторе. Некоторые уточнения мы сделаем в следующей главе, когда начнем писать программы. Оператору SELECT мы посвятим всю *главу 7*.

5.5. Использование выражений и функций в операторах

Во многих предложениях операторов INSERT, DELETE, UPDATE и SELECT можно использовать выражения и функции. Скажем два слова о каждой из функций.

В дальнейшем мы рассмотрим их более подробно в конкретных операторах. Помимо рассмотренных функций `GEN_ID` и `UPPER` существует еще несколько полезных функций.

`COUNT`. Является так называемой агрегатной функцией, которая работает с некоторым количеством строк. Обычно используется для подсчета количества каких-то объектов. Синтаксис:

```
COUNT ({* | <значение> | DISTINCT <значение> });
```

Здесь `<значение>` — имя столбца или выражение. Ключевое слово `DISTINCT` указывает, что в операции подсчета не учитываются повторяющиеся значения.

Для выполнения операторов `SELECT` нужно в IBExpert соединиться с нашей базой данных и вызвать инструмент SQL Editor. В этом окне можно ввести и выполнить только *один* оператор.

Пример. Подсчитаем количество регионов (штатов) в стране США:

```
SELECT COUNT(*) FROM REFREG WHERE CODCTR = 'USA';
```

Результатом будет, разумеется, 50.

Для подсчета количества людей в таблице `PEOPLE` выполните оператор:

```
SELECT COUNT(NAME3) FROM PEOPLE;
```

Вы получите какое-то число в зависимости от того, какие строки к настоящему моменту вы добавляли в эту таблицу. Чтобы узнать количество различных фамилий, нужно выполнить оператор:

```
SELECT COUNT(DISTINCT NAME3) FROM PEOPLE;
```

Результатом будет число, меньшее, чем в предыдущем случае, поскольку в этой таблице, скорее всего, присутствуют однофамильцы.

`AVG`. Рассчитывает среднее значение числовых столбцов. Синтаксис:

```
AVG ({<значение> | DISTINCT <значение> });
```

`MAX`, `MIN`. Находят, соответственно, максимальное и минимальное значение числовых столбцов. Синтаксис аналогичный.

```
MAX ({<значение> | DISTINCT <значение> });
```

`SUM`. Возвращает сумму числовых столбцов:

```
SUM ({<значение> | DISTINCT <значение> });
```

`SUBSTRING`. Возвращает подстроку. Как функция поддерживается в Firebird, начиная с версии 1.5. В других серверах баз данных является внешней функцией (UDF).

```
SUBSTRING (<значение> FROM <начальная позиция> [FOR <длина>]);
```

Позиции при обращении к этой функции нумеруются, начиная с единицы.

Кроме "родных" функций, т. е. внутренних функций, включенных в язык SQL, существует еще множество определенных пользователем функций — User Defined Function, UDF. Иначе их еще называют внешними функциями. Вы можете написать свои собственные функции и обращаться к ним из своих операторов. Ну, вообще-то "можете" — это сильно сказано. Написание таких функций — дело довольно сложное и хлопотное. Существует большое количество функций, созданных для нас с вами, их мы можем свободно применять в своих разработках.

Более подробно использование этих функций в примерах мы рассмотрим в главе, посвященной оператору `SELECT`.

5.6. Выполнение скриптов добавления и изменения данных

Сейчас самое время заполнить вашу базу данных справочными и оперативными данными. Вам нужно выполнить, как минимум, четыре скрипта. Если же вы вносили какие-то изменения в базу данных, добавляли или удаляли данные, то лучше всего пересоздать базу данных и выполнить необходимые действия по созданию доменов и таблиц, т. е. последовательно выполнить скрипты RecreateDatabase.sql, CreateDomains.sql и скрипт CreateTables.sql.

InsertReference.sql заполняет справочные таблицы. Справочник стран содержит всего лишь несколько стран. Для России помещаются данные о некоторых республиках, краях и областях. Помимо регионов России в базу заносятся и данные по их районам. Там же создаются небольшие фрагменты справочника видов деятельности, справочника организационно-правовых форм и справочника административных и уголовных правонарушений.

Прежде чем заполнять данными оперативные таблицы, необходимо выполнить скрипт создания триггеров — CreateTriggers.sql. Там, в том числе, находятся триггеры, которые вызываются при добавлении и изменении оперативных данных.

InsertOperative.sql заполняет оперативные таблицы — список организаций, людей, сотрудников организаций и правонарушений людей. Если вы вдруг увидите в помещенных в базу данных что-то знакомое или кого-то знакомого, знайте, что эти совпадения совершенно случайны.

Помещение в базу данных организаций выполняется довольно просто (листинг 5.5).

Листинг 5.5. Фрагмент скрипта, выполняющий добавление в базу данных организаций

```
INSERT INTO ORGANIZATION (COD, CODCTR, CODREG, CODAREA, LOCATION, NAME,
CODFORMORG)
  VALUES (GEN_ID(GEN_ORGANIZATION, 1),  '558', '50', NULL,
  '0', 'БИН', '67');
INSERT INTO ORGANIZATION (COD, CODCTR, CODREG, CODAREA, LOCATION, NAME,
CODFORMORG)
  VALUES (GEN_ID(GEN_ORGANIZATION, 1),  '558', '50', NULL,
  '0', 'БРИКО', '67');
INSERT INTO ORGANIZATION (COD, CODCTR, CODREG, CODAREA, LOCATION, NAME,
CODFORMORG)
  VALUES (GEN_ID(GEN_ORGANIZATION, 1),  '558', '64', '38',
  '1', 'ЭНГЕЛЬС-БАНК', '67');
INSERT INTO ORGANIZATION (COD, CODCTR, CODREG, CODAREA, LOCATION, NAME,
CODFORMORG)
  VALUES (GEN_ID(GEN_ORGANIZATION, 1),  '558', '64', '38',
  '1', 'НАРАТБАНК ЭНГЕЛЬСКИЙ ФИЛИАЛ', '67');
INSERT INTO ORGANIZATION (COD, CODCTR, CODREG, CODAREA, LOCATION, NAME,
CODFORMORG)
  VALUES (GEN_ID(GEN_ORGANIZATION, 1),  '558', '64', '38',
  '1', 'ЭКОНОМБАНК ЭНГЕЛЬС', '67');
INSERT INTO ORGANIZATION (COD, CODCTR, CODREG, CODAREA, LOCATION, NAME,
CODFORMORG)
  VALUES (GEN_ID(GEN_ORGANIZATION, 1),  '558', '64', NULL,
  '0', 'ВОЛГОИНВЕСТБАНК', '67');
```

Все добавляемые в базу данных организации по странному стечению обстоятельств оказались расположенными в России. Для каждой организации нам нужно указать код страны, для России это 558, код региона, а если организация расположена в районном центре или в глухой деревушке, то и код района.

Первые два оператора добавляют организации, расположенные в Москве (код региона 50), именно в городе, а не в области. По этой причине мы указываем признак размещения 0, а коду района присваиваем пустое значение.

Следующие три организации расположены в Саратовской области (код региона 64), в районном центре (признак размещения 1), а точнее — в городе Энгельсе (код района 38) Саратовской области.

Последняя организация в этом фрагменте скрипта находится в самой "столице Поволжья", в городе Саратове.

Кроме того, для каждой организации в операторе добавления указывается код организационно-правовой формы, который является внешним ключом, ссылающимся на соответствующий справочник.

С данными по людям дело обстоит сложнее. Запись содержит три внешних ключа, ссылающихся на записи той же самой таблицы. Два внешних ключа ссылаются на записи родителей, если такие сведения нам известны, один внешний ключ ссылается на запись супруга, если таковой (таковая) имеется. Похоже, у нас будет немало сложностей с помещением в базу данных сведений о людях.

Для того чтобы в некоторых записях людей — это несколько первых записей в таблице — задать значения внешних ключей, ссылающихся на родителей и на супругов, после выполнения добавления мне пришлось выполнить выборку (SELECT) всех строк и просмотреть значения их первичных ключей (листинг 5.6). Они ведь формировались на основании генератора, и в процессе помещения записей в базу данных мне были неизвестны[10]. После этого я подготовил еще один скрипт UpdateOperative.sql, в котором задаются значения нужных внешних ключей.

Надо сказать, что процесс определения всех значений первичных и внешних ключей оказался очень нудным и трудоемким даже для такого небольшого количества строк.

Листинг 5.6. Скрипт, выполняющий корректировку внешних ключей, ссылающихся на родителей и на супругов

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';
/* Добавление ссылок на родителей */
UPDATE PEOPLE SET CODMOTHER = 1, CODFATHER = 2 WHERE COD = 3;
UPDATE PEOPLE SET CODMOTHER = 1, CODFATHER = 2 WHERE COD = 4;
UPDATE PEOPLE SET CODFATHER = 5 WHERE COD = 6;
UPDATE PEOPLE SET CODMOTHER = 8 WHERE COD = 9;
UPDATE PEOPLE SET CODMOTHER = 10, CODFATHER = 11 WHERE COD = 12;
UPDATE PEOPLE SET CODMOTHER = 13, CODFATHER = 14 WHERE COD = 15;

/* Добавление ссылок на супругов */
UPDATE PEOPLE SET CODOTHERHALF = 2 WHERE COD = 1;
UPDATE PEOPLE SET CODOTHERHALF = 10 WHERE COD = 11;
```

[10] При использовании IBExpert просмотреть данные можно, не выдавая явно оператор SELECT. После регистрации базы данных нужно с ней соединиться, выбрав в меню **Database** элемент **Connect to Database**. Затем в окне **Database Explorer** раскрыть список таблиц, щелкнув по символу + слева от строки **Tables**. Дважды щелкнув мышью по нужному имени таблицы, мы получаем информационное окно, в котором можно просмотреть различные характеристики, в том числе и хранимые в таблице данные.

```
UPDATE PEOPLE SET CODOTHERHALF = 13 WHERE COD = 14;

COMMIT;

EXIT;
```

Вначале мы задаем диалект клиента и набор символов клиента. После этого соединяемся с базой данных и вносим в эти записи изменения в код отца и код матери или только код одного из родителей, если данные о другом родителе отсутствуют. В другой группе операторов мы добавляем сведения о супругах в некоторые строки нашей таблицы.

> ***Замечание***
>
> Кто-то может подумать, что я слукавил, заявляя, что значения кодов мне были неизвестны при помещении новых записей людей в базу данных. Ведь понятно, что первый оператор `INSERT` получит от генератора значение 1, следующий — 2 и т. д. То есть значение первичного ключа в точности равняется последовательному номеру оператора `INSERT` в списке.
>
> Все верно... если вы работаете на локальном компьютере, а не в сети. Если же параллельно с вами в сети кто-то использует этот же генератор, то уверенность, что вам каждый раз известно получаемое от генератора значение, быстро улетучивается.
>
> Пожалуй, на этом примере следует сформулировать одно хорошее правило работы с *настоящими* реляционными СУБД (впрочем, как и с любыми другими системами) — не следует создавать собственных мифов о том, как работает система, а нужно все проверять самим.
>
> По этому поводу мне вспоминается одна история, которая разрушила один из моих мифов. Когда я писал статью про транзакции, то для исследования взаимодействия транзакций с различными уровнями изоляции (эти умные слова мы с вами начнем легко и непринужденно использовать несколько позже) посадил полтора десятка студентов за клиентские компьютеры, попросив по возможности одновременно выполнить некое действие, приводящее к конфликту блокировки в базе данных на сервере. Я был уверен, что конфликт на самом деле не произойдет ни у одного клиента, поскольку предполагал (это и есть миф), что при использовании коротких обновляющих транзакций сервер вначале успешно и быстро обработает поступивший первым по времени запрос, затем второй и т. д. Результат был несколько иным и по тем временам для меня обескураживающим — мы получали в различных случаях от нуля до трех-четырех исключений по той причине, что на сервере для работы с каждым клиентом создается поток (я, естественно, использую Суперсервер — см. *приложение 1*). И как происходит переключение с одного потока на другой одному Биллу Гейтсу известно (а может быть, и нет).

Для большинства других записей людей код матери, код отца и код супруга в операторе `INSERT` определяется с помощью оператора `SELECT`. Просмотрите скрипты (листинг 5.7).

Листинг 5.7. Фрагмент скрипта для добавления человека с указанием его родителей

```
INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, BIRTHDAY, SEX,
  CODMOTHER, CODFATHER)
  VALUES (GEN_ID(GEN_PEOPLE, 1), 'СЕРГЕЙ', 'НИКОЛАЕВИЧ', 'ПИНТЕРА',
   '13.12.1988', '0',
   (SELECT COD FROM PEOPLE
     WHERE NAME1 = 'ИРИНА' AND
           NAME2 = 'ИВАНОВНА' AND
           NAME3 = 'ПИНТЕРА'),
   (SELECT COD FROM PEOPLE
     WHERE NAME1 = 'НИКОЛАЙ' AND
           NAME2 = 'НИКОЛАЕВИЧ' AND
           NAME3 = 'ПИНТЕРА' AND
           BIRTHDAY = '01.01.1967'));
```

В этом операторе добавляется запись одного человека с указанием его родителей, записи которых уже помещены в базу данных.

Здесь для получения кода матери мы используем оператор `SELECT`, в котором в условиях выборки указываем имя, отчество и фамилию матери. Оператор возвращает единственное значение столбца — первичного ключа — нужной записи, кода этой записи. По крайней мере, должен возвращать единственное значение, иначе мы получим исключение базы данных. На основании этого оператора мы действительно получим единственное значение кода, которое будет помещено во внешний ключ — код матери. Для получения же кода отца нам в соответствующем операторе `SELECT` пришлось указать еще и дату рождения, поскольку в базе данных содержится более одной записи людей, имеющих те же имя, отчество и фамилию.

В следующем фрагменте скрипта добавляется новая строка в таблицу людей (листинг 5.8). При этом с помощью операторов `SELECT` формируются значения внешних ключей, ссылающихся и на родителей, и на супруга.

Листинг 5.8. Фрагмент скрипта для добавления человека с заданием его родителей и супруга

```
INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, BIRTHDAY, SEX, CODMOTHER,
CODFATHER, CODOTHERHALF)
  VALUES (GEN_ID(GEN_PEOPLE, 1), 'НАТАЛЬЯ', 'ИЛЬИНИЧНА', 'МУРАВЬЕВА',
'25.01.1986', '1',
  (SELECT COD FROM PEOPLE
     WHERE NAME1 = 'ЕКАТЕРИНА' AND
     NAME2 = 'АНДРЕЕВНА' AND
     NAME3 = 'ЧЕРНЫШОВА'),
  (SELECT COD FROM PEOPLE
```

```
    WHERE NAME1 = 'ИЛЬЯ' AND
    NAME2 = 'НИКОЛАЕВИЧ' AND
    NAME3 = 'МУРАВЬЕВ'),
  (SELECT COD FROM PEOPLE
    WHERE NAME1 = 'АНТОН' AND
    NAME2 = 'ИВАНОВИЧ' AND
    NAME3 = 'ЕРМИШИН'));
```

Однако задача, связанная с фиксацией факта супружеских отношений, не ограничивается лишь использованием операторов добавления новых строк и изменения значений существующих внешних ключей, ссылающихся на супругов.

Существует вероятность, что по причине халатности пользователь для жены укажет одного мужа, а для этого мужа — совершенно другую жену. Или вариант, при котором мужчина будет женат, а его жена не будет об этом знать — значение соответствующего внешнего ключа в ее строке будет установлено в `NULL`. Хуже, когда в нашей благопристойной базе данных будут существовать однополые браки.

Поскольку наша задача заключается в разработке хорошо функционирующей программной системы, удобной для практически любого пользователя, обладающего определенным интеллектом (при этом на всякий случай не следует слишком полагаться на уровень этого интеллекта), то нам следует так создать наш программный продукт, чтобы он был защищен от максимально возможного количества ошибочных действий пользователя.

Для поддержания правильных связей между супругами можно использовать специально написанные триггеры, которые, во-первых, возьмут на себя защиту базы данных от неверных действий пользователя и, во-вторых, осуществят автоматическое дополнение недостающих данных. Для нормальной семьи характерны следующие моменты: супругов ровно двое — мужчина и женщина, если для мужчины указана в качестве жены конкретная женщина, то для этой женщины тот же мужчина должен быть указан как муж.

Опять же забегая немного вперед, рассмотрим вкратце триггеры, которые позволяют при добавлении новых строк поддерживать в базе данных нормальные связи между супругами.

Приведу фрагменты из скрипта CreateTriggers.sql.

Вначале нам нужно создать несколько исключений (листинг 5.9). Об исключениях мы еще раз поговорим попозже. Сейчас же можно сказать, что это некоторые создаваемые пользователем объекты базы данных, которые могут использоваться в хранимых процедурах и триггерах и позволяют прерывать

выполнение операций с базой данных, возвращая вызвавшей программе или программе, инициировавшей вызов триггера, сообщение об ошибке.

Для вызова пользовательского исключения используется оператор EXCEPTION.

Листинг 5.9. Фрагмент скрипта для создания пользовательских исключений

```
CREATE EXCEPTION SEX_PEOPLE_INSERT 'Неверный пол супруга при добавлении
записи';
CREATE EXCEPTION SEX_PEOPLE_UPDATE 'Неверный пол супруга при изменении
записи';
CREATE EXCEPTION NO_PEOPLE 'Запись человека отсутствует в базе данных';
CREATE EXCEPTION OTHER_PEOPLE 'У супруга установлена связь с другим
человеком';

COMMIT;
```

Вот триггер, который будет автоматически вызываться перед (BEFORE) добавлением (INSERT) новой строки в таблицу людей — листинг 5.10.

Листинг 5.10. Триггер, выполняющийся перед добавлением новой строки

```
/* Проверка наличия супруга и его пола
   перед помещением новой записи в базу данных */
CREATE TRIGGER TBI_PEOPLE FOR PEOPLE
  BEFORE INSERT
AS
DECLARE VARIABLE CODI INTEGER;
DECLARE VARIABLE SEXI CHAR(1);
BEGIN
  IF (NEW.CODOTHERHALF IS NOT NULL) THEN
  BEGIN
    IF (NOT EXISTS(SELECT COD
                   FROM PEOPLE
                   WHERE COD = NEW.CODOTHERHALF)) THEN
       EXCEPTION NO_PEOPLE;
    SELECT SEX, CODOTHERHALF FROM PEOPLE WHERE COD = NEW.CODOTHERHALF
      INTO :SEXI, :CODI;
    IF (NEW.SEX = SEXI) THEN
      EXCEPTION SEX_PEOPLE_INSERT;
    IF (CODI IS NOT NULL) THEN
      EXCEPTION OTHER_PEOPLE;
  END
END
```

Прежде чем запись будет помещена в базу данных, вызывается этот триггер. Если в процессе его выполнения появится исключение, строка в базу не помещается.

В самом начале проверяется, указано ли значение внешнего ключа, ссылающегося на супруга. Только в этом случае будет выполнен код триггера:

```
IF (NEW.CODOTHERHALF IS NOT NULL) THEN ...
```

Префикс `NEW` относится к контекстным переменным, допустимым в операторах при написании триггеров. Он означает, что используется новое, измененное (в нашем случае — созданное) значение столбца. Существуют контекстные переменные `OLD`, позволяющие получить значения столбцов до их изменения. `OLD` может использоваться в триггерах, где допустимы старые и новые значения. В нашем случае это будет триггер, вызываемый после добавления записи.

Далее выполняется проверка на существование в базе данных строки, на которую ссылается внешний ключ. Для этого с помощью оператора `SELECT` выбираются строки (разумеется, одна строка) с соответствующим первичным ключом. Проверка существования строк выполняется функцией `EXISTS`. То, что я назвал по привычке функцией, на самом деле называется предикатом, предикатом существования. Если строк нет, выдается исключение `NO_PEOPLE`.

Потом проверяется отличие пола найденной строки от пола добавляемой строки. Для этого с помощью оператора `SELECT` выбирается значение пола и значение внешнего ключа супруга из второй строки. Если пол первой и второй строки совпадает, то выдается исключение `SEX_PEOPLE_INSERT`.

Под конец значение внешнего ключа супруга во второй строке проверяется на пустое значение `NULL`. Если его значение не `NULL` (то есть он уже женат или она уже замужем), то выдается исключение `OTHER_PEOPLE`.

На этом проверки в триггере завершаются, но это еще не все. Пока мы только выполнили необходимые проверки для того, чтобы можно было помещать новую запись в базу данных. Теперь нужно еще выполнить изменения внешнего ключа, ссылающегося на супруга у "второй половины", т. е. поместить туда значение первичного ключа вновь создаваемой записи. Однако в описываемом триггере этого сделать нельзя, потому что новая запись еще не помещена в базу данных, такое изменение приведет к нарушению целостности данных — внешний ключ второй записи будет ссылаться на несуществующий пока первичный ключ.

Чтобы решить эту задачу, нужно написать триггер, который будет вызываться после (`AFTER`) добавления новой строки (листинг 5.11).

Листинг 5.11. Триггер, выполняющийся после добавления новой строки

```
/* Корректировка кода супруга во второй строке после добавления записи */
CREATE TRIGGER TAI_PEOPLE FOR PEOPLE
  AFTER INSERT
AS
BEGIN
  IF (NEW.CODOTHERHALF IS NOT NULL) THEN
    UPDATE PEOPLE SET CODOTHERHALF = NEW.COD
      WHERE COD = NEW.CODOTHERHALF;
END
```

Здесь как раз и корректируется значение внешнего ключа у второго супруга.

На самом деле задача несколько сложнее, чем то решение, которое мы с таким трудом нашли. Нужно еще написать триггеры, вызываемые до и после изменения кода супруга, чтобы проверить допустимость изменения и выполнить симметричное изменение в записи второго супруга. Здесь есть хорошая возможность получить бесконечный цикл обновлений, когда изменение кода супруга в одной записи с помощью триггера приводит к изменению кода во второй записи, что приводит к вызову триггера, который будет изменять код первой записи... и т. д.

Когда мы будем заниматься триггерами и хранимыми процедурами, то подробно рассмотрим весь этот процесс.

Есть другие варианты решения задач содержательной (в отличие от формальной, декларативной) непротиворечивости данных.

Один из вариантов заключается в написании хранимых процедур, которые должны быть явно вызваны при добавлении новых строк или при изменении значений внешних ключей в существующих записях. В этом случае у вас больше возможностей контролировать ситуацию, связанную с проверками, нужными установками значений; проще предотвратить зацикливание процесса обновления данных.

Другой вариант — написание нормальной программы, в которой пользователю предлагается не только удобный для него интерфейс взаимодействия с базой данных, но и допустимые варианты выбора соответствующих значений, набора строк. Например, для добавления сведений о супруге пользователю можно предложить для выбора список людей соответствующего (то есть противоположного) пола. При этом надобность в дополнительной проверке правильности выбранного пола отпадает.

Надо полагать, всем понятно, что именно этот, второй, вариант является наиболее удобным для нашего пользователя. В этом направлении и следует приложить наши усилия.

Скрипт содержит также операторы добавления данных по сотрудникам некоторых организаций и по правонарушениям людей. Фрагмент скрипта добавления сотрудников показан в листинге 5.12.

Листинг 5.12. Добавление некоторых сотрудников организаций

```
INSERT INTO STAFF (COD, CODPEOPLE, CODORG, DUTIES, SALARY)
  VALUES (GEN_ID(GEN_STAFF, 1), 14, 11, 'Генеральный директор', 56000);
INSERT INTO STAFF (COD, CODPEOPLE, CODORG, DUTIES, SALARY)
  VALUES (GEN_ID(GEN_STAFF, 1), 14, 18, 'Исполнительный директор',
38500);
INSERT INTO STAFF (COD, CODPEOPLE, CODORG, DUTIES, SALARY)
  VALUES (GEN_ID(GEN_STAFF, 1), 14, 16, 'Ночной сторож', 1500);
...
INSERT INTO STAFF (COD, CODPEOPLE, CODORG, DUTIES, SALARY)
  VALUES (GEN_ID(GEN_STAFF, 1), NULL, 18, 'Неизвестная должность1',
NULL);
INSERT INTO STAFF (COD, CODPEOPLE, CODORG, DUTIES, SALARY)
  VALUES (GEN_ID(GEN_STAFF, 1), NULL, 18, 'Неизвестная должность2',
NULL);
INSERT INTO STAFF (COD, CODPEOPLE, CODORG, DUTIES, SALARY)
  VALUES (GEN_ID(GEN_STAFF, 1), NULL, 18, 'Неизвестная должность3',
NULL);
```

Оказалось, что один и тот же человек работает в разных должностях в трех различных организациях.

Последние три строки добавляют троих неизвестных сотрудников — внешний ключ, код человека, имеет пустое значение. Не известны также их оклады и названия должностей. Нам эти загадочные люди понадобятся в дальнейшем для иллюстрации некоторых особенностей поиска данных.

Теперь о грустном. Следующий фрагмент скрипта содержит сведения о неблаговидной деятельности одной нехорошей семейки (листинг 5.13). В преступную деятельность были вовлечены мать, отец и двое их детей. Сведения предоставлены комиссией по делам несовершеннолетних. Имена, фамилии, даты и виды правонарушений изменены.

Листинг 5.13. Добавление данных о правонарушениях людей

```
INSERT INTO PEOPLEADMIN (COD, CODPEOPLE, CODADMIN, DATEADMIN)
  VALUES (GEN_ID(GEN_PEOPLEADMIN, 1), 2, '06.08', '07.12.1998');
INSERT INTO PEOPLEADMIN (COD, CODPEOPLE, CODADMIN, DATEADMIN)
  VALUES (GEN_ID(GEN_PEOPLEADMIN, 1), 2, '06.08', '01.01.1999');
```

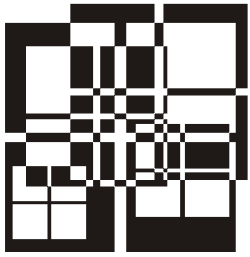
```
INSERT INTO PEOPLEADMIN (COD, CODPEOPLE, CODADMIN, DATEADMIN)
  VALUES (GEN_ID(GEN_PEOPLEADMIN, 1), 1, '06.11', '01.03.2000');
INSERT INTO PEOPLEADMIN (COD, CODPEOPLE, CODADMIN, DATEADMIN)
  VALUES (GEN_ID(GEN_PEOPLEADMIN, 1), 3, '07.27', '09.09.2006');
INSERT INTO PEOPLEADMIN (COD, CODPEOPLE, CODADMIN, DATEADMIN)
  VALUES (GEN_ID(GEN_PEOPLEADMIN, 1), 4, '07.17', '01.01.2000');
```

Что там за перевалом?

Добавление, изменение и удаление данных — довольно интересные задачи, в чем мы с вами только что убедились. Временами сталкиваешься с немалыми трудностями, которые мы преодолеваем, используя всю мощь реляционных баз данных, наши собственные знания и собственный врожденный ум.

Наиболее важное и интересное в реляционных базах данных — поиск, выборка данных. Это осуществляется с помощью оператора `SELECT`, который мы рассмотрим чуть позже. А сейчас — чувствую, что вы уже соскучились, — перейдем к написанию настоящих программ.

ГЛАВА 6



Для особо одаренных. Написание программ работы с базой данных

Сейчас мы начнем реализовывать профессиональный программистский подход к использованию баз данных. Мы напишем великое множество полезных программ, с которыми сможет работать человек, вовсе незнакомый с премудростями языка SQL.

В самом начале, прежде чем приступить к разработке программ, для того, чтобы они получились по-настоящему хорошими, необходимо проникнуться чувством любви к вашим будущим пользователям. Ну ладно, хотя бы уважением.

В первых программах я опишу обычные процедуры использования строки состояния и выдачи подсказок. Во всех других программах вы при необходимости также сможете реализовать эти маленькие радости для ваших будущих клиентов.

Каждую программу я напишу дважды. Первая будет использовать компоненты FIBPlus, вторая — IBX. Вы можете использовать либо обе версии (почувствуйте разницу!), либо только одну из них.

Еще один дружеский совет — никогда в работе с базами данных не используйте компоненты BDE (Borland Database Engine). Причин к тому множество. Одна из них — Borland остановил развитие этих компонентов.

Чтобы нормально работали все ваши программы, вы должны создать в базе данных безопасности учетную запись пользователя `WIZARD` с паролем master *(см. главу 2)*. Этот пользователь является владельцем нашей учебной базы данных Work.fdb. Необходимо создать и саму базу данных, используя скрипты, которые вы скачали с сайта издательства.

Выполняемые программы (EXE-файлы) будут нормально работать с базой данных, если вы создадите базу данных Work.fdb на диске D: вашего компьютера. То есть расположение вашей базы данных должно быть:

D:\BestDatabase\Work.fdb

Разумеется, у вас должен быть запущен сервер базы данных InterBase или Firebird.

Работу по созданию программ мы будем выполнять по принципу от простого к сложному. Первые программы будут достаточно простыми. Последующие же будут дополнять и улучшать функциональность первых программ. Мы освоим способы выполнения просмотра данных, удаления, добавления и изменения существующих данных. Отработаем технологию программной реализации отображения отношения между таблицами главная — подчиненная. Кроме того, мы научимся легко и просто изменять упорядоченность отображаемых наборов данных.

В дальнейших главах мы закрепим наши успехи, знания и умения работать с базами данных и напишем еще ряд полезных программ.

6.1. Программа просмотра и удаления стран

6.1.1. Использование компонентов FIBPlus

Запустите Delphi 7. Создайте новый проект (меню **File | New | Application**). Для появившейся формы в Инспекторе объектов задайте свойства:

- `Caption` — `Справочник стран`
- `Name` — `FormMain`
- `WindowState` — `wsMaximized`

Сохраните модуль с именем `Main` (меню **File | Save As** (Файл | Сохранить как)). Сохраните в каком-нибудь каталоге проект с именем DisplayDelete (**File | Save Project As** (Сохранить проект как)).

Положите на форму панель, выровняв по верхнему краю, и очистите ее свойство `Caption`. Установите свойство `ShowHint` в `True`, чтобы в строке состояния появлялись подсказки при наведении пользователем мыши на некоторые элементы, размещенные на этой панели. При этом также будут появляться "хинты" (hint) рядом с курсором мыши — тексты подсказок на желтом фоне, заключенные в прямоугольник.

Поместите с вкладки **Win32** компонент (на форму, а не на панель!) `StatusBar`. Он будет автоматически выровнен по нижнему краю формы.

Щелкните на этом компоненте правой кнопкой мыши и в контекстном меню выберите элемент **Panels Editor** (Редактор панелей). В появившемся окне редактора панелей добавьте три элемента (щелчок мышью по кнопке **Add New** или нажатие клавиши <Ins>).

Для первого элемента установите следующие свойства:

- `Bevel` = `pbNone`
- `Text` = `Количество стран:`
- `Width` = `110`

Для второго только:

- `Alignment` = `taRightJustify`

Во второй элемент будет помещаться количество считанных строк набора данных.

Для третьего элемента можете не изменять никакие свойства. В это поле будут помещаться тексты подсказок, когда пользователь подведет курсор мыши к одному из элементов, содержащему текст подсказки.

Напишите процедуру, отображающую на панели `StatusBar` подсказки (Hint), которые должны появляться при перемещении курсора мыши по элементам меню и кнопкам панели инструментов.

Для этого в разделе **`type`** в части личных объектов (**`private`**) нужно написать прототип процедуры:

```
procedure ShowHint(Sender: TObject);
```

Сама процедура должна располагаться в части **`implementation`** после вот этих двух строк, созданных автоматически:

```
implementation
{$R *.dfm}
```

Процедура содержит только один выполняемый оператор (листинг 6.1).

Листинг 6.1. Процедура отображения подсказок

```
procedure TFormMain.ShowHint(Sender: TObject);
begin
  StatusBar1.Panels[2].Text := Application.Hint;
end;
```

Эта процедура будет помещать в третий элемент строки состояния (элементы нумеруются, начиная с нуля) тексты подсказок, когда курсор мыши окажется на элементе, содержащем текст подсказки (Hint).

ВНИМАНИЕ!

В этом и только в этом случае, когда вы полностью создаете свою процедуру или функцию, вам нужно написать все строки процедуры и поместить прототип этой процедуры в раздел **type**. Во всех остальных случаях, когда вы пишете обработчики каких-либо событий, заготовку для обработчика вы получаете автоматически после двойного щелчка мышью справа от названия соответствующего события на вкладке **Events** вашего компонента. Прототип для обработчика события в разделе **type** (предварительное определение процедуры, требуемое в языке Delphi) также будет создан автоматически. Вам нужно только написать операторы процедуры между строками **begin** и **end**.

Поместите на форму компонент главного меню (`MainMenu`). Создайте элементы меню, дважды щелкнув мышью по компоненту (рис. 6.1).

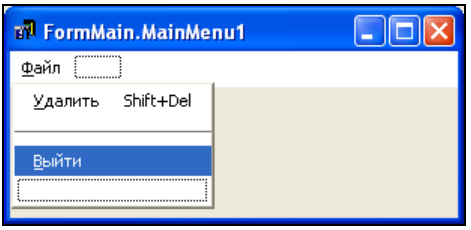


Рис. 6.1. Главное меню программы

Элементу меню **Файл** задайте имя `MFile` (свойство `Name`), **Удалить** — `MDelete`, **Выйти** — `MExit`. Для элемента **Удалить** задайте клавиши быстрого доступа, установив свойство `ShortCut` в `Shift+Del`, свойство `Hint` (текст подсказки)
установите в `Удаление текущей записи`. Для элемента **Выйти** задайте `Hint` = `Выход из программы`.

В свойстве `Caption` элементов меню каждый текст начинайте с символа &. Это позволит сделать первые символы подчеркнутыми. В дальнейшем при работе программы пользователь сможет обращаться к элементам меню, используя клавишу <Alt> и соответствующую букву на клавиатуре.

На панель положите две кнопки (компоненты `Button` с вкладки **Standard**). Для одной установите значение `Caption` в `Удалить` (задайте имя `BDelete`), для другой — `Выйти` (имя `BExit`).

Для кнопки **Удалить** задайте в свойстве `Hint` (подсказка) следующий текст:

```
Удалить (Shift+Del)|Удаление текущей записи
```

Первая часть текста, до вертикальной черты, будет на некоторое время появляться рядом с курсором мыши, когда пользователь подведет его к кнопке. Вторая часть текста будет в этот момент появляться в третьей панели строки состояния. Это произойдет благодаря написанной нами процедуре, содержа-

щей один оператор `ShowHint`. Кроме того, в начале работы программы нужно указать системе, что для вывода подсказок нужно использовать именно эту процедуру:

```
Application.OnHint := ShowHint;
```

Аналогично, для кнопки **Выход** задайте в `Hint` текст:

```
Выйти|Выход из программы
```

С вкладки **FIBPlus** положите на форму компоненты: `Database`, `Transaction` (два компонента) и `DataSet`. Компоненту базы данных присвойте имя `Database1`. Транзакциям присвойте имена `ReadTransaction` и `WriteTransaction`. Набору данных присвойте имя `DataSet1`.

Для `Database` установите следующие значения в Инспекторе объектов.

В свойствах `ConnectParams`:

- `CharSet` = `WIN1251` (набор символов по умолчанию);
- `Password` = `master` (пароль);
- `UserName` = `wizard` (имя пользователя — в любом регистре).

В результате задания этих параметров соединения с базой данных автоматически будут созданы следующие параметры в свойстве `DBParams` компонента `Database`:

```
lc_ctype = WIN1251;
password = master;
user_name = wizard;
```

Остальные свойства будут выглядеть так:

- имя компонента (`Name`): `Database1`;
- `DBName` = `D:\BestDatabase\Work.fdb`, вы можете не набирать этот путь вручную, а щелкнув справа от этого свойства по кнопке  , в диалоговом окне выбрать нужный каталог и файл;
- `DefaultTransaction` = `ReadTransaction` (транзакция по умолчанию — транзакция, в контексте которой будут выполняться все операции чтения);
- `DefaultUpdateTransaction` = `WriteTransaction` (транзакция, в контексте которой будут выполняться все обновления — добавление, изменение и удаление данных).

Для компонента `DataSet` установите следующие свойства:

- `AutoCommit` = `True` (автоматическое подтверждение транзакции сразу после выполнения обновлений данных);

- `Database` = `Database1` (связать этот компонент с компонентом базы данных). Не вводим вручную, а выбираем из списка;
- раскрыв свойство `PrepareOptions` (щелчок по знаку + слева от названия этого свойства), установите в `True` подсвойство `psAskRecordCount`. Это позволит получить фактическое количество записей в справочнике, а не то, которое было считано (английский термин fetch) на момент открытия набора данных. Это число помещается в свойство `RecordCount` компонента `DataSet`;
- `Transaction` = `ReadTransaction` (транзакция для чтения);
- `UpdateTransaction` = `WriteTransaction` (транзакция для обновлений).

Для обоих компонентов транзакции можно не устанавливать никаких значений (кроме имен). Нас вполне устроит значение всех свойств по умолчанию. По крайней мере, сейчас. Мы сразу используем разделенные транзакции для чтения и для изменения данных. Это будет хорошей привычкой. В дальнейшем мы подробно рассмотрим характеристики таких пар транзакций и убедимся, насколько бывает полезным использовать разделенные транзакции. Правда, это может использоваться только в компонентах FIBPlus, а не в IBX.

Выделите мышью на форме компонент `DataSet`. Щелкните справа от свойства `SQLs` по кнопке [...] или щелкните правой кнопкой мыши по компоненту `DataSet` и выберите в контекстном меню вариант **SQL Generator** (генератор операторов SQL). И в том и в другом случае появится окно редактирования запросов SQL. В правой части окна в разделе **Tables/Views** (таблицы/представления) в упорядоченном списке выберите имя нужной нам таблицы `REFCTR` и дважды щелкните мышью по этой строке. В левой части появится сгенерированный оператор `SELECT`. Последней строкой в этом операторе добавьте вручную предложение `ORDER BY NAME`, чтобы упорядочить полученный набор данных по кратким названиям стран.

> ***ВНИМАНИЕ!***
>
> Если эта форма выглядит несколько иначе (просто в текущей вкладке содержит одно окно для ввода оператора `SELECT`), то вы, скорее всего, не связали этот компонент набора данных с компонентом базы данных. Если же вы получили вполне приличный вид окна, как показано на рис. 6.2, однако список таблицы/представления (**Tables/Views**) пустой, то у вас явно ошибки в описании компонента базы данных — чаще всего неверно задан путь к файлу базы данных либо неверно записаны имя и пароль пользователя.
>
> Исправьте ошибки в соответствующих компонентах и снова приступайте к созданию нужных операторов SQL.

Здесь мы видим, что оператор `SELECT` выбирает все столбцы из всех строк нашей таблицы `REFCTR`. Этот оператор SQL будет выполняться при вызове

метода `Open` компонента `DataSet` или при установлении в `True` значения его свойства `Active`. В результате выбранные из базы данных на сервере строки будут размещены на нашем клиентском компьютере в виде *набора данных*. Этот набор данных и данные в базе данных на сервере будут обновляться в процессе изменения данных пользователем нашей программы, когда он будет добавлять, изменять или удалять данные.

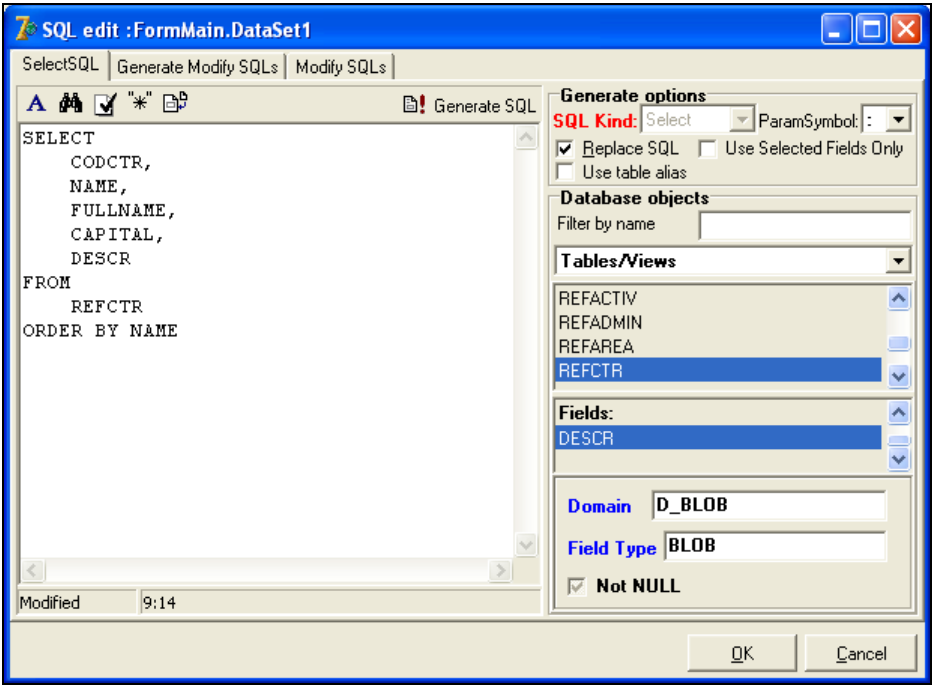


Рис. 6.2. Окно редактирования запросов SQL

Выберите вкладку **Generate Modify SQLs** (Генерация модифицирующих операторов SQL). Убедитесь, что в выпадающем списке слева выбран вариант **Generate All SQLs** (Генерировать все операторы SQL). Щелкните по кнопке **Get Table Fields** (Получить поля таблицы) для получения списка всех столбцов таблицы (скорее всего, они и так уже присутствуют в правом списке — **Update Fields** (Изменяемые поля)).

Убедитесь, что в левом списке столбцов **Key Fields** (Ключевые поля) отмечен только один столбец первичного ключа `CODCTR`, а в правом списке столбцов **Update Fields** отмечены все столбцы, как показано на рис. 6.3. Снимите флажок **Non update primary key** (Не изменять первичный ключ), чтобы разрешить изменения первичного ключа, убедитесь, что в списке **ParamSymbol** (Символ признака параметра) заданы символы двоеточия (**:**) или вопроси-

тельного знака (**?**). Обратите внимание, что в левом списке, содержащем ключевые столбцы, нет столбца `DESCR`. Этот столбец у нас имеет тип данных `BLOB`, а `BLOB` не может выступать в качестве какого-либо ключа.

Щелкните по кнопке **Generate SQLs** (Генерировать операторы SQL). Будут сгенерированы все операторы для добавления, изменения, удаления и повторного чтения записей таблицы.

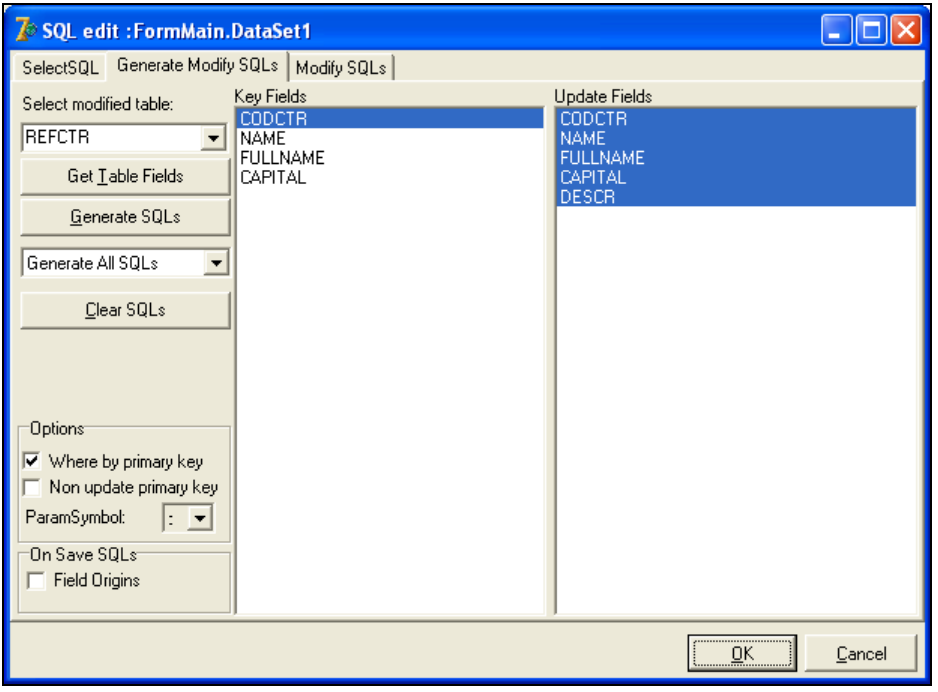


Рис. 6.3. Окно генерации модифицирующих запросов SQL

Текущей вкладкой станет **Modify SQLs**. Любопытства ради посмотрите на все сгенерированные операторы (листинг 6.2).

Листинг 6.2. Оператор добавления новой строки

```
INSERT INTO REFCTR(
    CODCTR,
    NAME,
    FULLNAME,
    CAPITAL,
    DESCR
)
VALUES(
```

```
    :CODCTR,
    :NAME,
    :FULLNAME,
    :CAPITAL,
    :DESCR
)
```

Это обычный оператор `INSERT`. Необычным для нас является только список значений. Здесь указываются имена параметров, которые будут содержать нужные значения. Перед именем каждого параметра стоит двоеточие. Если же вы выбрали в списке **ParamSymbol** знак вопроса, то перед именами параметров будет стоять не двоеточие, а именно знак вопроса, что, откровенно говоря, не имеет никакого особого смысла. Сами значения для вновь создаваемой строки будут переданы при вызове метода `Insert` или `Append` компонента `DataSet`.

Листинг 6.3. Оператор изменения одной строки

```
UPDATE REFCTR
SET
    CODCTR = :CODCTR,
    NAME = :NAME,
    FULLNAME = :FULLNAME,
    CAPITAL = :CAPITAL,
    DESCR = :DESCR
WHERE
    CODCTR = :OLD_CODCTR
```

В листинге 6.3 новые значения также передаются в виде параметров. Конкретные значения будут создаваться при вызове метода `Edit` компонента `DataSet`. Обратите внимание на предложение `WHERE`. Оно определяет, что изменения касаются одной записи, у которой первичный ключ равен значению первичного ключа текущей записи *до изменения* этого значения пользователем (перед именем параметра стоит префикс `OLD_`). Говоря человеческим языком, это та самая текущая запись в наборе данных, которую только что изменил пользователь. Поскольку пользователь мог изменить и первичный ключ, здесь передается параметр, который содержит значение первичного ключа до его изменения.

Напомню, что в InterBase и Firebird наличие первичного ключа в таблице не является строго обязательным, а стандарт требует присутствия первичного ключа в каждой таблице. Лучше следовать стандарту, тогда не будет больших проблем, связанных с тем, как однозначно идентифицировать текущую запись для изменения, удаления или обновления. Альтернативный вариант

(если не задан первичный ключ) — выбор всех полей таблицы. Тогда предложение WHERE выглядело бы следующим образом:

```
WHERE
    CODCTR = :OLD_CODCTR
    and NAME = :OLD_NAME
    and FULLNAME = :OLD_FULLNAME
    and CAPITAL = :OLD_CAPITAL
```

Нельзя сказать, что это хорошо и красиво, однако тоже работает.

Оператор удаления одной строки показан в листинге 6.4.

Листинг 6.4. Оператор удаления текущей строки

```
DELETE FROM
    REFCTR
WHERE
        CODCTR = :OLD_CODCTR
```

Думаю, здесь все понятно. Удаляемая строка также определяется в предложении WHERE, ссылающемся на первичный ключ. (Мне только не совсем понятно, зачем здесь-то в имени параметра используется префикс OLD_, ведь нет ни "OLD", ни "NEW" — в смысле ни старой, ни новой, — а существует ровно одна текущая запись, которая подлежит удалению; казалось бы, префикс можно совсем убрать. Если у вас есть время и желание — поэкспериментируйте немного.)

Наконец, оператор обновления текущей записи — листинг 6.5.

Листинг 6.5. Оператор обновления текущей записи

```
SELECT
    CODCTR,
    NAME,
    FULLNAME,
    CAPITAL,
    DESCR
FROM REFCTR
 WHERE
        REFCTR.CODCTR = :OLD_CODCTR
```

Назначение этого оператора — считать с сервера ровно одну добавленную или измененную текущую запись, которая после чтения включается в набор данных, доступный нашей программе. Оператор вызывается после каждого

добавления или изменения записи (точнее — после выполнения метода Post компонента DataSet).

Щелкните по кнопке **OK**, чтобы сохранить выполненные действия.

С вкладки **DataAccess** поместите на форму компонент DataSource. Его свойство DataSet установите в DataSet1 (выберите из выпадающего списка).

С вкладки DataControls поместите на форму DBGrid — сетку, в которой будут отображаться записи полученного набора данных. Его свойство DataSource установите в DataSource1, связав таким образом сетку с набором данных. Выравнивание компонента (Align) установите в alClient — компонент будет занимать всю свободную площадь формы от инструментальной панели до строки состояния.

Отредактируйте характеристики DBGrid. Щелкните правой кнопкой мыши на компоненте и в контекстном меню выберите **Columns Editor** (Редактор столбцов). Появится небольшое окно редактирования столбцов. Щелкните мышью по кнопке добавления нового столбца (**Add New**) или нажмите клавишу <Ins>. В списке появится первый столбец. Установите его характеристики.

- FieldName — из выпадающего списка выберите CODCTR. Это код страны.

> ***ВНИМАНИЕ!***
> Если выпадающий список пустой, значит, у вас были ошибки при задании свойств компонентов Database или DataSet. Проверьте и исправьте свойства этих компонентов. Возможно также, что вы не связали DBGrid, DataSource и DataSet. Выполните необходимые установки свойств, как было описано выше.

- Свойство Alignment первого столбца установите в taCenter. Коды стран будут размещаться в сетке по центру соответствующего столбца.
- Раскройте свойство Title (заголовок), щелкнув по символу + слева от свойства. Установите следующие значения:
  - Alignment = taCenter, текст заголовка будет размещен по центру,
  - Caption = Код, это и есть текст заголовка;
  - шрифт (Font) сделайте жирным, щелкнув справа от этого свойства по кнопке  и в диалоговом окне задав жирное начертание.

Характеристики шрифта можно задать и другим способом. Раскройте свойство Font, щелкнув по символу + слева от имени свойства. Таким же образом далее раскройте свойство Style (понятно — стиль шрифта). Установите в True значение свойства fsBold.

Замечание

Некоторые из моих студентов изощряются, играясь со шрифтами в различных элементах создаваемой формы — задавая различные имена шрифтов, цвета, устанавливая подчеркивание, курсив и т. д. Если вам все еще хочется делать то же самое, дождитесь того момента, когда это пройдет, и только после этого приступайте, наконец, к написанию серьезных программ.

Для нормальных программ обычно вполне подойдет тот шрифт, который предлагается по умолчанию. В Windows это MS Sans Serif с размером 8. Некоторые элементы, как, например, заголовки, можно выделить полужирным шрифтом. Не советую использовать курсив — на экране монитора он смотрится не слишком презентабельно и разборчиво. При формировании печатных отчетов для большинства текстов подойдет шрифт Times New Roman. Заголовочные части отчетов я обычно выделяю полужирным шрифтом Arial, но иногда использую Arial Narrow, особенно если текст достаточно объемен. Этот шрифт довольно плотный и позволяет на той же полосе печатного листа разместить больше текста.

Добавьте еще один столбец, установите его свойства:

- `FieldName` = `NAME` (опять выбираем из выпадающего списка);
- `Caption` = `Краткое` название страны;
- задайте для шрифта жирное начертание. Выравнивание заголовка — по центру.

Добавьте третий столбец:

- `FieldName` = `FULLNAME`;
- `Caption` = `Полное название`;
- задайте для шрифта жирное начертание. Выравнивание заголовка — по центру.

Добавьте четвертый столбец:

- `FieldName` = `CAPITAL`;
- `Caption` = `Столица`;
- задайте для текста выравнивание по центру, а для шрифта жирное начертание.

Отметив компонент сетки (`DBGrid`), измените некоторые его свойства. Раскройте его свойство `Options`, щелкнув мышью слева от названия свойства по символу +. Установите следующие его подсвойства:

- `dgEditing` = `False`, чтобы запретить пользователю изменять данные на сетке;
- `dgIndicator` = `False`, чтобы убрать из сетки столбец индикатора слева (лично мне этот индикатор не нравится, у вас же могут быть собственные

эстетические взгляды на внешний вид вашего программного продукта. Индикатор бывает полезным, если вы допускаете множественный выбор строк на сетке; чуть позже, доводя до блеска совершенства нашу программу, мы вернем этот индикатор);

- `dgRowSelect` = `True`, чтобы на сетке была отмечена вся текущая строка, а не отдельное поле;
- `dgAlwaysShowSelection` = `True`, чтобы всегда текущая строка была отмеченной, даже если фокус ввода находится на другом элементе формы.

Установите значение свойства `ReadOnly` в `True`, чтобы пользователь не мог ни изменять, ни добавлять, ни удалять данные непосредственно на сетке. Для этого вы со временем напишете собственный код. Некоторые программисты любят предоставлять своим пользователям возможность редактирования данных на сетке. Лично мне такая практика не очень нравится, а точнее — совсем не нравится. Во-первых, далеко не все данные таким образом можно изменить, например, `BLOB`. Во-вторых, в этом случае меньше возможностей проконтролировать вводимые пользователем данные. И, в-третьих, довольно сложно предоставить пользователю достаточный комфорт при вводе данных — выпадающие списки, переключатели, флажки, хотя многое из этого тоже возможно.

Измените мышью непосредственно на сетке или задавая нужные числа в редакторе столбцов размеры столбцов, чтобы получить приблизительно следующий вид формы — рис. 6.4.

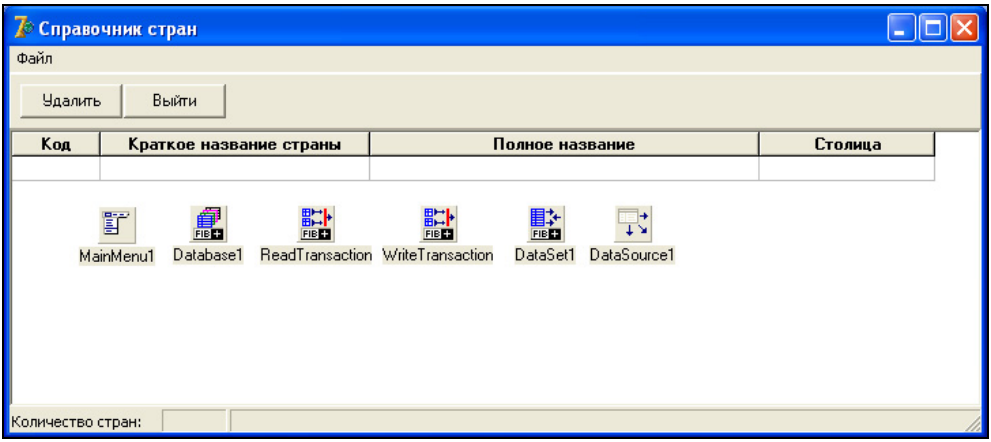


Рис. 6.4. Окончательный вид проекта

Выделите форму и напишите обработчик события активации формы (`OnActivate`), где следует соединиться с базой данных, открыть набор данных и записать в строке состояния количество записей набора данных.

> **ВНИМАНИЕ!**
> Поскольку вся наша форма закрыта различными компонентами, то для того, чтобы выделить форму, вам нужно щелкнуть по любому компоненту и нажимать клавишу <Esc> до тех пор, пока в Инспекторе объектов не появится сама форма. Факт появления формы вы сможете увидеть в самой верхней строке Инспектора объектов. Вы также можете из выпадающего списка компонентов в верхней строке Инспектора объектов выбрать форму.

Обработчик события активации формы будет следующим — листинг 6.6.

Листинг 6.6. Обработчик события активации формы

```
procedure TFormMain.FormActivate(Sender: TObject);
begin
  Database1.Connected := True;     // Database1.Open;
  DataSet1.Active := True;         // DataSet1.Open;
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
  Application.OnHint := ShowHint;
end;
```

Здесь мы соединяемся с базой данных, открываем набор данных, помещаем количество стран в строку состояния и указываем, что обработку отображения подсказок должна выполнять написанная нами процедура `ShowHint`.

Напомню, что для соединения с базой данных и для открытия набора данных мы также можем использовать методы `Open` этих компонентов, как указано в примечаниях в соответствующих строках. Хотя названия методов совпадают, надо понимать, что это разные методы и выполняют совершенно различные функции — один соединяет нас с базой данных, другой считывает записи набора данных.

В обработчике события формы `OnClose` (при закрытии или, что то же самое, при завершении работы) выполните закрытие набора данных и отключение от базы данных (листинг 6.7).

Листинг 6.7. Обработчик события закрытия формы

```
procedure TFormMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  DataSet1.Active := False;        // DataSet1.Close;
  Database1.Connected := False;    // Database1.Close;
end;
```

Точно так же можно закрыть набор данных и отсоединиться от базы данных, используя методы `Close` этих компонентов. Это показано в комментариях.

Напишите обработчик события выбора элемента меню **Выйти**. Сделайте двойной щелчок мышью по компоненту главного меню, затем двойной щелчок по элементу **Выйти** (листинг 6.8).

Листинг 6.8. Завершение работы программы

```
procedure TFormMain.MExitClick(Sender: TObject);
begin
  Application.Terminate;
end;
```

Вообще говоря, для завершения программы можно выполнить просто закрытие главной формы приложения, т. е. в обработчике этого события можно было бы написать `Close` или, чуть более правильно, `FormMain.Close`. Однако я припоминаю, что в каких-то версиях C++Builder или Delphi это не срабатывало, если программа работала с базами данных.

Тот же обработчик выберите и для события щелчка по кнопке **Выйти**. Для этого выделите на форме кнопку **Выйти**, в Инспекторе объектов перейдите к вкладке **Events**, щелкните по событию `OnClick` и справа в выпадающем списке выберите обработчик `MExitClick`.

> ***ВНИМАНИЕ!***
> Еще раз хочу напомнить — прежде чем писать обработчики событий или связывать событие с каким-либо существующим обработчиком, убедитесь, что на форме выделен именно тот компонент, событие которого вам нужно обрабатывать. Что удивительно, это весьма распространенная ошибка — писать обработчики событий не для тех компонентов, для которых они были задуманы. Часто в результате ошибок программиста программа завершает работу, когда пользователь щелкает мышью не по кнопке закрытия, а по самой форме.

Для выполнения удаления записи нужно написать обработчик события щелчка по кнопке **Удалить**. Сначала нужно проверить, не является ли наш набор данных пустым. Если да, следует выйти из процедуры удаления. В противном случае после подтверждения пользователем удаления и удаления записи нужно скорректировать количество записей, выводимое в строку состояния (листинг 6.9).

Листинг 6.9. Процедура удаления текущей строки

```
procedure TFormMain.BDeleteClick(Sender: TObject);
begin
  if DataSet1.RecordCount = 0 then exit;
  if Application.MessageBox('Удалить текущую запись?',
```

```
    'Удаление', MB_YESNO + MB_ICONQUESTION) = mrYes then
    DataSet1.Delete;
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
end;
```

Свяжите с этим обработчиком щелчок мышью по элементу меню **Удалить**. Для этого выполните двойной щелчок мышью по компоненту главного меню, выделите в диалоговом окне редактирования меню элемент **Удалить**, перейдите в Инспекторе объектов на вкладку **Events**, щелкните по событию `OnClick` и из выпадающего списка справа выберите в качестве обработчика имя уже созданного обработчика события щелчка по кнопке удаления `BDeleteClick`.

> ***Замечание***
>
> Обратите внимание, что в одном случае мы писали обработчик события выбора элемента меню, а затем связывали с этим событием щелчок по соответствующей кнопке. В другом случае поступили наоборот. Должен сказать, что это нехорошая практика. Лучше все-таки писать обработчики событий для одной группы компонентов, например, для главного меню, а затем связывать с этими обработчиками щелчки мышью по соответствующим кнопкам и выборы элементов контекстного меню. За основу надо брать именно главное меню, поскольку оно по правилам хорошего интерфейса должно содержать *все* необходимые функциональные элементы. Кнопки быстрого доступа и элементы контекстного меню не всегда в полном объеме дублируют эти функции.

Запустите программу на выполнение, исправьте ошибки. Проверьте выполняемые функции. Заметьте, что для пользователя вашей программы существует множество способов выполнить удаление. Можно щелкнуть по кнопке удаления, выбрать в меню элемент удаления, нажать клавишу <Shift> и, не отпуская ее, клавишу <Del>. Кроме того (помните символ & перед именами элементов меню?), можно нажать клавишу <Alt>, клавишу с русской буквой "Ф", при этом раскроется главное меню. После этого нажать клавишу с русской буквой "У".

Что ни говорите, есть в Windows много избыточного, однако это кому-то может пригодиться. По этой причине не стесняйтесь создавать в своих программах достаточно богатый интерфейс. Попробуйте, например, поработать с программой, не используя мышь, а лишь клавиатуру. Для большинства хороших программ это вполне реально и для пользователя достаточно комфортно.

Последний штрих

Многим из нас свойственна тяга к совершенству. Размышляя о написанной программе, мы можем задаться мыслью, а если пользователю понадобится удалять большое количество записей, неужели мы заставим его удалять все

это множество по одной строке? Тут самое время вспомнить о том, с какими чувствами к нашему будущему пользователю мы приступили к написанию нашей программы — с чувством любви или как минимум уважения.

Давайте предоставим ему возможность выделять на сетке несколько записей и удалять все отмеченные. В нашей демонстрационной базе данных это может быть не так уж актуально, поскольку тут существует всего несколько стран, однако при работе с большими по объему наборами данных эта возможность может оказаться весьма полезной для нашего пользователя. В ответ от него мы можем получить чувство большой благодарности.

Для `DBGrid` в свойстве `Options` установите в `True` значения следующих подсвойств: `dgIndicator` (разместит в левой части сетки индикатор, который я все-таки недолюбливаю) и `dgMultiSelect` (разрешит множественный выбор строк на сетке). На индикаторе будут появляться дополнительные символы, если запись является отмеченной или если запись является отмеченной и текущей в то же самое время. Такими символами будут жирная точка, символ > или совмещение этих двух символов.

Процедуру удаления перепишите в следующем виде — листинг 6.10.

Листинг 6.10. Процедура удаления текущей строки или группы отмеченных строк

```
procedure TFormMain.BDeleteClick(Sender: TObject);
var i: Integer;
begin
  if DataSet1.RecordCount = 0 then exit;
  if DBGrid1.SelectedRows.Count > 1 then
  begin
    if Application.MessageBox(PAnsiChar('Удалить отмеченные записи: ' +
       IntToStr(DBGrid1.SelectedRows.Count) + '?'),
       'Удаление', MB_OKCANCEL + MB_ICONQUESTION) = mrOk then
    begin
      for i := 0 to DBGrid1.SelectedRows.Count – 1 do
      begin
        DataSet1.GotoBookmark(pointer(DBGrid1.SelectedRows.Items[i]));
        DataSet1.Delete;
      end;
    end;
  end
  else
  begin
    if Application.MessageBox('Удалить текущую запись?',
       'Удаление', MB_YESNO + MB_ICONQUESTION) = mrYes then
```

```
      DataSet1.Delete;
  end;
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
end;
```

Здесь вначале проверяется, отмечено ли несколько записей — проверяется свойство сетки `DBGrid1.SelectedRows.Count`, которое содержит количество отмеченных строк на сетке. В этом случае выдается запрос на удаление записей с указанием их количества. О явном преобразовании текста к типу `PAnsiChar` в случае конкатенации нескольких строк для метода `MessageBox` мы говорили ранее в *главе 1*. Затем в цикле просматриваются все отмеченные строки и удаляются соответствующие записи. Чтобы программа (а точнее — система управления базами данных) знала, какие именно записи следует удалять, мы используем такое вот хитрое выражение:

```
DataSet1.GotoBookmark(pointer(DBGrid1.SelectedRows.Items[i]));
```

Если вы не обидитесь, я не стану подробно обсуждать эту конструкцию. Рассматривайте ее как средство, которое дает нужный результат. Несколько позже в следующих главах мы рассмотрим методы набора данных `GetBookmark` и `GotoBookmark`.

Если же нет отмеченных записей, то удаляется текущая запись, как мы делали в предыдущей версии нашей программы.

> ***Замечание***
>
> По-хорошему следовало бы внести еще некоторые полезные изменения в нашу улучшенную версию программы, добавив в строку состояния (`StatusBar`) поле, в которое бы помещалось количество отмеченных записей, и написав обработчики некоторых событий (нажатие клавиш и щелчки мышью по сетке). Сделайте это сами. Кроме этого, следовало бы разместить на форме и контекстное меню, связав его с `DBGrid`. Однако наша учебная программа пока малофункциональна, и потребностей в таком меню нет. В следующей программе мы обязательно наведем порядок с этим делом.

Запустите программу на выполнение. Отметьте несколько строк. Для этого нужно, держа нажатой клавишу <Ctrl>, щелкать мышью по соответствующим строкам. Отмеченные строки будут выделены цветом, а индикатор будет содержать символ крупной точки. Если вы ошибочно отметили какую-то строку, то, продолжая держать нажатой клавишу <Ctrl>, щелкните по этой строке для снятия отметки. Чтобы снять все отметки, нужно, отпустив клавишу <Ctrl>, щелкнуть мышью по любой строке или нажать любую клавишу перемещения курсора: "стрелка вверх", "стрелка вниз", <Home> и т. д. или клавишу <Esc>.

Текст программы находится в каталоге Chapter06\DisplayDelete\FIBPlus.

6.1.2. Использование компонентов IBX

Мы можем выполнить те же самые действия и при использовании компонентов IBX.

> ***Предупреждение***
> Я в этих примерах использую стандартный набор компонентов IBX, поставляемый с Delphi 7. Существуют другие, улучшенные, версии этих компонентов, свойства и функциональность которых могут отличаться от этого стандарта.

Скопируйте ваш проект в другой каталог. Откройте его в Delphi. Удалите с формы компоненты FIBPlus и вместо них положите на форму следующие компоненты с вкладки **InterBase**: `Database` (присвоив компоненту имя `Database1`), `Transaction` (имя `ReadTransaction`; хотя никакая она, конечно, не read, а и read, и write. Конечно, в порыве вдохновения ее можно было бы назвать `Transaction1`, что, кстати, мы и сделаем в следующей программе) и компонент `DataSet` (имя `DataSet1`).

Вместо компонента `DataSet` можно было бы использовать компонент `Query`, однако его реализация в IBX настолько странная и загадочная, что я не рискну рекомендовать вам его использование.

> ***ВНИМАНИЕ!***
> При работе с реляционными базами данных никогда не используйте компонент `Table`. Это пережиток страшного "дореляционного" прошлого.

В отличие от компонентов FIBPlus здесь вам придется при программировании выполнить несколько больше действий. При небольшом количестве компонентов это, конечно, не так страшно. Хуже, когда вы пишете настоящую программу с большим количеством таблиц.

Если вы не создавали проект с компонентами FIBPlus (а зря), вы можете начать этот проект с нуля, выполнив все действия предыдущего раздела, за исключением размещения компонентов FIBPlus.

Для компонента `Database` установите свойства:

- `DatabaseName` в `D:\BestDatabase\work.fdb` (придется вводить вручную, компоненты IBX позволяют в диалоговом окне выбирать только файлы с расширением gdb);
- `DefaultTransaction` установите в `ReadTransaction` (здесь-то можно выбрать из списка);
- `LoginPrompt` установите в `False`, чтобы исключить вызов диалогового окна ввода имени пользователя и пароля при открытии базы данных.

Щелкните правой кнопкой мыши на компоненте базы данных и из появившегося контекстного меню выберите элемент **Database Editor** (Редактор базы

данных). Появится окно **Database Component Editor** (Редактор компонента базы данных) (рис. 6.5).

В поле **Database** уже будет присутствовать путь к файлу базы данных. Введите имя пользователя `WIZARD` (напоминаю — в любом регистре) и пароль `master`, соответственно, в поля **User Name** и **Password**, из выпадающего списка **Character Set** выберите набор символов **WIN1251**. Обратите внимание, как по мере ввода этих данных справа в списке **Settings** (Установки) будут появляться строки, необходимые для установления связи с базой данных. Эти строки по завершении установок будут помещены в список **Params**.

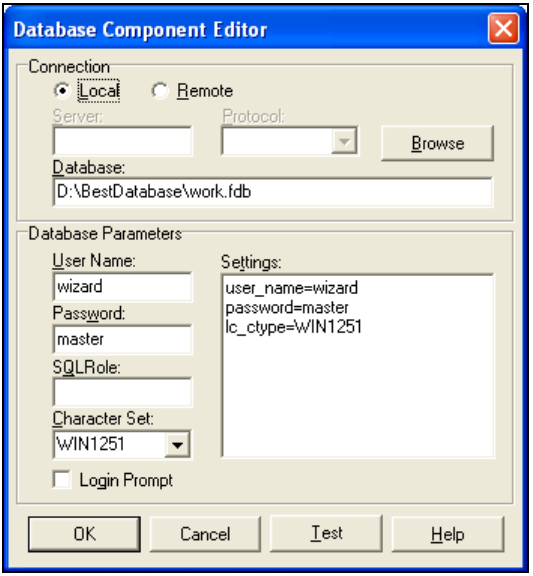


Рис. 6.5. Формирование параметров компонента базы данных

Вы можете проверить правильность введенных данных, щелкнув по кнопке **Test**. При этом будет выполнена попытка соединения с базой данных. Если вы допустили ошибки, то получите соответствующее диагностическое сообщение. Иначе система сообщит об успешности соединения.

Щелкните по кнопке **OK** для завершения установок соединения. В результате в список **Params** будут помещены строки:

```
user_name=wizard
password=master
lc_ctype=WIN1251
```

Для компонента транзакции установите свойства:

- `AutoStopAction` = `saCommit`;
- `DefaultDatabase` = `Database1`.

Для компонента DataSet основная задача — сгенерировать операторы SQL для выборки, добавления, изменения, удаления и обновления данных. Разумеется, это такие же операторы, которые мы получили и в FIBPlus. Правда, там нам удалось все это сделать с меньшими затратами сил. Приступим.

Свойство Database компонента DataSet установите в Database1.

Вначале отдельно от всех остальных операторов нужно сгенерировать оператор выборки данных.

Щелкните по кнопке [...] справа от свойства **SelectSQL**. Появится окно создания оператора SELECT (рис. 6.6). В левой части в списке **Tables** выберите имя нашей таблицы — REFCTR. Щелкните по кнопке **Add Table to SQL** (Добавить таблицу в оператор SQL). В правой части появится заготовка оператора. Слева внизу в списке полей таблицы **Fields** щелкните по символу *, а затем по кнопке **Add Field to SQL** (Добавить поле в оператор SQL). В результате будет создан нужный оператор SELECT, который выбирает все столбцы таблицы. Добавьте вручную к нему предложение ORDER BY NAME, чтобы, как и в предыдущей версии программы, упорядочить набор данных по краткому названию страны.

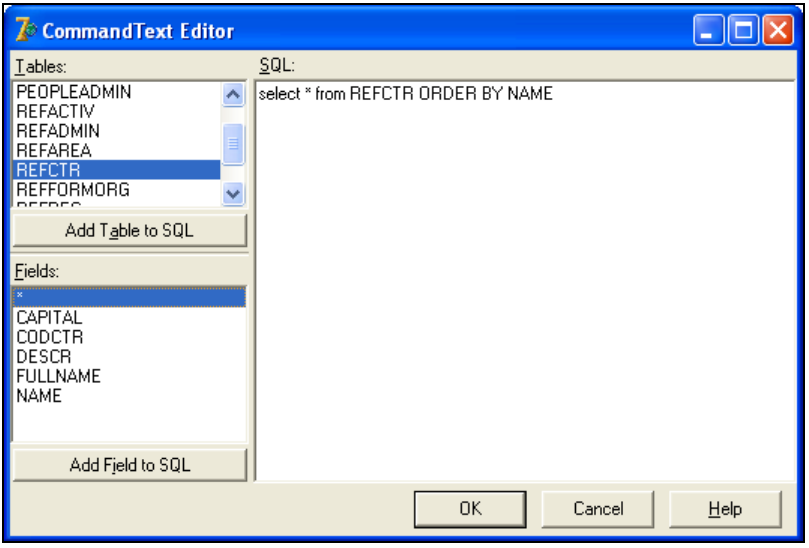


Рис. 6.6. Создание и редактирование оператора SELECT в IBX

Щелкните по кнопке **OK**, чтобы завершить создание оператора.

Теперь щелкните на компоненте DataSet правой кнопкой мыши и в контекстном меню выберите **Dataset Editor** (Редактор набора данных). Появится окно генерации остальных необходимых операторов SQL (рис. 6.7). Текущей

вкладкой будет **Options** (Режимы). Здесь мы также видим два списка — ключевых полей (**Key Fields**) и изменяемых полей (**Update Fields**). Немного странно, что среди ключевых полей мы наблюдаем и поле `DESCR`, которое имеет тип данных `BLOB`, что никак в состав ключа входить не может.

Щелкните по кнопке **Select Primary Keys** (Выбрать первичные ключи). По идее в результате должен стать отмеченным столбец первичного ключа нашей таблицы, `CODCTR`. У меня при этом ничего не произошло, вернее, были сняты все отметки ключевых полей. Похоже, IBX не видит первичного ключа. Не страшно, вручную отмечаем поле `CODCTR` (см. рис. 6.7).

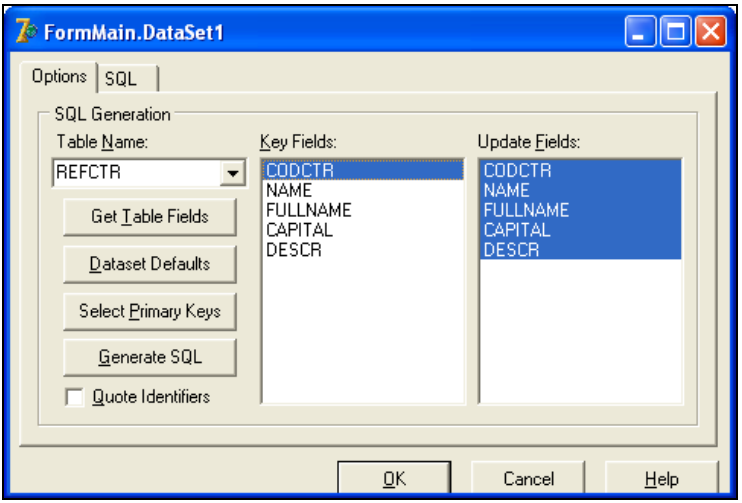


Рис. 6.7. Генерация операторов SQL в IBX (вы видите не дефекты картинки, а дефекты формы в IBX)

Щелкаем по кнопке **Generate SQL** (Сгенерировать операторы SQL). Текущей становится вкладка **SQL**, где мы можем посмотреть на сгенерированные операторы для изменения, добавления, удаления и обновления данных. Они такие же, как и в случае использования FIBPlus.

Щелкните по кнопке **OK**, чтобы завершить процесс создания необходимых операторов SQL.

> ***Замечание***
>
> Некоторая неправильность формы на рис. 6.7 не является полиграфическим дефектом. Так форма выглядит в реальной жизни.

Запустите программу на выполнение. Удалите какую-нибудь страну. Она пропадет из списка. Однако, если, не закрывая программу, запустить вторую ее копию или любую другую программу просмотра справочника стран, мы увидим, что страна не удалена. Если же вы попробуете в этой второй про-

грамме удалить ту же самую страну, то получите конфликт блокировки. Дело в том, что в первой программе не подтверждена транзакция, в контексте которой удалялась страна (в базах данных недопустимо одновременное изменение одной и той же записи различными процессами). Чего-то в нашей программе явно не хватает.

Закройте программу. Здесь необходимо, в отличие от FIBPlus (там мы установили свойство `AutoCommit = True`), явно подтвердить транзакцию после удаления строки таблицы. Добавьте в процедуру удаления записи одну строку — листинг 6.11.

Листинг 6.11. Процедура удаления текущей строки

```
procedure TFormMain.BDeleteClick(Sender: TObject);
begin
  if DataSet1.RecordCount = 0 then exit;
  if Application.MessageBox('Удалить текущую запись?',
     'Удаление', MB_YESNO + MB_ICONQUESTION) = mrYes then
    DataSet1.Delete;
  ReadTransaction.CommitRetaining;  // Строка подтверждения транзакции
                                    // для одной удаленной записи
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
end;
```

Мы добавили обращение к методу транзакции `CommitRetaining`. Этот метод подтверждает транзакцию — делает видимым для других процессов выполненные изменения. При этом сохраняется контекст транзакции: вам не нужно заново запускать транзакцию и переоткрывать набор данных.

В случае когда на сетке вы допускаете множественный выбор строк, процедура удаления будет выглядеть несколько иначе — листинг 6.12.

Листинг 6.12. Процедура удаления текущей строки или группы отмеченных строк

```
procedure TFormMain.BDeleteClick(Sender: TObject);
var i: Integer;
begin
  if DataSet1.RecordCount = 0 then exit;
  if DBGrid1.SelectedRows.Count > 1 then
  begin
    if Application.MessageBox(PAnsiChar('Удалить отмеченные записи: ' +
       IntToStr(DBGrid1.SelectedRows.Count) + '?'),
       'Удаление', MB_OKCANCEL + MB_ICONQUESTION) = mrOk then
    begin
```

```
      for i := 0 to DBGrid1.SelectedRows.Count – 1 do
      begin
        DataSet1.GotoBookmark(pointer(DBGrid1.SelectedRows.Items[i]));
        DataSet1.Delete;
      end;
    end;
  end
  else
  begin
    if Application.MessageBox('Удалить текущую запись?',
       'Удаление', MB_YESNO + MB_ICONQUESTION) = mrYes then
      DataSet1.Delete;
  end;
  ReadTransaction.CommitRetaining;  // Строка подтверждения транзакции
                                    // сразу для всех удаленных записей
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
end;
```

Запустите программу на выполнение, убедитесь, что все работает нормально. Есть только один неприятный момент — здесь свойство `RecordCount` компонента `DataSet` содержит не общее количество строк в наборе данных, а только количество прочитанных строк на момент открытия набора данных (fetched rows). Если же у вас достаточно много записей, а на форме умещается только 26 строк, то вы получите количество записей 26.

Чтобы устранить такую досадную неувязку, необходимо после открытия набора данных вызвать его метод `FetchAll`, который считает (fetch) действительно все строки и установит правильное значение в `RecordCount`. Добавим следующую строку в обработчик события активации формы — листинг 6.13.

Листинг 6.13. Обработчик события активации формы

```
procedure TFormMain.FormActivate(Sender: TObject);
begin
  Database1.Connected := True;
  DataSet1.Active := True;
  DataSet1.FetchAll;   // Добавленная строка для чтения всех записей
                       // набора данных
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
  Application.OnHint := ShowHint;
end;
```

Теперь уж все! Проверьте, что программа работает правильно.

Текст программы находится в каталоге Chapter06\DisplayDelete\IBX.

Замечание

Если вы получили проект для IBX, используя созданный перед этим проект для FIBPlus, то по-хорошему следовало бы удалить из предложения **uses** в самом начале модуля главной (в этом случае — единственной) формы ссылки на модули, относящиеся к набору компонентов FIBPlus: `FIBDataSet`, `pFIBDataSet`, `FIBDatabase` и `pFIBDatabase`. В их присутствии ничего страшного нет, если вы не станете переносить исходный текст программы в среду Delphi, где не установлены эти компоненты. Тогда вы получите сообщения об ошибках.

При всем при этом на душе скребут кошки — мы написали программу, выполняющую просмотр данных и деструктивные действия с этими данными, можем только удалять с таким трудом созданные данные. Следовало бы включить в нашу программу и конструктивные моменты — добавление новых данных и изменение существующих.

Именно этим мы сейчас и займемся, создав программу, выполняющую основные полезные действия с нашим справочником стран.

6.2. Полнофункциональная программа работы со справочником стран

6.2.1. Использование компонентов FIBPlus

Сейчас мы внесем довольно серьезные изменения в нашу программу, включив возможности добавления новых данных, изменения существующих и введя возможность повторного открытия (обновления, refresh) нашего набора данных. Последнее может быть полезным, когда с нашей базой данных одновременно работает несколько клиентских программ. Обновление позволит получить актуальное состояние справочника стран в любой момент времени.

По причине лености, присущей нормальным программистам, следует скопировать все модули нашего проекта DisplayDelete в другой каталог, открыть в Delphi этот проект и сохранить его в том же каталоге с другим именем — InsertUpdate (меню **File | Save Project As**). Удалите из этого каталога все модули с именем DisplayDelete и с любыми расширениями (cfg, dof, dpr, exe, res).

При желании совершать подвиги или набить руку, вы можете создать проект на пустом месте и добавить функциональность предыдущей программы.

Сохраним существующую функциональность предыдущей программы и добавим новую. Вначале внесем изменения в главное меню, включив элементы для добавления, изменения и обновления.

Дважды щелкнем мышью по элементу главного меню. Добавим новые элементы в меню, выделяя элемент меню, *перед* которым нужно вставить новый элемент, и нажимая клавишу <Ins>. Создадим следующие элементы:

- **Добавить**, имя `MAdd`, `ShortCut` = `Shift+Ins`, `Hint` = `Добавить новую запись`;
- **Изменить**, имя `MEdit`, `ShortCut` = `Enter`, `Hint` = `Изменить текущую запись`;
- **Обновить**, имя `MRefresh`, `ShortCut` = `F5`, `Hint` = `Обновить список`.

По ходу дела мы добавим немного красоты нашей программе.

Поместим на форму с вкладки **Win32** компонент `ImageList` и заполним его нужными иконками. Дважды щелкните мышью по компоненту. Появится окно (рис. 6.8), в которое можно добавлять иконки (рисунки формата BMP). Щелкните по кнопке **Add** и в появившемся окне **Add Images** поочередно добавьте рисунки для элементов меню из каталога Icons: AddM, EditM, RefreshM, DelM и ExitM. Поскольку список будет использоваться для элементов меню, размер всех рисунков не должен превышать 16×16 пикселов. У меня все рисунки, заканчивающиеся буквой "M" (то есть, для меню), имеют соответствующий размер.

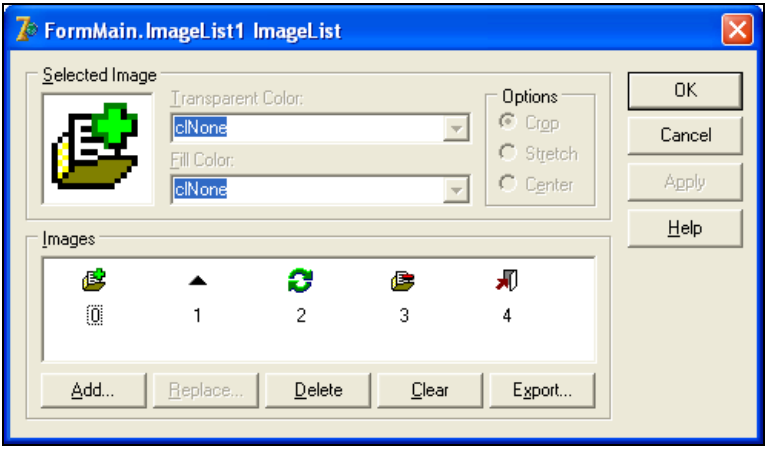


Рис. 6.8. Формирование списка иконок в компоненте `ImageList`

В компоненте главного меню установим свойство `Images` в `ImageList1` (выберем из выпадающего списка, содержащего ровно один элемент). Для каждого элемента меню, кроме элемента самого верхнего уровня **Файл**, установим свойство `ImageIndex` в соответствующее значение, чтобы каждый элемент содержал иконку, аналогичную тем, что будут располагаться на кнопках быстрого доступа. Нет необходимости вводить в свойство `ImageIndex` каж-

дый раз число — нужный рисунок можно выбрать из выпадающего списка, который будет содержать как номера рисунков в компоненте `ImageList1`, так и сами рисунки.

Посмотрите, как все же это удобно. Если вам нужно отменить использование рисунка в каком-либо элементе, то в свойстве `ImageIndex` нужно ввести `-1`.

В процессе проектирования меню будет выглядеть следующим образом — рис. 6.9.

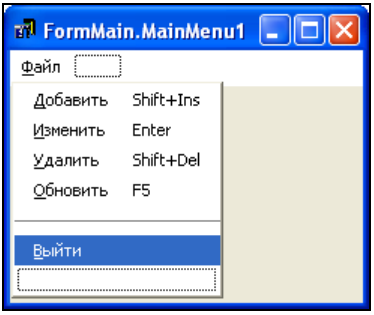


Рис. 6.9. Главное меню полнофункциональной программы с FIBPlus в режиме проектирования

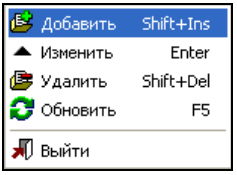


Рис. 6.10. Раскрытое меню

Однако во время выполнения программы раскрывшееся меню будет содержать и иконки — рис. 6.10.

Создадим, наконец, и контекстное меню, в которое аналогичным образом поместим только две строки, присутствующие в главном меню — изменить и удалить. Присвоим им имена `PEdit` и `PDelete`. Мы сделаем так, что контекстное меню будет всплывать, когда пользователь на сетке отображения строк набора данных щелкнет правой кнопкой мыши. В этот момент текущей будет ровно одна строка нашей таблицы. По этой причине контекстное меню будет содержать только те элементы, которые относятся к отдельной записи, изменение и удаление. Свяжем контекстное меню с сеткой — в `DBGrid` установим значение свойства `PopupMenu` в `PopupMenu1` (выбираем из выпадающего списка, содержащего только один элемент).

Для появления иконок в соответствующих элементах контекстного меню установим его свойство `Images` в `ImageList1`, а свойствам `ImageIndex` элементов зададим нужные номера (также выберем из выпадающего списка).

На панели инструментов расположим соответствующие кнопки. С вашего позволения я уберу кнопки `Button` и заменю их на более приличный вариант — `SpeedButton`. Они позволяют хранить иконки, что, несомненно, улучшит внешний вид нашей формы. Эти кнопки могут содержать и тексты, однако по сложившейся традиции на кнопках в панели быстрого доступа, как

правило, помещаются только рисунки, но не надписи. Необходимые тексты могут появляться в виде подсказок, когда пользователь наводит на кнопку курсор мыши. Это поведение мы реализуем в нашей программе — таким же образом, как мы сделали в программе просмотра и удаления записей справочника стран.

Зададим кнопкам мнемонические ("человеческие") имена, внесем нужные тексты в свойства `Hint`, установим размер 26×26 пикселов (свойства `Height` и `Width`), загрузим иконки, используя свойство `Glyph`.

Выделив мышью нужную кнопку, щелкните в Инспекторе объектов справа от свойства `Glyph` по кнопке [...]. Появится окно **Picture Editor** (рис. 6.11).

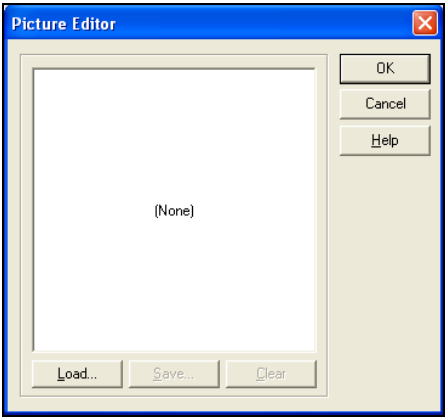


Рис. 6.11. Окно **Picture Editor** для добавления к кнопке иконки

Щелкните по кнопке **Load** (Загрузить) и в диалоговом окне **Load Picture** из каталога Icons загрузите нужный рисунок. Имена рисунков: Add, Edit, Del, Refresh и Exit. При нашем размере кнопок рисунки не должны превышать 24×24 пикселов.

Нужно выполнить еще одно важное действие — внести маленькое изменение в свойство `Options` набора данных. Установите в `True` значение подсвойства `poKeepSorting`. Это приведет к тому, что вновь добавляемые записи будут размещаться в списке в соответствии с заданным порядком сортировки.

Теперь общий вид проекта будет приблизительно следующим — рис. 6.12.

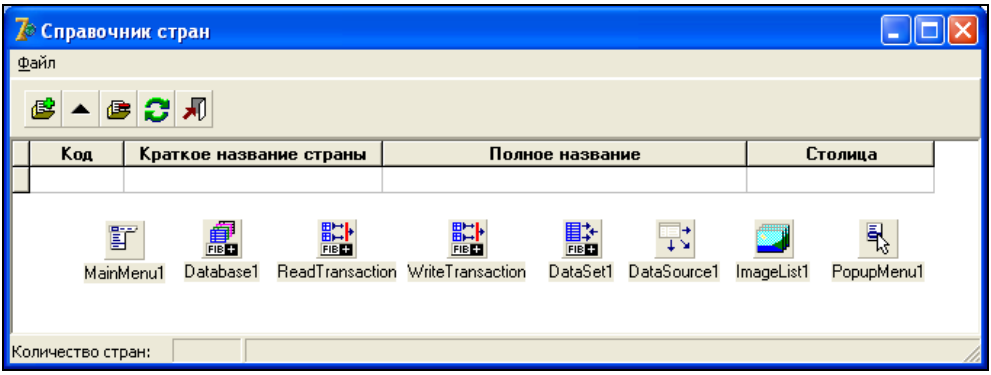


Рис. 6.12. Общий вид главной формы полнофункционального проекта

Пока это только украшения (но согласитесь, что красота — великая сила). Теперь займемся функциональной частью от простого к сложному.

Самое простое — реализация обновления списка. Напишите следующий обработчик обновления списка для элемента меню и свяжите с ним обычным образом щелчок по кнопке обновления (листинг 6.14).

Листинг 6.14. Обновление списка

```
procedure TFormMain.MRefreshClick(Sender: TObject);
begin
  DataSet1.FullRefresh;
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
end;
```

Метод `FullRefresh` набора данных позволяет заново считать с сервера весь набор данных. Если с момента последнего чтения изменилось количество записей, благодаря работе параллельных процессов, то мы опять размещаем в соответствующем поле строки состояния новое значение. В дальнейшем при исследовании свойств транзакций мы постоянно будем моделировать на локальном компьютере работу в архитектуре "клиент-сервер", с некоторыми ограничениями, разумеется. При этом в каждой программе нам понадобится использовать метод `FullRefresh`. По сложившейся в Windows традиции клавишей быстрого доступа для обновления у нас будет <F5>.

Теперь займемся добавлением и редактированием данных. Если помните, при создании программы работы с учетными записями пользователя мы использовали один модуль и для добавления, и для изменения данных. Сейчас мы поступим более правильно и методически более грамотно и создадим две формы — одну для добавления новой страны, другую для изменения существующей страны.

Создайте новую форму: меню **File | New | Form**. Установите для нее следующие свойства:

- `Caption` = `Добавление страны`;
- `Name` = `FormAddCtr`.

Сохраните модуль с именем AddCtr.

Поместите на форму панель, убрав свойство `Caption` и выровняв ее по верхнему краю. На панель поместите нужные метки и поля редактирования для ввода данных по стране. Положите еще одну панель и выровняйте ее по нижнему краю. На этой панели разместите кнопки (на этот раз опять `TButton`) **OK** и **Отменить**, установив для них соответствующие свойства. Для **OK** задайте свойство `Default` = `True` (будет моделироваться щелчок по кнопке, когда пользователь нажмет клавишу <Enter>). Для кнопки **Отменить** задайте `Cancel` = `True` (будет моделироваться щелчок по кнопке, когда пользователь нажмет клавишу <Esc>), `ModalResult` = `mrCancel` (при щелчке по кнопке будет завершена работа этого модуля, а вызывающая программа получит код возврата `mrCancel`).

Заметьте, что для кнопки **OK** мы не задали значения свойства `ModalResult` — оставили значение по умолчанию `mrNone`. В обработчике события щелчка по этой кнопке мы принимаем решение о завершении работы модуля. В противном случае (при любом другом значении `ModalResult`) управление всегда бы передавалось вызвавшему модулю.

Наконец, на центральную часть формы с вкладки **Win32** поместите компонент `RichEdit` и задайте выравнивание по всей форме. Здесь будет располагаться описание страны — любой форматированный текст.

Теперь форма добавления новой записи должна иметь такой вид — рис. 6.13.

Следует связать эту форму с главной формой — вызываем меню **File | Use Unit** и из списка выбираем модуль **Main**. Аналогичным образом нужно связать главную форму с формой `FormAddCtr`, перейдя в главную форму и выбрав в списке используемых модулей имя модуля **AddCtr**. Подобную связку можно выполнить и вручную, просто добавив имя нужного модуля в предложение **`uses`** в начальной части связываемой формы.

Для начала напишите обработчик события `OnShow` для формы добавления страны (листинг 6.15).

Листинг 6.15. Обработчик события `OnShow` для формы добавления страны

```
procedure TFormAddCtr.FormShow(Sender: TObject);
begin
  CODCTR.Text := '';
```

```
  NAME1.Text := '';
  FULLNAME.Text := '';
  CAPITAL.Text := '';
  RichEdit1.Clear;
  CODCTR.SetFocus;
end;
```

Здесь устанавливаются пустые начальные значения полей создаваемой записи.

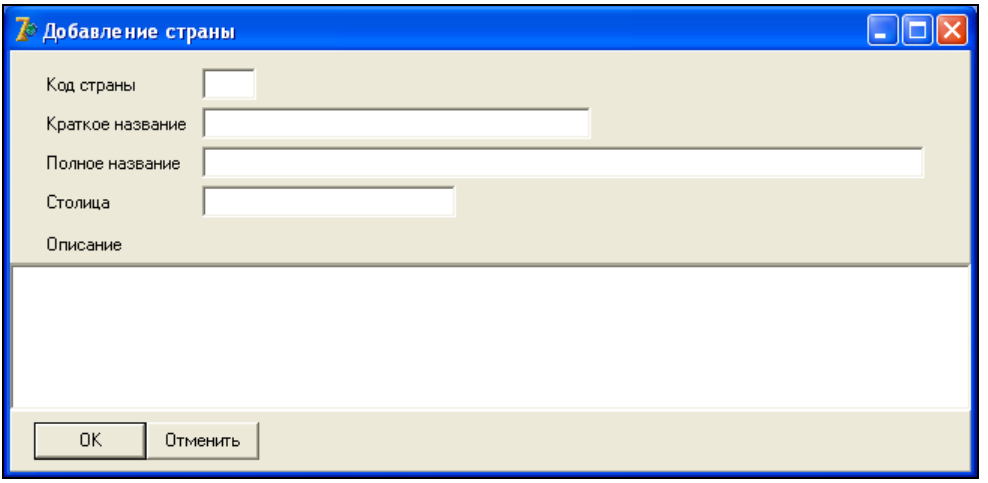


Рис. 6.13. Форма для добавления страны

> ***Замечание***
>
> Полям `Edit` и `RichEdit` мы присвоили те же имена, что и у столбцов нашей таблицы. Кроме поля `Name`. Здесь мы задали `NAME1`. Дело в том, что сама форма имеет свойство `Name`, и для устранения двусмысленности мы внесли это изменение. Хотя, надо сказать, и в том варианте все почему-то работает правильно.

Далее принимаем решение, что в записи страны должны обязательно присутствовать по меньшей мере код и краткое название страны. Возможность выполнения добавления новой строки будем осуществлять, устанавливая доступность кнопки **OK**. Напишите один обработчик события изменения данных в полях кода и названия страны (`OnChange`) — листинг 6.16.

> ***Замечание***
>
> Напомню, что для этого нужно вначале выделить компонент `Edit` для ввода кода и написать для него обработчик события `OnChange`, как показано в листинге 6.16, а затем обычным образом связать с этим обработчиком событие `OnChange` у компонента `Edit`, предназначенного для ввода названия страны.

Листинг 6.16. Обработчик события изменения данных в коде и названии страны

```
procedure TFormAddCtr.CODCTRChange(Sender: TObject);
begin
  BOK.Enabled := (Trim(CODCTR.Text) <> '') and (Trim(NAME1.Text) <> '');
end;
```

Мы используем функцию `Trim`, которая убирает начальные и конечные пробелы. Кнопка **OK** будет доступной, если и код, и название страны являются непустыми и содержат непробельные символы.

Основное событие — щелчок мышью по кнопке **OK** (листинг 6.17).

Листинг 6.17. Добавление в таблицу новой страны

```
procedure TFormAddCtr.BOKClick(Sender: TObject);
var  SQL: String;
     Stream: TStream;
begin
  CODCTR.Text := Trim(CODCTR.Text);
  SQL := 'SELECT COUNT(CODCTR) FROM REFCTR WHERE CODCTR = ''' +
    CODCTR.Text + '''';
  if (FormMain.Database1.QueryValueAsStr(SQL, 0) <> '0') then
  begin
    Application.MessageBox(
     'Страна с тем же кодом присутствует в базе',
     'Дублирование', MB_OK + MB_ICONSTOP);
    CODCTR.SetFocus;
    exit;
  end;
  SQL := 'SELECT COUNT(CODCTR) FROM REFCTR WHERE NAME = ''' +
    NAME1.Text + '''';
  if (FormMain.Database1.QueryValueAsStr(SQL, 0) <> '0') then
  begin
    Application.MessageBox(
     'Страна с тем же названием присутствует в базе',
     'Дублирование', MB_OK + MB_ICONSTOP);
    NAME1.SetFocus;
    exit;
  end;
  FormMain.DataSet1.Insert;
  FormMain.DataSet1.FieldByName('NAME').AsString := Trim(NAME1.Text);
  FormMain.DataSet1.FieldByName('CODCTR').AsString := CODCTR.Text;
```

```
  FormMain.DataSet1.FieldByName('FULLNAME').AsString := FULLNAME.Text;
  FormMain.DataSet1.FieldByName('CAPITAL').AsString := CAPITAL.Text;
  Stream := TMemoryStream.Create;
  try
    RichEdit1.Lines.SaveToStream(Stream);
    Stream.Position := 0;
    TFIBBlobField(FormMain.DataSet1.FieldByName('DESCR')).LoadFromStream(
      Stream);
  finally
    Stream.Free;
  end;
  FormMain.DataSet1.Post;
  ModalResult := mrOk;
end;
```

Чтобы все это заработало, необходимо в предложении **uses** данной формы явно указать модуль FIBDataSet. Предложение **uses** будет иметь приблизительно следующий вид:

```
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ComCtrls, ExtCtrls, FIBDataSet;
```

Замечу, что если бы мы поместили на эту форму один из компонентов FIBPlus, модуль FIBDataSet был бы добавлен в предложение **uses** автоматически.

В обработчике щелчка по кнопке **OK** мы выполняем проверку дублирования в базе данных записей с тем же кодом (это первичный ключ, и дубликат не может быть помещен в базу данных). Кроме того, мы решили не допускать дублирования и кратких названий стран. Для этих целей мы используем метод базы данных `QueryValueAsStr`, который выполняет заданный оператор SQL и возвращает одно значение. Оператор SQL имеет вид:

```
SQL := 'SELECT COUNT(CODCTR) FROM REFCTR WHERE CODCTR = ''' +
    CODCTR.Text + '''';
```

В этом операторе `SELECT` мы подсчитываем количество записей с тем же значением первичного ключа. Если в базе данных уже есть подобная запись, то выдается сообщение об ошибке, и фокус ввода устанавливается на поле кода страны.

Аналогичным образом выполняется проверка на дублирование краткого названия страны.

> ***Замечание***
>
> Независимо от нашей проверки, если мы попытаемся поместить в базу запись с тем же значением первичного ключа, что уже присутствует в базе, мы получим исключение. Обработка исключений в программе — особое искусство, некоторые приемы которого мы рассмотрим в одной из следующих глав.
>
> Аналогично, мы могли бы в базе данных описать столбец краткого названия страны с атрибутом `UNIQUE` или создать на основании этого столбца уникальный индекс, что также приводило бы к исключению при попытке поместить в базу дубликат.

Для запуска процесса помещения в базу данных новой строки используется метод набора данных `Insert`. Далее полям присваиваются значения с помощью метода `FieldByName`. Собственно отправка на сервер новой записи осуществляется с помощью метода `Post`.

Особой заботы требует поле `BLOB`, для которого в программе мы используем компонент `RichEdit`. Чтобы поместить содержимое `RichEdit` в поле `BLOB`, мы используем поток:

```
Stream := TMemoryStream.Create;
try
  RichEdit1.Lines.SaveToStream(Stream);
  Stream.Position := 0;
  TFIBBlobField(FormMain.DataSet1.FieldByName('DESCR')).LoadFromStream(
    Stream);
finally
  Stream.Free;
end;
```

Вначале мы создаем поток оператором `Create`, затем сохраняем в этом потоке содержимое поля `RichEdit` методом `SaveToStream`. Устанавливаем текущую позицию потока в `0`, чтобы помещать в `BLOB` весь текст с самого начала, и перемещаем текст с помощью метода `LoadFromStream` из потока в столбец `BLOB`. Всю эту деятельность мы по правилам хорошего тона заключаем в блок **`try`**, чтобы даже в случае возникших ошибок освободить память, выделенную под поток. Освобождение памяти осуществляется в блоке **`finally`**, который будет выполнен в любом случае, независимо от наличия или отсутствия ошибок в процессе перемещения текста. Выделяемую динамически память нужно всегда освобождать, иначе происходит перерасход ресурсов операционной системы, что может сильно ухудшить производительность.

Вернитесь в главный модуль и напишите обработчик выбора элемента меню **Добавить** (листинг 6.18).

Листинг 6.18. Процедура добавления страны

```
procedure TFormMain.MAddClick(Sender: TObject);
begin
  FormAddCtr.ShowModal;
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
end;
```

Свяжите с этим обработчиком щелчок мышью по кнопке **Добавить**.

Запустите программу на выполнение. Добавьте несколько новых записей. Заметьте, что новые записи располагаются в нужном месте на сетке в соответствии с заданным порядком сортировки. Это происходит потому, что мы для набора данных установили в `True` значение подсвойства `poKeepSorting`.

Добавим в программу функцию редактирования существующих данных. Я научу вас еще одному трюку, который может согреть душу особо ленивым программистам (к которым я, естественно, причисляю и себя).

У нас есть форма с подходящим внешним видом и близкой функциональностью — добавление данных. Скопируйте файлы AddCtr.dfm и AddCtr.pas в какой-нибудь другой каталог. В среде Delphi откройте созданную копию этого модуля. В Инспекторе объектов измените название формы с `FormAddCtr` на `FormEditCtr` и сохраните (**File | Save As**) в каталоге вашего проекта с именем EditCtr. Добавьте этот модуль в проект — меню **Project | Add to Project**. Теперь можно вносить в модуль нужные изменения. Кстати, если вы не поменяете имя формы, то не сможете добавить этот модуль в проект. Система выдаст сообщение о существовании в проекте формы с тем же именем.

Поменяйте заголовок формы (`Caption`) на `Изменение страны`. В раздел **`private`** формы добавьте две строки:

```
OLD_CODCTR: String;
OLD_NAME: String;
```

В эти переменные мы будем помещать старые значения кода и краткого названия страны (значения до изменения их пользователем).

Обработчик события формы `OnShow` перепишем в следующем виде — листинг 6.19.

Листинг 6.19. Обработка события `OnShow` для формы редактирования страны

```
procedure TFormEditCtr.FormShow(Sender: TObject);
begin
  CODCTR.Text := FormMain.DataSet1.FieldByName('CODCTR').AsString;
  NAME1.Text := FormMain.DataSet1.FieldByName('NAME').AsString;
```

```
  FULLNAME.Text := FormMain.DataSet1.FieldByName('FULLNAME').AsString;
  CAPITAL.Text := FormMain.DataSet1.FieldByName('CAPITAL').AsString;
  RichEdit1.Text := FormMain.DataSet1.FieldByName('DESCR').AsString;
  OLD_CODCTR := CODCTR.Text;
  OLD_NAME := NAME1.Text;
  CODCTR.SetFocus;
end;
```

Здесь мы помещаем в поля редактирования значения из текущей записи страны и запоминаем старые значения кода и краткого названия страны. Для помещения данных из поля `BLOB` мы используем свойство `Text` компонента `RichEdit`. Конечно, здесь мы также могли бы использовать и поток (чего не избежать при помещении этого текста обратно в столбец таблицы `BLOB`), однако более простой вариант добавления кажется более удобным и естественным.

Обработчик щелчка по кнопке **OK** — листинг 6.20.

Листинг 6.20. Процедура редактирования страны

```
procedure TFormEditCtr.BOKClick(Sender: TObject);
var  SQL: String;
     Stream: TStream;
begin
  CODCTR.Text := Trim(CODCTR.Text);
  NAME1.Text := Trim(NAME1.Text);
  if CODCTR.Text <> OLD_CODCTR then
  begin
    SQL := 'SELECT COUNT(CODCTR) FROM REFCTR WHERE CODCTR = ''' +
      CODCTR.Text + '''';
    if (FormMain.Database1.QueryValueAsStr(SQL, 0) <> '0') then
    begin
      Application.MessageBox(
       'Страна с тем же кодом присутствует в базе',
       'Дублирование', MB_OK + MB_ICONSTOP);
      CODCTR.SetFocus;
      exit;
    end;
  end;
  if NAME1.Text <> OLD_NAME then
  begin
    SQL := 'SELECT COUNT(CODCTR) FROM REFCTR WHERE NAME = ''' +
      NAME1.Text + '''';
```

```
    if (FormMain.Database1.QueryValueAsStr(SQL, 0) <> '0') then
    begin
      Application.MessageBox(
       'Страна с тем же названием присутствует в базе',
       'Дублирование', MB_OK + MB_ICONSTOP);
      NAME1.SetFocus;
      exit;
    end;
  end;
  FormMain.DataSet1.Edit;
  FormMain.DataSet1.FieldByName('NAME').AsString := Trim(NAME1.Text);
  FormMain.DataSet1.FieldByName('CODCTR').AsString := CODCTR.Text;
  FormMain.DataSet1.FieldByName('FULLNAME').AsString := FULLNAME.Text;
  FormMain.DataSet1.FieldByName('CAPITAL').AsString := CAPITAL.Text;
  Stream := TMemoryStream.Create;
  try
    RichEdit1.Lines.SaveToStream(Stream);
    Stream.Position := 0;
    TFIBBlobField(FormMain.DataSet1.FieldByName('DESCR')).LoadFromStream(
       Stream);
  finally
    Stream.Free;
  end;
  FormMain.DataSet1.Post;
  ModalResult := mrOk;
end;
```

Выполняемые здесь действия похожи на те, что были при добавлении новой страны. Для внесения изменений в набор данных используется метод `Edit`. Кроме того, прежде чем выполнять проверки на дублирование кода и названия, мы проверяем, изменялись ли эти значения.

Вернитесь в главный модуль, свяжите его с модулем редактирования записи EditCtr (меню **File | Use Unit**) и напишите обработчик события выбора в главном меню элемента **Изменить** (листинг 6.21).

Листинг 6.21. Редактирование страны

```
procedure TFormMain.MEditClick(Sender: TObject);
begin
  if DataSet1.RecordCount = 0 then exit;
  FormEditCtr.ShowModal;
end;
```

Здесь, как и в случае удаления текущей записи, вначале проверяется наличие записей в наборе данных. Иначе при пустом списке мы можем получить ненужное нам исключение.

Свяжите с этим обработчиком выбор в контекстном меню элемента **Изменить**, щелчок по кнопке быстрого доступа **Изменить**, а также двойной щелчок по сетке `DBGrid` (событие `OnDblClick`). Посмотрите, сколько у нас вариантов обращения к редактированию текущей записи — выбор в главном или контекстном меню элемента **Изменить**, щелчок по кнопке **Изменить**, нажатие клавиши <Enter> (клавиша быстрого доступа, заданная в главном меню), двойной щелчок мышью по сетке, использование клавиши <Alt> и букв для доступа к нужному элементу главного меню. Наш пользователь может выбрать любой удобный для него вариант, к которому он привык. Похоже, нам придется просто купаться в волнах любви и благодарности, которые будут исходить от наших пользователей.

Текст программы находится в каталоге Chapter06\InsertUpdate\FIBPlus.

Несколько слов о *BLOB* и *RichEdit*

Как мы уже говорили, столбцы `BLOB` могут хранить что угодно — форматированный текст, рисунки, звук, видео. Для работы с этими столбцами как с текстовыми мы обычно используем компоненты `RichEdit` или `DBRichEdit`. Эти компоненты могут хранить форматированный текст в формате RTF (Rich Text Format — обогащенный текстовый формат), однако не позволяют сохранять графику, которая допустима в формате RTF.

В настоящей версии созданной нами программы никаких средств редактирования текста в программе не предусматривается. Чтобы поместить форматированный текст в базу данных, нужно в любом хорошем редакторе (например, Word или WordPad) создать необходимый текст и перенести его через Буфер обмена в наш компонент `RichEdit` при добавлении или изменении записи страны.

Разумеется, мы с вами в состоянии написать и собственный редактор. Если у нас будет время и останутся силы, мы это сделаем несколько позже.

6.2.2. Использование компонентов IBX

Скопируйте проект в другой каталог. Откройте его в среде Delphi. Удалите компоненты FIBPlus. С вкладки **InterBase** поместите на форму компоненты базы данных, транзакции и набора данных. Установите для них соответствующие значения свойств.

Еще один полезный трюк. Можно не тратить сейчас время на размещение на форме этих компонентов и установление их свойств. В Delphi откройте модуль Main из проекта DisplayDelete для IBX, нажмите клавишу <Shift> и, не отпуская ее, отметьте мышью все три компонента IBX. Скопируйте отмеченные компоненты в Буфер обмена, выбрав в меню **Edit | Copy** или нажав клавиши <Ctrl>+<C>. Перейдите к главной форме создаваемой программы и вставьте компоненты, выбрав в меню **Edit | Paste** или нажав клавиши <Ctrl>+<V>.

Далее нужно выполнить небольшие корректировки исходного текста с учетом отличий IBX от FIBPlus.

Имеет смысл из всех модулей в предложении `uses` удалить ссылки на модули компонентов FIBPlus. В начальной части этих имен присутствуют буквы FIB.

При активации формы после открытия набора данных нужно добавить строку:

```
DataSet1.FetchAll;
```

чтобы были считаны все записи набора данных.

В обработчике обновления списка хотелось бы использовать не метод `FullRefresh`, отсутствующий в IBX, а `Refresh`, который, казалось бы, выполняет те же функции, однако он не дает нужных результатов (по крайней мере, в стандартном наборе компонентов, поставляемых с Delphi 7). Поэтому приходится переоткрывать набор данных, предварительно запомнив ключ текущей записи, а затем выполнив установку на эту запись. Теперь этот обработчик выглядит так — листинг 6.22.

Листинг 6.22. Обновление списка

```
procedure TFormMain.MRefreshClick(Sender: TObject);
var Cod: String;
begin
  if DataSet1.RecordCount = 0 then exit;
  Cod := DataSet1.FieldByName('CODCTR').AsString;
  DataSet1.Close;
  DataSet1.Open;
  DataSet1.Locate('CODCTR', Cod, [loPartialKey]);
  StatusBar1.Panels[1].Text := IntToStr(DataSet1.RecordCount);
end;
```

Вначале в переменной `Cod`, которая является строкой, запоминается код текущей записи, затем закрывается и снова открывается набор данных. После этого запись с запомненным ключом делается текущей при вызове метода `Locate`.

После добавления новой записи нужно обратиться к этому обработчику, включив после вызова модуля добавления строку:

```
MRefreshClick(nil);
```

Более серьезные изменения нужны для форм добавления и редактирования. Откройте модуль добавления. Поскольку в IBX у компонента базы данных не существует метода для выполнения запроса к базе данных, который мы использовали в предыдущей программе для проверки дублирования кода и краткого названия, то следует на эту форму поместить компонент `Query` и установить только одно свойство `Database` = `FormMain.Database1`.

Проверка дублирования кода страны выполняется следующим образом — листинг 6.23.

Листинг 6.23. Проверка дублирования кода страны в компонентах IBX

```
SQL := 'SELECT CODCTR FROM REFCTR WHERE CODCTR = ''' +
    CODCTR.Text + '''';
Query1.Close;
Query1.SQL.Text := SQL;
Query1.Open;
if Query1.RecordCount > 0 then
...
```

Аналогично проверяется и дублирование названия страны. Нужно не забыть после отправки новой записи на сервер (метод `Post`) выполнить подтверждение транзакции:

```
FormMain.Transaction1.CommitRetaining;
```

В модулях добавления новой страны и редактирования существующей при работе с потоками заменяем строку

```
TFIBBlobField(FormMain.DataSet1.FieldByName('DESCR')).LoadFromStream(
   Stream);
```

на

```
TBlobField(FormMain.DataSet1.FieldByName('DESCR')).LoadFromStream(
   Stream);
```

Остальные строки кода похожи на добавление записи с использованием FIBPlus.

Редактирование записи выполняется аналогичным образом. Также помещаем на форму редактирования компонент `Query`, который используем для проверки дублирования.

IBX возвращает код страны, добавив при необходимости справа пробелы до размера поля в три байта. Это происходит для столбцов, имеющих тип данных `CHAR`. Для `VARCHAR` этого не происходит. По этой причине при помещении в строку редактирования кода нужно убрать пробелы с помощью функции `Trim`. Иначе даже при отсутствии изменений кода программа будет считать, что код изменялся.

Текст программы находится в каталоге Chapter06\InsertUpdate\IBX.

6.3. Программа работы со справочниками стран и регионов

6.3.1. Использование компонентов FIBPlus

Между справочником стран и справочником регионов существует иерархическая связь (отношение "один-ко-многим") — в одной стране существует некоторое количество регионов, может быть и нулевое. Такая связь в программировании часто называется связь "главная-подчиненная" или master-detail. В этом случае пользователю удобно было бы видеть на экране одновременно список стран и список регионов текущей страны.

Создайте в Delphi новый проект, разместите на форме компоненты, показанные на рис. 6.14.

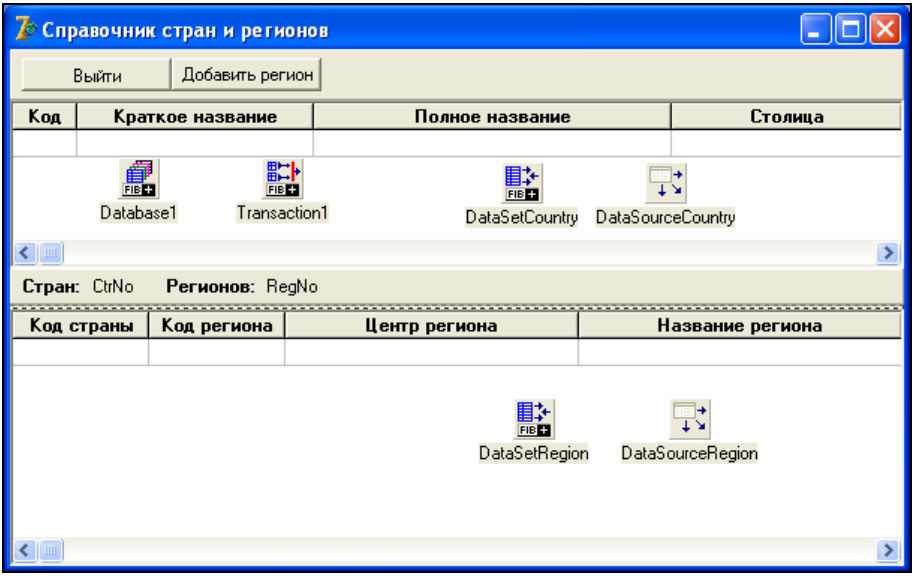


Рис. 6.14. Программа просмотра стран и регионов

Обратите внимание, что в этой программе мы, наконец, начали присваивать осмысленные имена компонентам, имеющим отношение к базе данных. Это касается наборов данных, `DataSource` и `DBGrid`. Конечно, это следует делать во всех программах, в особенности если в них содержится большое количество наборов данных.

Нижнюю сетку (`DBGridRegion`), отображающую регионы текущей страны, нужно выровнять по нижнему краю (`Align` = `alBottom`), с вкладки **Additional** поместить на форму компонент `Splitter` (разделитель) и также выровнять по нижнему краю. После этого на форме размещается панель, которая будет указывать количество стран и количество регионов текущей страны, она выравнивается по нижнему краю. На эту панель положите четыре метки `Label`, как показано на рис. 6.14. Другая, инструментальная, панель выравнивается по верхнему краю; на нее кладется кнопка завершения работы программы. Верхнюю сетку (`DBGridCountry`) выравниваем по всей клиентской области (`Align` = `alClient`). При выполнении программы разделитель будет использоваться для изменения размеров сеток, отображающих страны и регионы.

Здесь мы вместо строки состояния (`StatusBar`) используем обычную панель, на которой размещаем метки — просто для разнообразия. Однако надо признать, что эта практика не является хорошей. Вам следует придерживаться определенного единого стандарта при размещении элементов управления и информационных элементов на формах ваших программ. Если только ваш заказчик не потребует чего-то иного.

Характеристики всех компонентов устанавливаются обычным образом. Кроме компонента `DataSetRegion`.

Прежде всего, этот компонент нужно связать с родительским (`DataSetCountry`), установив значение его свойства `DataSource` в `DataSourceCountry`. После этого раскройте свойство `DetailConditions` и установите в `True` его подсвойства: `dcForceOpen` (чтобы автоматически выполнялось открытие этого набора данных при изменении текущей записи родительского набора данных) и `dcWaitEndMasterScroll` (чтобы выполнялась задержка открытия набора данных на указанное время; время задержки мы зададим динамически, в процессе выполнения программы). Наличие такой задержки позволит не переоткрывать раньше времени подчиненный набор данных, в то время, когда пользователь просто прокручивает список в поисках нужной ему записи. Когда пользователь остановился, найдя нужную строку, тут и происходит открытие списка регионов этой страны. При работе в сети это позволит несколько сэкономить сетевой трафик.

Следует внести изменения в сгенерированные операторы добавления и изменения в двух свойствах (листинги 6.24 и 6.25).

Листинг 6.24. Оператор добавления новой строки (записан в свойстве `InsertSQL`)

```
INSERT INTO REFREG(
    CODCTR,
    CODREG,
    NAMEREG,
    CENTER
)
VALUES(
    :MAS_CODCTR,  /* Изменение */
    :CODREG,
    :NAMEREG,
    :CENTER
)
```

Листинг 6.25. Оператор изменения текущей строки (записан в свойстве `UpdateSQL`)

```
UPDATE REFREG
SET
    CODCTR = :MAS_CODCTR,  /* Изменение */
    CODREG = :CODREG,
    NAMEREG = :NAMEREG,
    CENTER = :CENTER
WHERE
    CODCTR = :OLD_CODCTR
    and CODREG = :OLD_CODREG
```

Здесь префикс `MAS_` означает ссылку на поле родительской (основной, родительской, главной, master) таблицы.

В обработчике события отображения формы нужно написать код из листинга 6.26.

Листинг 6.26. Обработчик события отображения формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin
  Database1.Connected := True;
  DataSetCountry.Open;
  CtrNo.Caption := IntToStr(DataSetCountry.RecordCount);
```

```
  DataSetRegion.WaitEndMasterInterval := 500;
  DBGridCountry.SetFocus;
end;
```

Помимо соединения с базой данных и открытия набора данных страны здесь выполняется установка интервала ожидания завершения перемещения по родительскому набору данных в 500 миллисекунд путем задания значения свойству набора данных `WaitEndMasterInterval`. Иными словами, если пользователь выполняет навигацию по таблице стран, то после изменения текущей записи система не сразу открывает набор данных регионов, выбирая оттуда только регионы текущей страны, а ожидает в течение 500 миллисекунд. Это позволяет открывать регионы только в том случае, когда пользователь закончит перемещение по таблице и действительно захочет рассмотреть страну и ее регионы.

Кроме того, здесь на информационную панель помещается количество записей стран путем присваивания свойству `Caption` метки `CtrNo` соответствующего значения.

Чтобы на информационной панели отображалось и количество регионов текущей страны, необходимо написать обработчик события `AfterOpen` (после открытия) для компонента регионов `DataSetRegion` (листинг 6.27).

Листинг 6.27. Обработчик события, наступающего после открытия набора данных регионов

```
procedure TFormMain.DataSetRegionAfterOpen(DataSet: TDataSet);
begin
  RegNo.Caption := IntToStr(DataSetRegion.RecordCount);
end;
```

Обычным образом пишется обработчик события щелчка по кнопке **Выйти**.

Теперь дополнительно реализуем только функцию добавления нового региона для иллюстрации взаимодействия главной и подчиненной таблиц в операторе `MODIFY`.

Создайте новую форму, присвойте ей имя `FormAddReg`, сохраните модуль с именем AddReg. Разместите на этой форме метки, поля редактирования и кнопки, чтобы получился следующий вид — рис. 6.15.

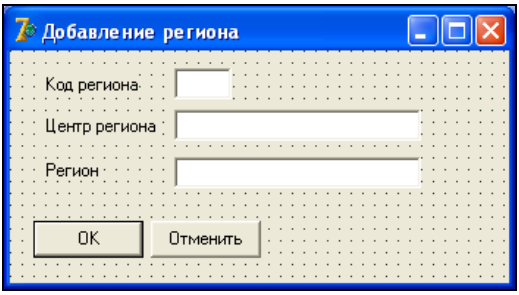


Рис. 6.15. Модуль добавления региона к текущей стране

Не будем слишком щепетильными и не станем сейчас реализовывать необходимые проверки — вы уже сами сможете все правильно сделать. Напишем только обработчики появления формы и щелчка по кнопке **OK**.

В событии `OnShow` очищаем все поля редактирования и устанавливаем фокус ввода на код региона (листинг 6.28).

Листинг 6.28. Обработчик события отображения формы добавления регионов

```
procedure TFormAddReg.FormShow(Sender: TObject);
begin
  CODREG.Text := '';
  CENTER.Text := '';
  NAMEREG.Text := '';
  CODREG.SetFocus;
end;
```

При щелчке по кнопке **OK** выполняем добавление новой записи в таблицу регионов (листинг 6.29).

Листинг 6.29. Добавление региона

```
procedure TFormAddReg.BOKClick(Sender: TObject);
begin
  FormMain.DataSetRegion.Insert;
  FormMain.DataSetRegion.FieldByName('CODREG').AsString := CODREG.Text;
  FormMain.DataSetRegion.FieldByName('CENTER').AsString := CENTER.Text;
  FormMain.DataSetRegion.FieldByName('NAMEREG').AsString := NAMEREG.Text;
  FormMain.DataSetRegion.Post;
end;
```

Заметьте, что здесь мы не устанавливаем значения кода страны, `CODCTR`, не без основания рассчитывая, что это будет выполнено автоматически за счет правильного написания оператора `INSERT`.

Свяжем обычным образом главный и вспомогательный модули. В главном модуле напишем следующий обработчик события щелчка по кнопке **Добавить регион** — листинг 6.30.

Листинг 6.30. Добавление региона из главной формы

```
procedure TFormMain.BAddRegClick(Sender: TObject);
begin
  FormAddReg.ShowModal;
  DataSetRegion.FullRefresh;
  RegNo.Caption := IntToStr(DataSetRegion.RecordCount);
end;
```

Здесь нам необходимо явно вызвать метод `FullRefresh`, чтобы в сетке регионов отобразился и код страны.

Вот и все. Мы создали программу при минимальном написании кода. Похорошему помимо выполнения необходимых проверок при добавлении региона следовало бы также обеспечить в программе средства, позволяющие отображать и скрывать список регионов на этой форме. Это делается присваиванием у компонента `DBGridRegion` свойству `Visible` значений `True` или `False`.

Текст программы находится в каталоге Chapter06\MasterDetail\FIBPlus.

6.3.2. Использование компонентов IBX

Скопируйте проект в новый каталог или выполните те же действия по размещению на форме компонентов, за исключением компонентов FIBPlus. Положите на форму с вкладки **InterBase** компоненты `Database`, `Transaction` и два компонента `DataSet`. Присвойте им соответствующие имена и установите необходимые свойства.

Удалите из предложения **uses** ссылки на компоненты FIBPlus.

У компонента `DataSetCountry` обычным образом сформируйте оператор `SELECT`, добавив предложение `ORDER BY NAME`:

```
select * from REFCTR ORDER BY NAME
```

Создайте остальные операторы SQL.

Компонент DataSetRegion свяжите с родительским набором данных, присвоив его свойству DataSource значение DataSourceCountry. Создайте оператор SELECT:

```
select * from REFREG WHERE CODCTR = :CODCTR
ORDER BY NAMEREG
```

Предложение WHERE здесь указывает, что из таблицы выбираются только те записи регионов, которые относятся к текущей стране.

Обычным образом сгенерируйте остальные операторы SQL.

Обработчики событий похожи на те, которые были в программе с использованием компонентов FIBPlus. В обработчик события FormShow нужно добавить строку открытия набора данных регионов:

```
DataSetRegion.Open;
```

В обработчик события AfterOpen компонента набора данных регионов нужно добавить строку чтения всех записей:

```
DataSetRegion.FetchAll;
```

В модуле добавления региона в обработчике щелчка по кнопке **OK** нужно явно задать присваивание кода страны из таблицы стран:

```
FormMain.DataSetRegion.FieldByName('CODCTR').AsString :=
    FormMain.DataSetCountry.FieldByName('CODCTR').AsString;
```

Надо сказать, что изменения здесь небольшие по отношению к программе, написанной с использованием компонентов FIBPlus.

Текст программы находится в каталоге Chapter06\MasterDetail\IBX.

6.4. Изменение упорядоченности наборов данных

Часто пользователю бывает нужно изменять упорядоченность отображаемых данных в процессе выполнения программы. Например, он может пожелать отображать страны, упорядоченные по кодам или по кратким названиям, выводить списки людей в порядке их фамилий или упорядоченные по фамилии отца или матери. Давайте рассмотрим подобного типа задачу, обратив внимание на варианты модификации оператора SELECT и на интерфейс для пользователя.

6.4.1. Использование компонентов FIBPlus

Создайте новый проект в Delphi или скопируйте в новый каталог какой-нибудь из существующих. Я использовал проект просмотра и удаления DisplayDelete.

Разместите на форме нужные компоненты и установите для них те свойства, которые вы уже можете устанавливать с блеском. Форма должна принять следующий вид — рис. 6.16.

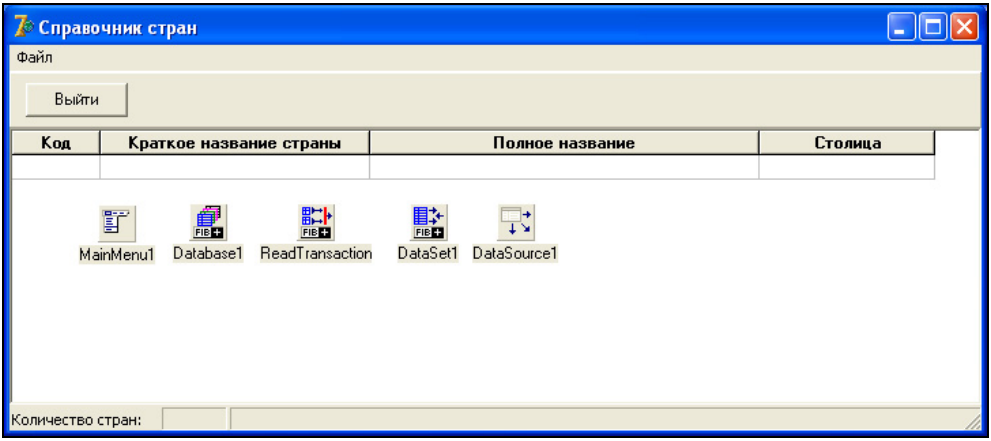


Рис. 6.16. Программа, позволяющая упорядочивать записи

В меню нужно поместить два элемента: **Порядок** и **Выйти**, не забыв разместить между ними обычный разделитель (рис. 6.17).

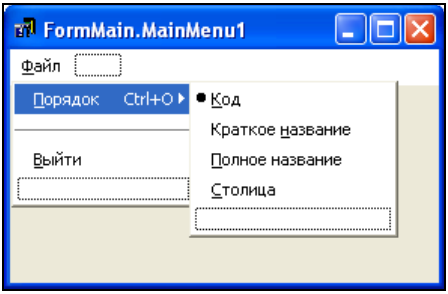


Рис. 6.17. Главное меню программы

Для элемента **Выйти** напишите оператор завершения приложения. Свяжите с ним событие щелчка по кнопке **Выйти**.

Для элемента **Порядок** нужно создать субменю — вложенное меню. Для этого на элементе меню **Порядок** необходимо щелкнуть правой кнопкой мыши

и в появившемся контекстном меню выбрать строку **Create Submenu**. В субменю нужно создать четыре элемента, как показано на рис. 6.17. Присвойте им соответственно имена `MByCode`, `MByName`, `MByFullName`, `MByCapital`. У первого элемента установите в `True` свойство `Checked`. Для каждого из этих элементов нужно установить в `True` значение свойства `RadioItem`. В результате все элементы превратятся в радиогруппу: отметка одного из элементов снимает отметки со всех остальных. Не забудьте также в поля `Hint` поместить тексты соответствующих подсказок для пользователя.

Теперь напишем обработчик события щелчка мышью по элементу заголовка сетки — `OnTitleClick` (листинг 6.31).

Листинг 6.31. Обработчик события щелчка по заголовку сетки

```
procedure TFormMain.DBGrid1TitleClick(Column: TColumn);
var S: String;
begin
  S := Column.FieldName;
  if DataSet1.OrderClause <> S then
  begin
    DataSet1.Close;
    DataSet1.OrderClause := S;
    DataSet1.Open;
  end;
end;
```

Обработчику передается параметр `Column`, у которого свойство `FieldName` содержит имя столбца, по заголовку которого пользователь щелкнул мышью.

Компонент набора данных в FIBPlus имеет свойство `OrderClause`, которое позволяет читать и устанавливать значение — содержимое предложения `ORDER BY` в операторе `SELECT`. Здесь переменной `S` присваивается значение имени столбца, набор данных закрывается, меняется значение его предложения `ORDER BY` и набор данных опять открывается. Дело в том, что при открытом наборе данных нельзя менять упорядоченность.

Так как переоткрытие набора данных требует времени и увеличивает сетевой трафик, если база данных находится на сервере, то прежде чем менять упорядоченность, проверяется, не установлен ли уже соответствующий порядок.

Вообще говоря, процедуру можно было бы чуть упростить, убрав внутреннюю переменную `S` и создав предложение `ORDER BY` следующим образом:

```
DataSet1.OrderClause := Column.FieldName;
```

Этот вариант пройдет, если мы позволяем выполнять упорядочение по *каждому* столбцу, отображаемому в сетке, как в нашем случае. Однако если изменение порядка требуется только для некоторых столбцов, а такое в реальной жизни бывает чаще всего, без локальной переменной не обойтись. Например, чтобы исключить упорядочивание по названию столицы, нужно было бы написать приблизительно такой код:

```
S := '';
if Column.FieldName = 'CODCTR' then
  S := 'CODCTR';
if Column.FieldName = 'NAME' then
  S := 'NAME';
if Column.FieldName = 'FULLNAME' then
  S := 'FULLNAME';
if S <> '' then
begin
  DataSet1.Close;
  DataSet1.OrderClause := S;
  DataSet1.Open;
end;
```

Всю нашу красивую картину мира разрушает присутствие элементов меню, выполняющих похожую функцию изменения порядка. Вот обработчик события выбора в меню элемента **Код**:

Листинг 6.32. Обработчик выбора в главном меню элемента *Код*

```
procedure TFormMain.MByCodeClick(Sender: TObject);
begin
  if MByCode.Checked then exit;
  MByCode.Checked := True;
  DataSet1.Close;
  DataSet1.OrderClause := 'CODCTR';
  DataSet1.Open;
end;
```

Аналогично создаются обработчики выбора других вариантов упорядочения. В них мы явно указываем, что соответствующий элемент должен стать отмеченным. Этого не происходит при обработке щелчка по заголовку сетки.

Проблему малой кровью можно решить следующим образом, переписав обработчик щелчка по заголовку сетки — листинг 6.33.

Листинг 6.33. Новая версия обработчика события щелчка по заголовку сетки

```
procedure TFormMain.DBGrid1TitleClick(Column: TColumn);
begin
  if Column.FieldName = 'CODCTR' then
    MByCodeClick(nil);
  if Column.FieldName = 'NAME' then
    MByNameClick(nil);
  if Column.FieldName = 'FULLNAME' then
    MByFullNameClick(nil);
  if Column.FieldName = 'CAPITAL' then
    MByCapitalClick(nil);
end;
```

Здесь происходит обращение к обработчику выбора соответствующего элемента меню. Эти обработчики получают один параметр `Sender` типа `TObject`. Так как нам нет необходимости передавать какие бы то ни было параметры, мы поступаем стандартным для Delphi способом — передаем пустое значение, **`nil`**. Это один из лучших вариантов[11]. В особенности, если у вас предусмотрено некоторое количество элементов, управляющих упорядочением. Например, вы для этих целей используете еще и переключатели или обращаетесь к другой форме, в которой помимо всего прочего задается новый порядок просмотра.

> ***Замечание***
>
> Предложение `ORDER BY` в операторе `SELECT`, как мы с вами рассмотрели в *главе 5*, может устанавливать направление упорядоченности (по возрастанию или по убыванию) путем задания после каждого имени столбца, по которому ведется упорядочение, еще и ключевых слов `ASCENDING` (по возрастанию, это значение принимается по умолчанию) и `DESCENDING` (по убыванию). Точно так же
> в методе `OrderClause` мы можем после имени столбца через пробел указать и направление упорядоченности. Действительно, многие реализации программных продуктов временами вызывают искреннее восхищение.

В FIBPlus у компонента набора данных существует интересный метод — `DoSort`. Он позволяет выполнить локальную сортировку набора данных — т. е. изменить порядок записей в наборе данных на клиентской машине, не обращаясь к базе данных на сервере. Это полностью исключает сетевой трафик при изменении упорядоченности набора данных.

---

[11] Спросите у Вострикова, какой он использует изощренный механизм в подобных случаях. Лично я не рискну предлагать вам такое.

При использовании этого метода обработчик выбора элемента меню **Код** будет выглядеть следующим образом — листинг 6.34.

Листинг 6.34. Обработчик выбора в главном меню элемента *Код* для локальной сортировки

```
procedure TFormMain.MByCodeClick(Sender: TObject);
begin
  if MByCode.Checked then exit;
  MByCode.Checked := True;
  DataSet1.DoSort(['CODCTR'], [True]);
end;
```

Методу передается два параметра, оба являются массивами. Первый содержит список полей, по которым выполняется локальная сортировка. Второй должен содержать столько же элементов, сколько и первый. Эти элементы задают порядок сортировки — возрастающий (тогда передается `True`) или убывающий (передается `False`).

Оба наши параметра-массива содержат по одному элементу. Мы должны включить эти одноэлементные массивы в квадратные скобки.

Упорядочение происходит мгновенно, при этом сохраняется и текущая запись (та, которая была отмечена на сетке). Поэкспериментировав с программой, добавив множество записей в справочник стран (совершенно бессмысленных, но позволяющих проверить, как работает сортировка в различных вариантах), замечаем, что все прописные буквы будут предшествовать строчным. Это не очень соответствует нашему представлению о записях, упорядоченных по строковым полям.

Использование набора символов `WIN1251` и порядка сортировки `PXW_CYRL` или `WIN1251_UA` позволяют в случае предложения `ORDER BY` получить подходящую упорядоченность прописных и строчных, русских и латинских букв (*см. приложение 2*, где подробно рассматриваются отличия в этих порядках сортировки). Не следует использовать порядок сортировки `WIN1251`.

В нашем случае сложности с упорядочением кодов стран, содержащих строчные и прописные русские и латинские буквы, связаны с тем, что при создании таблицы для домена, на котором основывается столбец кода страны, не было указано предложение `COLLATE`. В результате в качестве порядка сортировки был выбран порядок по умолчанию для этого набора символов — `WIN1251`. Он-то как раз и создает все те безобразия с упорядоченностью дан-

ных. Такие же результаты, разумеется, мы получим и в программе, созданной с использованием компонентов IBX.

Если мы можем воздействовать на средства упорядочения в предложении `ORDER BY`, задавая необходимый порядок сортировки для соответствующих столбцов таблицы, то с локальной сортировкой дело состоит несколько сложнее. Там тоже можно использовать любые алгоритмы упорядочения, однако это потребует достаточно больших усилий. Здесь мы этого рассматривать не станем. Одно можно сказать — локальную сортировку можно смело использовать для числовых столбцов.

Текст программы находится в каталоге Chapter06\Order\FIBPlus.

> ***Замечание***
>
> Как само собой разумеющееся при описании программ мы не акцентируем внимание на том, что в каждой программе у нас существуют средства вывода подсказок. Каждая программа содержит процедуру `ShowHint`, помещающую в нужное место строки состояния текст текущей подсказки, связанной с элементом меню или с кнопкой на инструментальной панели.

6.4.2. Использование компонентов IBX

В этом случае у нас не будет такого обилия вариантов. Для изменения упорядоченности набора нужно будет закрыть набор данных, изменить оператор `SELECT`, указав в предложении `ORDER BY` нужное имя столбца, и заново открыть набор данных.

Создайте новый проект в Delphi или скопируйте только что созданный проект в другой каталог и откройте его в Delphi, удалив компоненты FIBPlus и убрав из предложения **`uses`** ссылки на модули FIBPlus.

Из модуля Main проекта, например, InsertUpdate для IBX скопируйте компоненты базы данных, транзакции и набора данных. После этого следует опять связать компонент `DataSource1` с набором данных `DataSet1`, потому что после удаления компонентов FIBPlus такая связь была потеряна.

Переделка нашего проекта под компоненты IBX в этом случае требует минимальных изменений в программе. Нам нужно изменить лишь тексты четырех обработчиков выбора элементов меню, связанных с изменением упорядоченности.

Вот, например, как выглядит теперь обработчик выбора в главном меню элемента **Код** — листинг 6.35.

Листинг 6.35. Обработчик выбора в главном меню элемента *Код*

```
procedure TFormMain.MByCodeClick(Sender: TObject);
begin
  if MByCode.Checked then exit;
  MByCode.Checked := True;
  DataSet1.Close;
  DataSet1.SelectSQL.Text := 'select * from REFCTR ORDER BY CODCTR';
  DataSet1.Open;
  DataSet1.FetchAll;
end;
```

Чтобы изменить оператор `SELECT`, соответствующее значение нужно присвоить свойству `Text`, включенному в свойство `SelectSQL` компонента набора данных.

Поскольку `SelectSQL` является объектом класса `TStrings`, формирование нужной строки можно выполнить и другим способом, очистив значение (метод `Clear`) и добавив нужные строки (метод `Add`). Пример такого варианта представлен в обработчике выбора элемента меню **Краткое название** (листинг 6.36).

Листинг 6.36. Обработчик выбора в главном меню элемента *Краткое название*

```
procedure TFormMain.MByNameClick(Sender: TObject);
begin
  if MByName.Checked then exit;
  MByName.Checked := True;
  DataSet1.Close;
  DataSet1.SelectSQL.Clear;
  DataSet1.SelectSQL.Add('select * from REFCTR ORDER BY NAME');
  DataSet1.Open;
  DataSet1.FetchAll;
end;
```

Для одной строки этот вариант не кажется вполне подходящим, но в том случае, когда вам понадобится добавлять некоторое количество строк, следует использовать именно такую последовательность операций.

Запустите программу на выполнение, убедитесь, что она работает точно так же, как и предыдущая версия.

Текст программы находится в каталоге Chapter06\Order\IBX.

Что там за перевалом?

Мы написали много полезных программ, познакомились с компонентами, которые могут сильно облегчить нам жизнь при создании программ. Испытали немало приключений, выполняя упорядочение наборов данных.

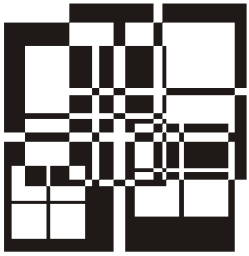
Сейчас вы можете легко соединить вместе функциональность всех написанных в этой главе программ в одну полную программу. Если есть время и желание, сделайте это.

На протяжении всей книги мы напишем с вами еще множество интересных и нужных в дальнейшей вашей работе программ.

Похоже, в результате этой деятельности мы немного подустали. Лучший отдых — смена вида деятельности. Запускайте isql или, лучше, IBExpert.

Настало время вплотную заняться необыкновенно интересным и достаточно сложным оператором `SELECT`, который мы и рассмотрим в следующей *главе*. Там мы проведем большое количество экспериментов с этим оператором, осуществляя различные выборки данных из нашей учебной базы данных.

ГЛАВА 7



Поиск данных

Для выборки данных из реляционной базы данных используется оператор SELECT — самый мощный и самый сложный оператор в языке SQL. Его сложность заключается даже не столько в синтаксисе, хотя и он достаточно сложен, сколько в большом количестве вариантов его использования для получения различных данных из существующих таблиц базы данных. Разумеется, всех его возможностей мы рассматривать не будем. Детально исследуем лишь те его особенности, которые нам, скорее всего, пригодятся в нашей деятельности по созданию программ, работающих с базами данных.

Для выполнения операторов этой главы нужно запустить на выполнение программу IBExpert, соединиться с нашей базы данных и вызвать инструмент SQL Editor.

7.1. Синтаксис оператора *SELECT*

Мы опять же рассмотрим упрощенный синтаксис этого оператора.

```
SELECT [ALL | DISTINCT] <список выбора>
  FROM <ссылка на таблицу> [, <ссылка на таблицу>]...
  [WHERE <условия поиска>]
  [GROUP BY <список столбцов> [HAVING <условия поиска>]]
  [ORDER BY <список упорядочения>];
```

Разберем предложения оператора.

`SELECT [ALL | DISTINCT] <список выбора>` — список выбираемых столбцов, констант, функций или выражений, разделенных запятыми.

Необязательное ключевое слово `ALL` означает, что выбираются все строки, соответствующие условию поиска. Это значение по умолчанию, и обычно в

операторе не указывается. Ключевое слово DISTINCT означает, что в выходной набор данных попадут лишь отличающиеся строки, дубликаты будут отбрасываться.

Сам список выбираемых столбцов (или список выбора) может содержать имена требуемых столбцов, константы, выражения или символ *, который означает, что выбираются все столбцы таблицы.

Столбцы могут быть представлены или просто своими именами или перед именем столбца может присутствовать имя таблицы (или псевдоним таблицы — см. далее), после которого стоит точка. Никаких пробелов в таких конструкциях использовать нельзя. Синтаксис уточненной конструкции выглядит следующим образом:

```
<имя таблицы>.<имя столбца>
```

или

```
<псевдоним таблицы>.<имя столбца>
```

Уточненные имена столбцов обязательно должны присутствовать в операторах SELECT, где присутствует несколько таблиц, это позволит избежать двусмысленности при обращении к столбцам с одинаковым именем из разных таблиц. Такая ситуация может возникнуть при выполнении соединений таблиц.

При задании имени столбца из таблицы, константы или любого выражения можно после ключевого слова AS указать текст, который будет помещаться в заголовок отображаемой таблицы. Такой текст называется псевдонимом или алиасом столбца. Если псевдоним содержит пробелы, специальные символы или буквы кириллицы, то этот текст нужно заключить в кавычки. Например:

```
SELECT CODCTR AS "Код страны",
       CODREG AS "Код региона",
       NAMEREG AS REGION_NAME
  FROM REFREG
  WHERE CODCTR = '558';
```

Русские тексты здесь заключены в кавычки, потому что содержат символы, недопустимые в именах, а например REGION_NAME, являясь правильным именем в SQL, может обойтись без кавычек.

> **ВНИМАНИЕ!**
>
> Не путайте описанные таким образом псевдонимы столбцов с псевдонимами таблиц (см. дальше). Если псевдонимы таблиц можно использовать в операторе, где они определены, для уточнения ссылок на столбцы, то с псевдонимами столбцов дело обстоит несколько сложнее. В разных версиях разных серверов баз данных их можно использовать по-разному. Чуть позже мы рассмотрим некоторые варианты.

`FROM <ссылка на таблицу>` задает список имен таблиц, в которых осуществляется поиск. Здесь же можно указывать соединения (`JOIN`) таблиц, представления (см. в конце этой главы), внешние функции (UDF) и имена процедур выбора. Хранимые процедуры мы будем рассматривать несколько позже в *главе 9*.

Необязательное предложение `WHERE <условия поиска>` определяет условия поиска строк в таблицах. На самом деле это предложение не является таким уж необязательным. Если таблица в базе данных содержит тысячи строк, то, во-первых, вам вряд ли понадобится за один раз такое количество записей, поскольку просмотреть их вы просто физически не сможете, а во-вторых, выборка больших объемов данных на сервере приводит к увеличению сетевого трафика и перегрузке сети.

Необязательное предложение `GROUP BY <имя столбца>` группирует найденные строки в соответствии со значением указанного столбца.

Необязательное предложение `HAVING <условия поиска>` определяет дополнительные условия поиска для использования в `GROUP BY`.

Необязательное предложение `ORDER BY <список имен>` позволяет упорядочить найденные строки, указывая список имен столбцов (здесь опять же можно использовать уточненные имена в виде `<имя таблицы>.<имя столбца>`), а также направление сортировки для каждого столбца — ключевые слова `ASCENDING` (по возрастанию) или `DESCENDING` (по убыванию).

7.2. Простые варианты поиска данных

Рассмотрим вначале наиболее простые варианты использования оператора. Мы будем отыскивать данные каждый раз только в одной из таблиц базы данных.

Запустите IBExpert. Если вы вносили изменения в базу данных, то лучше ее пересоздать заново, выполнив последовательно скрипты создания базы данных, доменов, таблиц, триггеров, добавив в таблицы справочные данные, оперативные данные и выполнив изменение оперативных данных.

Если вы все еще не зарегистрировали базу данных, зарегистрируйте ее в IBExpert (**Database | Register Database** (База данных | Регистрировать базу данных)). Соединитесь с базой данных (**Database | Connect to Database** (Соединиться с базой данных)). Вызовите инструмент SQL Editor — меню **Tools | SQL Editor** (Инструменты | Редактор операторов SQL) или нажмите клавишу <F12>. В этом окне вы вводите операторы выборки данных, для их выполнения нужно щелкнуть по кнопке **Run** или нажать клавишу <F9>.

Не рекомендую для выполнения примеров этой главы использовать isql. Кодировка DOS, под которой выполняется isql, отличается от кодировки Windows. Поэтому вместо русских текстов вы получите на экране набор диких символов.

> ***Замечание***
>
> Обратите, пожалуйста, внимание на вид, в котором представляются операторы SELECT в этой главе. Вместо того чтобы даже короткие операторы размещать на одной строке, я стараюсь отдельные группы переносить на новые строки. И IBExpert, и isql позволяют для ввода любого оператора использовать произвольное количество строк. Кроме того, я пытаюсь использовать соответствующие отступы от левого края (как правило, два пробела) для выделения вложенных или подчиненных синтаксических единиц.

7.2.1. Задание отображаемых столбцов

Сразу после ключевого слова SELECT указывается список выбора — список столбцов таблицы, литералов, констант или выражений, которые попадут в результирующий набор данных.

В самом простом варианте оператора SELECT можно указать вместо списка отображаемых столбцов символ *, например:

```
SELECT *
  FROM PEOPLE;
```

Этот оператор возвращает все столбцы всех строк таблицы PEOPLE.

Выполните оператор в isql или IBExpert для нашей учебной базы данных. Он вернет 112 строк (листинг 7.1).

Листинг 7.1. Выбор всех столбцов всех строк таблицы людей

```
COD NAME1      NAME2           NAME3       BIRTHDAY    SEX FULLNAME ...
=== ========== =============== =========== ========== === ===========
  1 НАТАЛЬЯ    АЛЕКСАНДРОВНА   ИВАНОВА     01.12.1961  1   ИВАНОВА...
  2 АЛЕКСАНДР  АЛЕКСАНДРОВИЧ   ИВАНОВ      07.01.1960  0   ИВАНОВ...
  3 ИРИНА      АЛЕКСАНДРОВНА   ИВАНОВА     30.08.1991  1   ИВАНОВА...
  4 НИКОЛАЙ    АЛЕКСАНДРОВИЧ   ИВАНОВ      21.02.1990  0   ИВАНОВ...
  5 ОЛЕГ       ЮРЬЕВИЧ         ГРАЧЕВ      02.08.1959  0   ГРАЧЕВ...
  6 РОМАН      ОЛЕГОВИЧ        ГРАЧЕВ      08.08.1990  0   ГРАЧЕВ...
  7 АЛЬФИЯ     РУСЛАНОВНА      ИБРАГИМОВА  01.07.1960  1   ИБРАГИМОВА...
...
```

Приличным людям показывать такой список как-то неудобно. Следует исключить из списка вычисляемое поле, которое дублирует фамилию, имя и

отчество. Также нет необходимости выводить коды родителей и супругов, которые обычному человеку ничего не говорят.

Выполним оператор в следующем виде:

```
SELECT COD, NAME3, NAME1, NAME2, BIRTHDAY FROM PEOPLE;
```

Мы получим тоже 112 записей. Каждая будет содержать код, фамилию, имя, отчество и дату рождения человека.

Заголовки нам с вами, как проектировщикам таблиц нашей базы данных, понятны. Однако другим людям следует долго объяснять, что это такое. Лучше сразу сформировать заголовки на русском языке. Выполните:

```
SELECT COD AS "Код",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       NAME3 AS "Фамилия",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE;
```

Результат станет более понятным, заголовки содержат правильные тексты (листинг 7.2).

Листинг 7.2. Добавление осмысленных заголовков

```
Код Имя          Отчество         Фамилия      Дата рождения
=== ========== ================ ============ =============
  1 НАТАЛЬЯ      АЛЕКСАНДРОВНА    ИВАНОВА      01.12.1961
  2 АЛЕКСАНДР    АЛЕКСАНДРОВИЧ    ИВАНОВ       07.01.1960
  3 ИРИНА        АЛЕКСАНДРОВНА    ИВАНОВА      30.08.1991
  4 НИКОЛАЙ      АЛЕКСАНДРОВИЧ    ИВАНОВ       21.02.1990
  5 ОЛЕГ         ЮРЬЕВИЧ          ГРАЧЕВ       02.08.1959
  6 РОМАН        ОЛЕГОВИЧ         ГРАЧЕВ       08.08.1990
  7 АЛЬФИЯ       РУСЛАНОВНА       ИБРАГИМОВА   01.07.1960
...
```

В списке выбора могут присутствовать не только имена столбцов, но и литералы. Выполните следующий оператор:

```
SELECT COD AS "Код",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       'Фамилия:' AS "Текст",
       NAME3 AS "Фамилия",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE;
```

Замечание

Все операторы SELECT в примерах я заканчиваю символом точка с запятой, однако в случае использования IBExpert это не является обязательным. Для isql в конце каждого оператора нужно вводить точку с запятой.

В результате выполнения запроса один столбец будет во всех строках содержать один и тот же текст — "Фамилия:". Ему мы и присвоили заголовок Текст (листинг 7.3).

Листинг 7.3. Добавление в результат выбора текстовой строки "*Фамилия:*"

```
Код Имя          Отчество         Текст     Фамилия      Дата рождения
=== ========== ================ ========  ============ =============
  1 НАТАЛЬЯ      АЛЕКСАНДРОВНА    Фамилия:  ИВАНОВА      01.12.1961
  2 АЛЕКСАНДР    АЛЕКСАНДРОВИЧ    Фамилия:  ИВАНОВ       07.01.1960
  3 ИРИНА        АЛЕКСАНДРОВНА    Фамилия:  ИВАНОВА      30.08.1991
  4 НИКОЛАЙ      АЛЕКСАНДРОВИЧ    Фамилия:  ИВАНОВ       21.02.1990
  5 ОЛЕГ         ЮРЬЕВИЧ          Фамилия:  ГРАЧЕВ       02.08.1959
  6 РОМАН        ОЛЕГОВИЧ         Фамилия:  ГРАЧЕВ       08.08.1990
  7 АЛЬФИЯ       РУСЛАНОВНА       Фамилия:  ИБРАГИМОВА   01.07.1960
...
```

Вряд ли такому оператору можно найти применение, но это работает.

Вместо имен столбцов мы можем в операторе указать и функции. Самая простая и наиболее часто используемая функция подсчета количества значений COUNT. Выполните оператор:

```
SELECT COUNT(*)
  FROM PEOPLE;
```

Вы получите число 112 — количество людей в таблице PEOPLE. В данном случае вместо символа * в функции COUNT вы можете указать имя любого столбца этой таблицы. Однако при использовании, например, ключевого слова DISTINCT (см. далее) указание различных имен столбцов вместо символа * может дать совершенно другие результаты.

Функцию COUNT в разных вариантах мы еще рассмотрим через несколько страниц.

Выполним следующий оператор для таблицы всех сотрудников всех организаций.

```
SELECT AVG(SALARY)
  FROM STAFF;
```

Здесь используется агрегатная функция AVG, которая рассчитывает среднюю заработную плату всех сотрудников всех организаций. Результатом будет число 21423.52.

Аналогичным образом мы можем найти минимальное (функция MIN) и максимальное (MAX) значение заработной платы всех сотрудников всех организаций, хранящихся в нашей базе данных. Выполните соответствующие операторы:

```
SELECT MIN(SALARY)
  FROM STAFF;
```

Довольно скромно — всего 1500.00.

```
SELECT MAX(SALARY)
  FROM STAFF;
```

Получим неплохую сумму 56000.00.

Мы можем также просуммировать все оклады всех сотрудников всех организаций:

```
SELECT SUM(SALARY)
  FROM STAFF;
```

Получим 347000.00. Не очень понятно, кому же это может пригодиться.

7.2.2. Упорядочение результата запроса. Предложение *ORDER BY*

По определению результат отображения никак не упорядочивается. Мы можем видеть, что в нашем случае он соответствует порядку помещения записей в базу данных (это видно по возрастанию значения первичного ключа, кода человека), однако никакого порядка реляционные базы данных не гарантируют.

Чтобы явно задать нужный нам порядок, в оператор следует ввести предложение ORDER BY. В предложении перечисляются имена столбцов таблицы или порядковые номера столбцов, указанных в списке выбора. По умолчанию сортировка выполняется в возрастающем порядке, как если бы вы задали ключевое слово ASCENDING, однако для различных столбцов в одном и том же предложении вы можете указать и убывающий порядок, задав ключевое слово DESCENDING. Для ключевого слова ASCENDING допустимо сокращение ASC, для DESCENDING — DESC.

Предложение ORDER BY имеет следующий синтаксис:

```
ORDER BY {<имя столбца> | <номер столбца>} [{ASC[ENDING] | DESC[ENDING]]}
```

Помимо имен столбцов в этом предложении мы можем указать и порядковые номера столбцов в списке выбора. Столбцы в списке нумеруются, начиная с единицы. Чтобы можно было использовать номера столбцов, вы должны яв-

но перечислить столбцы в списке выбора. Вариант * в этом случае недопустим.

Что интересно — столбцы, по которым выполняется сортировка данных, вовсе необязательно должны быть включены в состав выбираемых столбцов, хотя и не совсем понятно, кому это может понадобиться. Такую сортировку допускают делать далеко не все реляционные базы данных.

Рассмотрим примеры с нашей таблицей людей. Введите и выполните:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  ORDER BY NAME3, NAME1;
```

Вначале выполняется упорядочение списка по столбцу NAME3 (фамилия), а затем внутри списка еще и по NAME1 (имя). Второй уровень сортировки называется вложенной (nested) сортировкой. Количество уровней вложенности не ограничивается.

В нашем примере мы получили список, отсортированный по фамилиям в возрастающем порядке. Если несколько человек имеют одну и ту же фамилию, то строки еще упорядочиваются и по именам. Как и должно быть.

Кстати, такая сортировка строковых полей называется сортировкой в лексикографическом порядке, когда порядок символов в точности соответствует расположению букв в алфавите соответствующего языка. При использовании набора символов WIN1251 правильное расположение строк дают порядки сортировки PXW_CYRL и WIN1251_UA *(см. приложение 2)*.

Листинг 7.4. Упорядочение результатов выборки по фамилии и имени

```
Код Фамилия      Имя           Отчество        Дата рождения
=== ============ ============ ================ =============
...
104 ЕРМИШИН      АНТОН         ИВАНОВИЧ         03.08.1985
109 ЕРМИШИН      АРТЕМ         ИВАНОВИЧ         03.08.1985
111 ЕРМИШИН      ЕВГЕНИЙ       АРТЕМОВИЧ        11.11.2004
100 ЕРМИШИН      ИВАН          ПЕТРОВИЧ         14.05.1960
107 ЕРМИШИН      КОНСТАНТИН    АНТОНОВИЧ        14.08.2004
106 ЕРМИШИН      ПАВЕЛ         АНТОНОВИЧ        14.08.2004
108 ЕРМИШИНА     СВЕТЛАНА      АРТЕМОВНА        11.11.2004
112 ЕРМИШИНА     СВЕТЛАНА      АНТОНОВНА        05.12.2005
 68 ЗАБРОДИН     НИКОЛАЙ       ЮРЬЕВИЧ          01.12.1955
...
```

В листинге 7.4 можно увидеть, что по отчествам порядок не устанавливается: сначала идет ЕРМИШИНА СВЕТЛАНА АРТЕМОВНА, а затем ЕРМИШИНА СВЕТЛАНА АНТОНОВНА. Если нужно, мы можем добавить еще один уровень сортировки — по отчествам.

Зададим упорядочение не в виде имен столбцов, а указанием их номеров. Напомню, что столбцы в списке выбора нумеруются, начиная с единицы, а в самом списке выбора нельзя указывать символ *.

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  ORDER BY 2, 3;
```

Получаем такой же упорядоченный список, как и в предыдущем случае, что мы видели в листинге 7.4. Проверим, можно ли в одном предложении указывать и номера, и имена. Введем и выполним:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  ORDER BY NAME3, 3;
```

Это работает точно так же, как и в обоих предыдущих случаях.

Теперь проверим, действительно ли работает вариант различного направления сортировки в разных столбцах в одном предложении. Изменим первый оператор, добавив упорядочение по столбцу NAME1 (имя человека) в убывающем порядке:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  ORDER BY NAME3, NAME1 DESCENDING;
```

Все работает замечательно (листинг 7.5). Фамилии сортируются в возрастающем порядке, а имена у однофамильцев — в убывающем.

Листинг 7.5. Упорядочение результатов выборки в разных направлениях

```
Код Фамилия     Имя          Отчество        Дата рождения
=== =========== ============ =============== =============
...
106 ЕРМИШИН     ПАВЕЛ        АНТОНОВИЧ       14.08.2004
107 ЕРМИШИН     КОНСТАНТИН   АНТОНОВИЧ       14.08.2004
100 ЕРМИШИН     ИВАН         ПЕТРОВИЧ        14.05.1960
111 ЕРМИШИН     ЕВГЕНИЙ      АРТЕМОВИЧ       11.11.2004
109 ЕРМИШИН     АРТЕМ        ИВАНОВИЧ        03.08.1985
104 ЕРМИШИН     АНТОН        ИВАНОВИЧ        03.08.1985
112 ЕРМИШИНА    СВЕТЛАНА     АНТОНОВНА       05.12.2005
108 ЕРМИШИНА    СВЕТЛАНА     АРТЕМОВНА       11.11.2004
 68 ЗАБРОДИН    НИКОЛАЙ      ЮРЬЕВИЧ         01.12.1955
...
```

Интересно, как поведут себя пустые значения (`NULL`) в результате сортировки? Стандарт допускает помещение всех таких строк либо в самое начало списка, либо в самый конец сортируемого набора данных. Посмотрим, как решается этот вопрос в наших системах управления базами данных.

Выполните следующий оператор поиска данных для организации с номером 18. В эту организацию мы специально для иллюстраций поместили трех сотрудников, чьи коды и оклады имеют пустые значения.

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 18
  ORDER BY CODPEOPLE;
```

Получим следующий список, упорядоченный по кодам — листинг 7.6.

Листинг 7.6. Упорядочение результата при наличии пустых значений

```
Код сотрудника Должность                      Оклад
============== =============================== =========
            14 Исполнительный директор          38500.00
<null>         Неизвестная должность3             <null>
<null>         Неизвестная должность2             <null>
<null>         Неизвестная должность1             <null>
```

Видно, что все пустые значения помещаются в самый *конец* списка. Этот результат мы получим при использовании Firebird версии 1.5, в Firebird 2.0 все

получается с точностью до наоборот — пустые значения находятся в *начале* списка.

Теперь проведем эксперимент по использованию псевдонимов столбцов, указанных в списке выбора оператора `SELECT`.

Попробуем выполнить следующий оператор.

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  ORDER BY "Фамилия", "Имя";
```

Если вы попытаетесь выполнить этот оператор в Firebird версии 1.5 и ниже, то такой номер не пройдет. Вы получите сообщение об ошибке: неизвестный столбец `"Фамилия"`, неизвестный столбец `"Имя"`.

В Firebird 2.0 и InterBase 2007 все работает правильно. Однако только этим на сегодняшний день и ограничивается использование псевдонимов столбцов. Их можно использовать только в предложении `ORDER BY`. В предложении, например, `WHERE` использование псевдонима столбца даст ошибку — неправильное имя столбца.

7.2.3. Использование ключевого слова *DISTINCT*

Выполните следующий оператор:

```
SELECT NAME3 AS "Фамилия"
  FROM PEOPLE
  ORDER BY NAME3;
```

Вы получите список из 112 фамилий, среди которых есть много повторяющихся.

> ***Замечание***
>
> Это, кстати, один из примеров нарушения принципов реляционной алгебры. Результатом выборки данных из таблицы базы данных (из отношения, в терминах реляционной алгебры) должно быть также отношение. Однако то, что мы получили, отношением не является, потому что содержит и одинаковые строки.

Чтобы убрать ненужные нам повторы, необходимо в оператор ввести ключевое слово `DISTINCT`:

```
SELECT DISTINCT NAME3 AS "Фамилия"
  FROM PEOPLE
  ORDER BY NAME3;
```

Теперь мы получим список из 57 фамилий, среди которых не будет повторяющихся.

Чтобы не выводить все записи, а посмотреть только на общее количество фамилий и количество различных фамилий, можно использовать функцию COUNT, которая выполнит расчет количества строк и вернет нужное число.

Выполните:

```
SELECT COUNT(NAME3) AS "Количество записей"
  FROM PEOPLE;
```

Результатом будет число 112. Если выполнить оператор

```
SELECT COUNT(DISTINCT NAME3) AS "Количество записей"
  FROM PEOPLE;
```

то мы получим 57.

Коль речь пошла о функции COUNT, давайте проведем еще исследования. Выполним следующий оператор подсчета сотрудников организации с кодом 18. Там находится четыре сотрудника (см. листинг 7.6). Трое из них в столбцах "Код сотрудника" (CODPEOPLE) и "Оклад" содержат пустые значения.

```
SELECT COUNT(*) AS "COUNT(*)",
       COUNT(CODPEOPLE) AS "COUNT(CODPEOPLE)",
       COUNT(DISTINCT CODPEOPLE) AS "COUNT(DISTINCT CODPEOPLE)"
  FROM STAFF
  WHERE CODORG = 18;
```

Мы получим следующий результат — листинг 7.7.

Листинг 7.7. Результаты выполнения функции COUNT

```
COUNT(*) COUNT(CODPEOPLE) COUNT(DISTINCT CODPEOPLE)
======== ================ ========================
       4                1                        1
```

COUNT(*) — это счетчик всех строк. Возвращает, естественно, число 4.

COUNT(CODPEOPLE) возвращает 1. Следовательно, при выполнении этой функции пустые значения не учитываются.

Довольно странный на первый взгляд результат, единица, возвращаемый в случае выполнения функции COUNT(DISTINCT CODPEOPLE). Видно, что в данном случае пустые значения рассматриваются как одинаковые, что не соответствует общему подходу к значениям NULL, когда одно пустое значение не равно другому пустому значению. На самом деле это нормальное поведение функции COUNT, одинаковое для всех реляционных баз данных. Функция

подсчитывает все *непустые* значения. Точно так же выполняются все остальные агрегатные функции — в своих подсчетах они игнорируют пустые значения.

Кроме того, ключевое слово `DISTINCT` в операторе `SELECT` воспринимает все пустые значения (`NULL`) как одинаковые. Это, кстати, один из аргументов у критиков использования пустого значения — с одной стороны, при выполнении операций сравнения два пустых значения никогда не равны друг другу, с другой стороны, в агрегатных функциях и при использовании ключевого слова `DISTINCT` все пустые значения равны. Я не думаю, что подобное поведение и вообще отсутствие безупречной математической "чистоты" нас с вами сильно напугает.

Проведем простой эксперимент. В листинге 7.6 дан список всех сотрудников организации с номером 18. Там присутствует одна "нормальная" строка и три строки, содержащие пустые значения в столбцах "Код сотрудника" и "Оклад". Выполним оператор:

```
SELECT DISTINCT CODPEOPLE AS "Код сотрудника"
  FROM STAFF
  WHERE CODORG = 18
  ORDER BY CODPEOPLE;
```

Мы получим *две* строки, где только одна строка имеет пустое значение в столбце `CODPEOPLE`. Действительно, при использовании ключевого слова `DISTINCT` все пустые значения рассматриваются как одинаковые.

В одной умной книге по реляционным базам данных я как-то прочел, что существуют реляционные СУБД, в которых при задании ключевого слова `DISTINCT` нельзя использовать арифметические (а может быть, и не только арифметические) выражения. Давайте проверим, как с этим обстоят дела у нас, в наших любимых серверах баз данных.

Выполним оператор подсчета количества окладов, умноженных на два для организации с кодом 18:

```
SELECT COUNT(SALARY * 2)
  FROM STAFF WHERE CODORG = 18;
```

Эта организация содержит всего четыре записи сотрудников (см. листинг 7.6), где только одна запись содержит непустое значение оклада (столбец `SALARY`). Оператор вернет, естественно, число 1.

Проверим, есть ли у нас возможность использовать в этом случае ключевое слово `DISTINCT`. Выполним оператор:

```
SELECT COUNT(DISTINCT SALARY * 2)
```

```
FROM STAFF WHERE CODORG = 18;
```

Все работает нормально. Оператор также возвращает число 1. Может быть только немного смущает бессмысленность этих двух последних операторов. Мы их использовали исключительно в научно-исследовательских целях.

> ***Замечание***
>
> В некоторых реляционных системах управления базами данных ключевое слово DISTINCT может относиться не ко всей строке, а только к одному из столбцов или к группе столбцов в списке выбора. Тогда в результат поиска попадут лишь строки, не имеющие дублирующих значений по этому столбцу. В нашем же случае, при использовании InterBase и Firebird, DISTINCT относится ко всей выбираемой строке, не допуская дубликатов целых строк. Не думаю, что это как-то ограничивает наши возможности по работе с данными.

7.2.4. Задание условий выборки. Предложение *WHERE*

Предложение WHERE позволяет задать условие, на основании которого строки таблицы будут попадать в результирующий набор данных. Строка помещается в выходной набор данных, если она удовлетворяет указанному, подчас довольно сложному, условию. При отсутствии этого предложения в выходной набор данных помещаются все строки исходной таблицы (таблиц).

Следует отдавать себе отчет, что выражение в предложении WHERE является *логическим* выражением, возвращающим истинностное значение TRUE, FALSE или UNKNOWN. В выборку будут попадать только те записи таблицы, для которых это выражение дает истинный результат (TRUE). Соответственно, к выражению применимы все законы создания выражений исчисления высказываний. На практике чаще всего используется закон де Моргана, про который далее мы скажем несколько слов.

Надо еще помнить, что операции сравнения, в которых принимают участие пустые значения (NULL), никогда не дают истинного значения. (Вы помните, что это не действует для агрегатных функций и при использовании в списке выбора ключевого слова DISTINCT.) Для таких столбцов дополнительно следует также использовать проверку типа IS NULL или IS NOT NULL.

Вся мощь оператора SELECT в реляционных базах данных проявляется в первую очередь в предложении WHERE. Из большого, а временами очень большого количества исходных данных оператор позволяет выбрать релевантный (то есть соответствующий потребностям пользователя) объем данных.

Синтаксис предложения *WHERE*

Начальное определение синтаксиса предложения WHERE до неприличия простое:

```
WHERE <условия поиска>
```

В самих же условиях поиска, которые много сложнее по синтаксису, содержатся те самые условия, которые позволяют выполнить необходимый отбор данных. Несколько упрощенный синтаксис условий поиска:

```
<условия поиска> ::= <значение> <оператор> {<значение> | (<выбор одного>)}
  | <значение> [NOT] BETWEEN <значение> AND <значение>
  | <значение> [NOT] LIKE <значение>
  | <значение> [NOT] IN (<значение> [, <значение>] ... | <список выбора>)
  | <значение> IS [NOT] NULL
  | <значение> <оператор> {ALL | SOME | ANY} (<выбор списка значений одного столбца>)
  | EXISTS (<список выбора>)
  | SINGULAR (<список выбора>)
  | <значение> [NOT] CONTAINING <значение>
  | <значение> [NOT] STARTING [WITH] <значение>
  | <значение> <оператор> {ALL | SOME | ANY} (<выбор списка значений одного столбца>)
  | EXISTS (<список выбора>)
  | SINGULAR (<список выбора>)
  | (<условия поиска>)
  | NOT <условия поиска>
  | <условия поиска> OR <условия поиска>
  | <условия поиска> AND <условия поиска>
```

Здесь `<оператор>` — один из допустимых операторов сравнения: =, <, >, <=, >=, !<, !> <>, !=. Восклицательный знак используется в этих операторах как отрицание. Кроме того, в этом качестве также используется и символ ^. К перечню операторов сравнения следует также добавить и следующие: ^=, ^>, ^<.

```
<значение> ::= {
  <имя столбца>
  | <константа> | <выражение> | <функция>
  | udf ([<параметр> [, <параметр>]...])
  | NULL | USER}
```

Мы видим, что значением может быть имя столбца, любая допустимая константа, пустое значение, имя пользователя, подключенного в настоящий момент к базе данных, выражение, обращение к встроенной функции или к внешней функции, UDF.

Синтаксис условий поиска сильно похож на описание условий домена и условий в предложении `CHECK` в ограничениях таблицы.

Не спеша, просмотрим по порядку возможные варианты выбора нужных нам строк.

Использование операторов сравнения

Рассмотрим использование операторов сравнения в конструкции:

```
<значение> <оператор> {<значение> | (<выбор одного>)}
```

Операторы сравнения применимы к любым типам данных, кроме BLOB.

Выполним следующий оператор выборки всех данных из таблицы REFREG:

```
SELECT *
  FROM REFREG;
```

Мы получим список всех регионов всех стран, хранящихся в нашей базе данных — штаты Соединенных Штатов, графства Великобритании, регионы России. Трудно себе представить, что кого-то такой список может заинтересовать.

Предложение WHERE позволяет задать условие поиска — условие, на основании которого в результат выборки будут попадать данные из базы данных.

Введем в оператор предложение WHERE, в котором укажем код интересующей нас страны:

```
SELECT *
  FROM REFREG
  WHERE CODCTR = 'USA';
```

Мы получим все 50 штатов США.

Чтобы упорядочить список, например, по названию центров регионов (в нашем случае, по названию столиц штатов), выполним следующий оператор:

```
SELECT *
  FROM REFREG
  WHERE CODCTR = 'USA'
  ORDER BY CENTER;
```

Получим соответствующий результат.

Теперь можем обратить внимание на отсутствие нужной красоты в этом результате. Во-первых, код страны явно не нужен — он одинаков во всех строках. Во-вторых, следует задать разумные заголовки.

Введите и выполните следующий оператор:

```
SELECT CODREG  AS "Код штата",
       NAMEREG AS "Штат",
       CENTER  AS "Столица штата"
  FROM REFREG
  WHERE CODCTR = 'USA'
  ORDER BY CENTER;
```

Это уже совсем другое дело — понятно и красиво (листинг 7.8).

Листинг 7.8. Список штатов США

```
Код штата Штат            Столица штата
========= ================ ============================
NY        New York         ALBANY
MD        Maryland         ANNAPOLIS
GA        Georgia          ATLANTA
ME        Maine            AUGUSTA
TX        Texas            AUSTIN
LA        Louisiana        BATON ROUGE
ND        North Dakota     BISMARCK
ID        Idaho            BOISE
MA        Massachusetts    BOSTON
...
```

Пора, пожалуй, проявить патриотизм и выбрать в нашей базе все регионы *нашей* страны, России.

```
SELECT CODREG  AS "Код региона",
       NAMEREG AS "Регион",
       CENTER  AS "Центр региона"
  FROM REFREG
  WHERE CODCTR = '558'
  ORDER BY CENTER;
```

Мы получим следующий список — листинг 7.9.

Листинг 7.9. Регионы России

```
Код региона Регион                      Центр региона
=========== ========================== ============================
19          Республика Хакасия          АБАКАН
29          Архангельская область       АРХАНГЕЛЬСК
30          Астраханская область        АСТРАХАНЬ
22          Алтайский край              БАРНАУЛ
31          Белгородская область        БЕЛГОРОД
28          Амурская область            БЛАГОВЕЩЕНСК
32          Брянская область            БРЯНСК
25          Приморский край             ВЛАДИВОСТОК
15          Республика Северная Осетия  ВЛАДИКАВКАЗ
33          Владимирская область        ВЛАДИМИР
34          Волгоградская область       ВОЛГОГРАД
35          Вологодская область         ВОЛОГДА
...
```

Здесь мы рассмотрели вариант <значение> <оператор> <значение>.

В следующем операторе поиска мы рассмотрим чуть более сложную структуру — <значение> <оператор> (<выбор одного>). "Выбор одного" — это оператор SELECT, который возвращает ровно одно значение одного столбца из любой таблицы нашей базы данных или ни одного значения.

Например, мы можем предыдущий оператор представить в таком виде, если бы мы не помнили код России:

```
SELECT CODREG  AS "Код региона",
       NAMEREG AS "Регион",
       CENTER  AS "Центр региона"
  FROM REFREG
  WHERE CODCTR = (SELECT CODCTR
                    FROM REFCTR
                    WHERE NAME = 'РОССИЯ')
  ORDER BY CENTER;
```

Результат будет точно таким же, как и в листинге 7.9. В этом варианте внутренний, или вложенный, оператор SELECT должен возвращать ровно одно значение или никакого значения. В последнем случае не возникает никакого исключения, просто результат будет содержать нулевое количество строк. Можете проверить это, выполнив, например, следующий оператор:

```
SELECT CODREG  AS "Код региона",
       NAMEREG AS "Регион",
       CENTER  AS "Центр региона"
  FROM REFREG
  WHERE CODCTR = (SELECT CODCTR
                    FROM REFCTR
                    WHERE NAME = 'dummy')
  ORDER BY CENTER;
```

Внутренние, вложенные операторы SELECT или, как их еще называют, подзапросы, мы подробнее рассмотрим ближе к концу этой главы.

Рассмотрим более сложный поиск данных в таблице сотрудников. В нашей базе данных содержатся списки сотрудников только для трех присутствующих в базе данных организаций. Выполните оператор:

```
SELECT *
  FROM STAFF;
```

Появится список, состоящий из 19 сотрудников различных организаций.

Чтобы получить список сотрудников только одной организации, нужно в оператор выборки добавить предложение WHERE. Пусть нам нужно получить

список сотрудников только организации с кодом 11. Это организация с названием ЗАО "ИнфоТел". Введем и выполним:

```
SELECT *
  FROM STAFF
  WHERE CODORG = 11;
```

Тот же результат мы получим, если выполним несколько более длинный, но, несомненно, более информативный для нас запрос:

```
SELECT *
  FROM STAFF
  WHERE CODORG = (SELECT COD
                  FROM ORGANIZATION
                  WHERE NAME = 'ЗАО "ИнфоТел"');
```

Появится следующий список сотрудников этой организации — листинг 7.10.

Листинг 7.10. Список сотрудников организации ЗАО "ИнфоТел"

```
COD CODPEOPLE CODORG DUTIES                          SALARY    NET_SALARY
=== ========= ====== ============================== ========= ==========
  1        14     11 Генеральный директор           56000.00  48720.00
  4        21     11 Заместитель директора          42000.00  36540.00
  5        23     11 Главный бухгалтер              15000.00  13050.00
  6        24     11 Финансовый директор            12000.00  10440.00
  7        16     11 Заместитель директора по кадрам 13200.00  11484.00
  8        30     11 Начальник отдела               12000.00  10440.00
  9        31     11 Заместитель начальника отдела  11000.00   9570.00
 10        33     11 Программист                    23700.00  20619.00
 11        41     11 Программист                    23800.00  20706.00
 12        46     11 Программист                    23900.00  20793.00
 13       109     11 Программист                    23600.00  20532.00
 14       110     11 Техник                          7300.00   6351.00
 15       104     11 Системный администратор        31400.00  27318.00
```

Чтобы придать списку более благообразный вид, следует несколько изменить наш оператор поиска, убрав код организации, вычисляемый столбец (NET_SALARY) и добавив человеческие заголовки:

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11;
```

Получим следующее — листинг 7.11.

Листинг 7.11. Улучшенный список сотрудников организации ЗАО "ИнфоТел"

```
Код сотрудника Должность                      Оклад
============== ============================== =========
            14 Генеральный директор           56000.00
            21 Заместитель директора          42000.00
            23 Главный бухгалтер              15000.00
            24 Финансовый директор            12000.00
            16 Заместитель директора по кадрам 13200.00
            30 Начальник отдела               12000.00
            31 Заместитель начальника отдела  11000.00
            33 Программист                    23700.00
            41 Программист                    23800.00
            46 Программист                    23900.00
           109 Программист                    23600.00
           110 Техник                          7300.00
           104 Системный администратор        31400.00
```

Понятно, что полной красоты мы еще не добились, у нас в списке присутствуют какие-то коды вместо фамилий сотрудников. Это мы обязательно исправим, но несколько позже, когда начнем рассматривать соединение таблиц.

Давайте теперь любопытства ради найдем среднюю зарплату сотрудников этой организации и месячный фонд заработной платы. Для этих целей используем агрегатные функции `AVG` и `SUM`:

```
SELECT AVG(SALARY), SUM(SALARY)
  FROM STAFF
  WHERE CODORG = 11;
```

Получим два числа: 22684.61 (средняя заработная плата) и 294900.00 (месячный фонд заработной платы).

Понятно, что такие функции, как `AVG` и `SUM`, применимы лишь к числовым данным — целым и дробным.

Теперь мы еще больше заинтересовались этой организацией и решили посмотреть списки сотрудников, чей оклад выше средней заработной платы, и заодно сотрудников с окладом, меньшим средней заработной платы.

Казалось бы, для получения первого списка можно ввести следующий оператор выборки данных:

```
SELECT *
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY > AVG(SALARY);
```

Однако не хочет сервер базы данных выполнять такой оператор. Мы получаем сообщение: `Cannot use an aggregate function in a WHERE clause, use HAVING instead`. То есть, нас уверяют, что нельзя использовать агрегатную функцию в предложении `WHERE`, следует вместо этого предложения использовать предложение `HAVING`. В это как-то не очень верится. Мы же знаем, что SQL — очень гибкий и мощный язык.

На самом деле упрек нам совершенно справедлив — в предложении `WHERE` действительно нельзя использовать никакие агрегатные функции в "чистом" виде. Наш оператор следует изменить следующим образом:

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY > (SELECT AVG(SALARY)
                     FROM STAFF
                     WHERE CODORG = 11);
```

Здесь для получения среднего значения оклада мы используем вложенный оператор `SELECT`, который возвращает *одно* значение, что требуется в данной конструкции предложения `WHERE`. Как таковая агрегатная функция в предложении `WHERE` не используется. Все теперь работает прекрасно, мы получаем следующий список из семи записей — листинг 7.12.

Листинг 7.12. Список сотрудников, имеющих оклад выше среднего

```
Код сотрудника Должность                      Оклад
============== ============================== =========
            14 Генеральный директор            56000.00
            21 Заместитель директора           42000.00
            33 Программист                     23700.00
            41 Программист                     23800.00
            46 Программист                     23900.00
           109 Программист                     23600.00
           104 Системный администратор         31400.00
```

Соответственно, чтобы получить список сотрудников, чьи оклады не превышают среднее значение, нужно выполнить оператор:

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
```

```
  WHERE CODORG = 11 AND
        SALARY <= (SELECT AVG(SALARY)
                   FROM STAFF
                   WHERE CODORG = 11);
```

Этот список будет содержать шесть записей (листинг 7.13).

Листинг 7.13. Список сотрудников, имеющих оклад ниже среднего

```
Код сотрудника Должность                       Оклад
============== ================================ =========
            23 Главный бухгалтер                15000.00
            24 Финансовый директор              12000.00
            16 Заместитель директора по кадрам  13200.00
            30 Начальник отдела                 12000.00
            31 Заместитель начальника отдела    11000.00
           110 Техник                            7300.00
```

Использование варианта *BETWEEN*

Близок к операторам сравнения вариант BETWEEN, который, напомню, выглядит следующим образом:

```
<значение> [NOT] BETWEEN <значение> AND <значение>
```

Это применимо как к числовым, так и к строковым столбцам, дате, времени — ко всем, за исключением BLOB.

Этот вариант требует, чтобы значение находилось в указанном диапазоне, *включая* граничные значения.

Еще раз обратимся к полному списку сотрудников организации ЗАО "ИнфоТел" — см. листинг 7.11. Выберем из этого списка только тех сотрудников, чей оклад находится в диапазоне от 12 000 до 23 700:

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY BETWEEN 12000 AND 23700;
```

Мы получим следующий результат — листинг 7.14.

Листинг 7.14. Список сотрудников организации с окладами от 12 000 до 23 700

```
Код сотрудника Должность                       Оклад
============== ================================ =========
            23 Главный бухгалтер                15000.00
```

```
 24 Финансовый директор                12000.00
 16 Заместитель директора по кадрам    13200.00
 30 Начальник отдела                   12000.00
 33 Программист                        23700.00
109 Программист                        23600.00
```

Посмотрите: действительно оклады находятся в указанном диапазоне, включая и граничные значения.

Точно такой же результат мы получим, если используем соответствующее логическое выражение и операторы сравнения:

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY >= 12000 AND
        SALARY <= 23700;
```

Замечание

Логическое выражение в предложении `WHERE` нашего примера содержит только операторы конъюнкции (логическое И). По этой причине я здесь не использую скобок. Вообще же очень рекомендую всегда использовать скобки для явного задания порядка выполнения любых операторов, в особенности логических, потому что не часто на лету, когда в поте лица своего пишешь гениальную программу (при обычном отставании от утвержденного графика), вспомнишь порядок выполнения этих операций. Я вот только помню, что конъюнкция выполняется прежде дизъюнкции. Скобки совершенно необходимы, когда у вас довольно сложное логическое выражение[12]. Иначе выяснение причин, почему неверно осуществляется выборка данных, может затянуться на длительное время. Правда, есть еще более скверный вариант — вы будете получать неверные данные и на голубом глазу уверять заказчика, что так все и должно быть.

Проверка на присутствие в списке значений (вариант *IN*)

В варианте `IN` вы можете задать список, среди элементов которого должно (или не должно) находиться значение указанного столбца:

```
<значение> [NOT] IN ({<значение> [, <значение>] ... | <список выбора>})
```

В этом варианте можно задать как явно представленный список литералов или выражений, так и указать оператор `SELECT`, который возвращает произвольное количество значений ровно одного столбца.

---

[12] Есть еще один неприятный момент для любителей языка Delphi, который раньше назывался Объектный Паскаль. Там при использовании операций сравнения и логических операций порядок выполнения действий отличается от того, что принято в языке SQL. В этом языке и сами операции сравнения следует заключать в скобки, чтобы избежать ошибок трансляции.

Пример явно заданного списка. Мы можем получить список всех штатов США. Если же нам нужно выбрать только несколько штатов по их кодам, мы выполним следующий оператор:

```
SELECT CODREG  AS "Код штата",
       NAMEREG AS "Штат",
       CENTER  AS "Столица штата"
  FROM REFREG
  WHERE CODCTR = 'USA' AND
        CODREG IN ('NY', 'MD', 'TX')
  ORDER BY CENTER;
```

В результате получится список — листинг 7.15.

Листинг 7.15. Список выбранных штатов

```
Код штата Штат            Столица штата
========= ================ ============================
NY        New York         ALBANY
MD        Maryland         ANNAPOLIS
TX        Texas            AUSTIN
```

Точно такой же результат мы получим, если несколько усложним наш оператор выборки и зададим внутренний `SELECT`, который также использует вариант `IN`.

```
SELECT CODREG  AS "Код штата",
       NAMEREG AS "Штат",
       CENTER  AS "Столица штата"
  FROM REFREG
  WHERE CODCTR = 'USA' AND
        CODREG IN (SELECT CODREG
                   FROM REFREG
                   WHERE CODCTR = 'USA' AND
                         CENTER IN ('ALBANY', 'ANNAPOLIS', 'AUSTIN'))
  ORDER BY CENTER;
```

Для знатоков географии США и английского языка такой запрос может оказаться весьма осмысленным.

Вернемся к нашей стране. Выберем теперь все организации, находящиеся в России, в "столице Поволжья", г. Саратове. Здесь список `IN` будет содержать только одно значение. Оператор поиска такой:

```
SELECT NAME AS "Организация"
  FROM ORGANIZATION
  WHERE CODCTR = '558' AND
```

```
      CODREG IN ('64')
  ORDER BY NAME;
```

Результат будет следующим — листинг 7.16.

Листинг 7.16. Выбор организаций г. Саратова

```
Организация
================================================
ЗАО "Саратовский авиационный завод"
ВОЛГОИНВЕСТБАНК
ГАЗПРОМБАНК САРАТОВ
ГП "Приборомеханический завод"
ГП "Приборомеханический завод"
ГП "САРАТОВСКИЙ АГРЕГАТНЫЙ ЗАВОД"
Детский сад № 123
ЗАО "ИнфоТел"
МУП "Городской центр размещения рекламы
НАРАТБАНК ЭНГЕЛЬСКИЙ ФИЛИАЛ
ОАО "Саратовдизельаппарат"
ОАО "САРАТОВСКИЙ ЗАВОД ТЯЖЕЛЫХ ЗУБОРЕЗНЫХ СТАНКОВ"
ОАО "Энергомаш"
ОАО"САРАТОВСКИЙ ЗАВОД ЗУБОСТРОГАЛЬНЫХ СТАНКОВ"
ОАО"САРАТОВСКИЙ ПОДШИПНИКОВЫЙ ЗАВОД"
ООО "Алан"
ООО "Какаду"
ООО "ПОДИУМ"
ООО" Инфомаркет"
Саратовское электроагрегатное производственное объединение
ЭКОНОМБАНК ЭНГЕЛЬС
ЭНГЕЛЬС-БАНК
```

Если вдруг, паче чаянья, вы захотите найти в вашей базе данных все предприятия, не находящиеся в "столице Поволжья", вы должны использовать отрицание только что выполненного оператора:

```
SELECT NAME AS "Организация"
  FROM ORGANIZATION
  WHERE CODCTR = '558' AND
        CODREG NOT IN ('64')
  ORDER BY NAME;
```

Вы получите 12 других организаций.

Замечание

Вообще-то естественнее было бы в этих двух примерах использовать обычный оператор сравнения — равенство в первом случае и неравенство во втором. Так что рассматривайте эти запросы только как примеры, которым подражать необязательно.

Нужные данные мы можем также получать, используя оператор `SELECT`, который возвращает произвольное (возможно, пустое) количество значений одного столбца. Мы также можем использовать и данные, получаемые из других таблиц.

Например, нам нужно получить список всех организаций, для которых в нашей базе данных присутствует описание сотрудников. Выполним оператор:

```
SELECT COD AS "Код",
       NAME AS "Организация"
  FROM ORGANIZATION
  WHERE COD IN (SELECT CODORG FROM STAFF)
  ORDER BY NAME;
```

Здесь внутренний оператор `SELECT` возвращает те коды организаций, которые хранятся в другой таблице, таблице сотрудников `STAFF`.

Получим список из трех организаций (листинг 7.17).

Листинг 7.17. Выбор организаций, для которых представлен список сотрудников

```
Код Организация
=== ================
 16 АЛЬФА-БАНК
 18 БАЛЧУГ
 11 ЗАО "ИнфоТел"
```

Так как в одной организации описано несколько сотрудников, то оператор `SELECT` вернет много одинаковых значений кодов организаций, которые будут помещены в список выбора `IN`. Не знаю, как это скажется на времени выполнения оператора, но лично мне все это не очень нравится. По этой причине
я бы ввел в оператор поиска ключевое слово `DISTINCT`, благодаря которому в первоначальный список попадут только отличающиеся значения кодов:

```
SELECT COD AS "Код",
       NAME AS "Организация"
  FROM ORGANIZATION
  WHERE COD IN (SELECT DISTINCT CODORG FROM STAFF)
  ORDER BY NAME;
```

Мы получим точно такой же список, что и представленный в листинге 7.17, однако на душе станет светлее — так или иначе мы улучшили вариант выполнения поиска.

Использование функций *ALL*, *SOME*, *ANY*, *EXISTS*, *SINGULAR*

В условиях поиска можно использовать несколько функций:

```
  | <значение> <оператор> {ALL | SOME | ANY} (<выбор списка значений одного столбца>)
  | EXISTS (<список выбора>)
  | SINGULAR (<список выбора>)
```

После имени каждой такой функции в скобках записывается оператор SELECT, вид которого и возвращаемые им значения зависят от конкретной функции.

Для исследования возможностей этих функций выведем список сотрудников организации с кодом 16:

```
SELECT DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 16;
```

В этой организации указано двое сотрудников (листинг 7.18).

Листинг 7.18. Список сотрудников организации с кодом 16

```
Должность                      Оклад
============================== =========
Ночной сторож                    1500.00
Служба безопасности             12000.00
```

Также выведем список всех сотрудников организации с кодом 11:

```
SELECT DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11;
```

Получим список — листинг 7.19.

Листинг 7.19. Список сотрудников организации с кодом 11

```
Должность                      Оклад
============================== =========
Генеральный директор            56000.00
```

```
Заместитель директора                42000.00
Главный бухгалтер                    15000.00
Финансовый директор                  12000.00
Заместитель директора по кадрам      13200.00
Начальник отдела                     12000.00
Заместитель начальника отдела        11000.00
Программист                          23700.00
Программист                          23800.00
Программист                          23900.00
Программист                          23600.00
Техник                                7300.00
Системный администратор              31400.00
```

Функция *ALL*

Функция используется в следующем виде:

```
<значение> <оператор> ALL(<выбор списка значений одного столбца>)
```

Здесь `<оператор>` — любой оператор сравнения. Аргументом функции `ALL` должен быть оператор `SELECT`, возвращающий произвольное количество значений одного столбца. Такой оператор в литературе часто называют *скалярным подзапросом*. Вся конструкция возвращает значение "истина", если операция сравнения истинна для всех возвращаемых оператором `SELECT` значений.

Рассмотрим пример.

```
SELECT DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY >= ALL(SELECT SALARY
                      FROM STAFF
                      WHERE CODORG = 16);
```

Из таблицы сотрудников организации с кодом 11 выбираются все сотрудники, чей оклад не меньше оклада любого сотрудника организации с кодом 16. Результатом будет список, состоящий из девяти записей (листинг 7.20).

Листинг 7.20. Пример использования функции *ALL*

```
Должность                          Оклад
================================== =========
Генеральный директор                56000.00
Заместитель директора               42000.00
Главный бухгалтер                   15000.00
```

```
Заместитель директора по кадрам   13200.00
Программист                       23700.00
Программист                       23800.00
Программист                       23900.00
Программист                       23600.00
Системный администратор           31400.00
```

Посмотрите на листинги 7.18 и 7.19. Мы видим, что оператор вернул нам правильный список. По правде сказать, здесь более естественно было бы использовать функцию MAX:

```
SELECT DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY <= (SELECT MAX(SALARY)
                   FROM STAFF
                   WHERE CODORG = 16);
```

Функции *ANY* и *SOME*

Это два названия одной и той же функции, т. е. синонимы.

```
<значение> <оператор> {ANY | SOME}(<выбор списка значений одного столбца>)
```

Конструкция возвращает значение "истина", если операция сравнения истинна хотя бы для одного значения, возвращаемого подзапросом.

Выполните следующий оператор:

```
SELECT DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY <= SOME(SELECT SALARY
                       FROM STAFF
                       WHERE CODORG = 16);
```

На человеческом языке это означает, что в список должны попасть те записи, для которых найдется такая запись, возвращаемая подзапросом, в которой оклад будет больше, чем оклад отыскиваемой записи. Запрос вернет четыре записи (листинг 7.21).

Листинг 7.21. Пример использования функции SOME

```
Должность                         Оклад
================================= =========
Финансовый директор                12000.00
```

```
Начальник отдела                    12000.00
Заместитель начальника отдела       11000.00
Техник                               7300.00
```

Вместо функции SOME можно применить в данном случае функцию MAX, и запрос будет более понятным:

```
SELECT DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY <= (SELECT MAX(SALARY)
                   FROM STAFF
                   WHERE CODORG = 16);
```

Этот запрос, разумеется, вернет те же четыре записи.

Функции *EXISTS* и *SINGULAR*

Это функции, которые возвращают значение "истина", если в списке выбора существует, как минимум, одно (EXISTS) или в точности одно (SINGULAR) возвращаемое значение. Фрагмент синтаксиса:

```
| EXISTS (<список выбора>)
| SINGULAR (<список выбора>)
```

Здесь <список выбора> — это оператор SELECT, который возвращает произвольное количество строк, содержащих произвольное количество столбцов. В литературе такой оператор часто называется *табличным подзапросом*.

***

Как мы могли увидеть, вместо перечисленных функций в предложении WHERE можно использовать другие, более наглядные средства. На самом деле эти функции очень удобны в триггерах и хранимых процедурах. Там они естественным образом применяются в операторах цикла и в условном операторе IF.

> ***Замечание***
>
> В одной книге, посвященной реляционным базам данных, я увидел диаметрально противоположное мнение. Там авторы считают, что использование подобных функций более удобно и естественно, чем применение таких слишком "сложных" функций, как MIN или MAX.

Проверка на пустое значение

Если столбцы ваших таблиц, входящих в условие поиска могут иметь пустое значение, то вам следует проявлять бдительность, задавая проверку их значений на NULL. Всегда следует перед выполнением "нормальных" проверок

уточнить, не является ли значение пустым. Мы можем использовать два варианта:

```
<значение> IS NULL
```

и

```
<значение> IS NOT NULL
```

Еще раз повторю, это действительно может стать источником больших ошибок — перед обычной, нормальной, проверкой проверяйте значения на NULL.

Поиск в строковых столбцах

Для строковых значений используются варианты LIKE, CONTAINING и STARTING WITH. Напомню синтаксис:

```
...
  | <значение> [NOT] LIKE <значение>
  | <значение> [NOT] CONTAINING <значение>
  | <значение> [NOT] STARTING [WITH] <значение>
...
```

Вариант *LIKE*

В варианте LIKE строковое значение должно содержать указанные символы. В этом варианте можно использовать шаблонные символы: процент (%) означает любое, в том числе и нулевое количество любых символов, знак подчеркивания (_) означает ровно один любой символ. LIKE является *чувствительным* к регистру, т. е. различает строчные и прописные буквы. На самом деле это неприятное ограничение можно очень легко и безболезненно обойти, применяя для имени столбца, используемого в выражении, функцию UPPER, которая возвращает все буквенные данные в верхнем регистре в любом, допустимом для набора используемых символов, алфавите.

Выберем все строки из таблицы PEOPLE:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  ORDER BY NAME3;
```

Получаем 112 записей, упорядоченных по фамилиям.

Введем условие, по которому фамилия должна заканчиваться на "ОВ". Для этого перед буквами "ОВ" используем шаблонный символ %.

```
SELECT COD AS "Код",
```

```
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  WHERE NAME3 LIKE '%ОВ'
  ORDER BY NAME3;
```

Здесь получаем 21 запись. В списке только мужчины (листинг 7.22).

Листинг 7.22. Выборка людей в варианте `LIKE '%ОВ'`

```
Код Фамилия      Имя          Отчество        Дата рождения
=== ========== ============ ============== =============
 66 БРЮКОВ       АЛЕКСАНДР    АЛЕКСЕЕВИЧ      15.02.1958
 70 БРЮКОВ       ИГОРЬ        АЛЕКСАНДРОВИЧ   12.05.1969
 72 БРЮКОВ       ИГОРЬ        ИГОРЕВИЧ        28.03.1989
 59 ВАЛЕНТИНОВ   ГЕННАДИЙ     ПАВЛОВИЧ        03.02.1928
 44 ВОЛОСОВ      АНДРЕЙ       СЕРГЕЕВИЧ       16.08.1968
...
```

Расширим условие поиска, выполним:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  WHERE NAME3 LIKE '%ОВ%'
  ORDER BY NAME3;
```

Мы добавили еще знак процента в условие: `LIKE '%ОВ%'`. То есть не только до, но и после символов "ОВ" может располагаться любое, в том числе и нулевое, количество любых символов. Теперь мы также получаем и женские фамилии. Список содержит 38 строк (листинг 7.23).

Листинг 7.23. Выборка людей в варианте `LIKE '%ОВ%'`

```
Код Фамилия      Имя          Отчество        Дата рождения
=== ========== ============ ============== =============
 66 БРЮКОВ       АЛЕКСАНДР    АЛЕКСЕЕВИЧ      15.02.1958
 70 БРЮКОВ       ИГОРЬ        АЛЕКСАНДРОВИЧ   12.05.1969
 72 БРЮКОВ       ИГОРЬ        ИГОРЕВИЧ        28.03.1989
 73 БРЮКОВА      АННА         ИГОРЕВНА        07.06.1994
 59 ВАЛЕНТИНОВ   ГЕННАДИЙ     ПАВЛОВИЧ        03.02.1928
 44 ВОЛОСОВ      АНДРЕЙ       СЕРГЕЕВИЧ       16.08.1968
 47 ВОЛОСОВА     ВАСИЛИСА     АНДРЕЕВНА       11.05.1988
...
```

Поскольку вариант LIKE является чувствительным к регистру, у вас, казалось бы, должны быть сложности при поиске данных, если вы не следовали моим мудрым советам и не вводили подобные строки в верхнем регистре. На самом деле ничего особенно страшного не произойдет. Даже если данные у вас введены вперемежку и прописными, и строчными буквами, вы можете использовать функцию UPPER, которая переведет все буквы строки в верхний регистр. Повторю — это относится только к символам алфавита, поддерживаемого вашим набором символов для этого столбца. Предыдущий запрос можно записать следующим образом:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  WHERE UPPER(NAME3) LIKE '%ОВ%'
  ORDER BY NAME3;
```

Благодаря возможности использовать шаблонные символы и функцию UPPER, вариант LIKE является наиболее гибким и может быть использован вместо двух других, часто используемых, строковых вариантов — STARTING WITH и CONTAINING.

Вариант *STARTING WITH*

Этот вариант требует, чтобы значение начиналось с указанных символов.

Предложение STARTING WITH является чувствительным к регистру. Чтобы из таблицы отобрать людей, чьи фамилии начинаются с буквы "А" (мы сделаем вид, что не знаем, в каком регистре набирались фамилии, и также используем функцию UPPER), нужно выполнить оператор:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  WHERE UPPER(NAME3) STARTING WITH 'А'
  ORDER BY NAME3;
```

Мы получим список, состоящий из двух строк (листинг 7.24).

Листинг 7.24. Выборка людей в варианте `STARTING WITH 'A'`

```
Код Фамилия      Имя          Отчество        Дата рождения
=== ========== ============ =============== =============
  9 АЛЬМЯШЕВ     РОМАН        ШАМИЛЬЕВИЧ      23.03.1991
  8 АЛЬМЯШЕВА    АЛИНА        ШАМИЛЬЕВНА      11.11.1971
```

Используя вариант `LIKE`, мы можем для получения того же результата выполнить оператор:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  WHERE UPPER(NAME3) LIKE 'A%'
  ORDER BY NAME3;
```

Вариант *CONTAINING*

Этот вариант не чувствителен к регистру. Он требует вхождения в значение указанных символов. Символы могут располагаться в любом месте значения — в начале строки, в середине или в конце.

Мы можем из списка людей выбрать только те фамилии, которые содержат конкретные символы, например, буквы "C":

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  WHERE NAME3 CONTAINING 'C'
  ORDER BY NAME3;
```

Здесь мы получим следующий список — листинг 7.25.

Листинг 7.25. Выборка людей в варианте `CONTAINING`

```
Код Фамилия      Имя          Отчество        Дата рождения
=== ========== ============ =============== =============
 44 ВОЛОСОВ      АНДРЕЙ       СЕРГЕЕВИЧ       16.08.1968
 47 ВОЛОСОВА     ВАСИЛИСА     АНДРЕЕВНА       11.05.1988
 77 САВЧЕНКО     АННА         ВАСИЛЬЕВНА      03.02.1950
 11 САФОНОВ      ВЛАДИМИР     НИКОЛАЕВИЧ      04.09.1974
 12 САФОНОВ      АРТЕМ        ВЛАДИМИРОВИЧ    24.09.1993
```

```
 10 САФОНОВА     МАРИНА        РОМАНОВНА       02.02.1973
101 СЕРЕБРЯКОВА  АНАСТАСИЯ     ИВАНОВНА        25.06.1965
 75 СИЛАЕВ       ЮРИЙ          СЕРГЕЕВИЧ       24.09.1952
 78 СИЛАЕВА      ЮЛИЯ          ЮРЬЕВНА         09.01.1977
```

При использовании варианта `LIKE` для получения того же самого результата мы можем предыдущий оператор представить в виде:

```
SELECT COD AS "Код",
       NAME3 AS "Фамилия",
       NAME1 AS "Имя",
       NAME2 AS "Отчество",
       BIRTHDAY AS "Дата рождения"
  FROM PEOPLE
  WHERE UPPER(NAME3) LIKE '%С%'
  ORDER BY NAME3;
```

Использование логических операций в условиях поиска

Вот последние четыре строки в описании синтаксиса предложения `WHERE`:

```
...
  | (<условия поиска>)
  | NOT <условия поиска>
  | <условия поиска> OR <условия поиска>
  | <условия поиска> AND <условия поиска>
```

Это означает, что любую правильную часть условия можно заключать в круглые скобки и что можно строить сколь угодно сложные логические условия, используя операции отрицания (`NOT`), дизъюнкции (`OR`) и конъюнкции (`AND`).

Следует быть аккуратным при построении сложных логических условий, особенно если вам нужно выполнить некоторые преобразования условий, содержащих отрицание. Люди часто допускают здесь ошибки.

В математической логике существует два замечательных закона де Моргана[13], которые позволяют выполнить преобразование отрицания конъюнкции и отрицания дизъюнкции. Законы простые:

```
NOT (A AND B) = NOT (A) OR NOT (B)
NOT (A OR B) = NOT (A) AND NOT (B)
```

То есть отрицание конъюнкции двух высказываний (в нашем случае — условий, которые так же, как и высказывания, возвращают истинностное значение: "истина" или "ложь") равно дизъюнкции отрицаний этих высказываний.

[13] Не могу удержаться, чтобы не напомнить, что в имени Моргана ударение на втором слоге.

Отрицание дизъюнкции равно конъюнкции отрицаний. Если вы это хорошо запомните, то у вас не будет головной боли при выполнении преобразований условий.

> ***Замечание***
> Тот факт, что в SQL используется не двухзначная, а трехзначная логика, ничего не меняет в используемых нами законах исчисления высказываний. В случае допустимости у столбцов пустых значений мы должны лишь проверить нужные нам столбцы на `NULL`. Этого достаточно.

Рассмотрим пример. Чуть раньше мы отображали список сотрудников, чей оклад находится в диапазоне от 12 000 до 23 700. Мы использовали следующий оператор (см. листинг 7.14):

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        SALARY >= 12000 AND SALARY <= 23700;
```

Если же нам нужен список сотрудников с окладами, не входящими в этот диапазон (то есть находящимися в точности за пределами этого диапазона), то мы должны выполнить отрицание выражения:

```
SALARY >= 12000 AND SALARY <= 23700
```

В соответствии с законом де Моргана мы получаем оператор:

```
SELECT CODPEOPLE AS "Код сотрудника",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
  WHERE CODORG = 11 AND
        (SALARY < 12000 OR SALARY > 23700);
```

Здесь нам нужно выполнить отрицание нашего выражения, задающего условия для окладов сотрудников. Отрицанием для операции сравнения `>=` будет `<` (или `!>=` или `^>=`), а для операции `<=` отрицанием будет `>` (`!<=`, `^<=`). Конъюнкцию мы меняем на дизъюнкцию. Кроме того, т. к. в предложении `WHERE` есть и конъюнкция, и дизъюнкция, мы должны использовать скобки, чтобы явно указать порядок выполнения вычислений.

Если кому-то вдруг по непонятной причине не очень нравится использование фактов из математической логики, то в данном случае легко можно использовать вариант `BETWEEN`, что мы и делали раньше:

```
SALARY BETWEEN 12000 AND 23700
```

для задания первоначального диапазона и

```
SALARY NOT BETWEEN 12000 AND 23700
```

для окладов за пределами этого диапазона. Применение закона де Моргана в этом случае выполнит сам сервер базы данных.

Порядок выполнения логических операций

Порядок выполнения логических операций на самом деле очень прост, хоть я и запугивал вас его сложностью:

8. Выполняются действия в скобках.
9. Арифметические операции умножения и деления.
10. Арифметические операции сложения и вычитания.
11. Отрицание (`NOT`).
12. Конъюнкция (`AND`).
13. Дизъюнкция (`OR`).

Операции с одинаковым приоритетом выполняются слева направо.

7.3. Соединение таблиц

Вы помните, что в процессе нормализации таблиц вам часто приходилось разделять таблицы на несколько взаимосвязанных таблиц, чтобы устранить избыточность данных. Тогда же я успокоил вас, сказав, что таблицы снова можно будет объединить с помощью операции соединения. Это действительно так.

Соединение таблиц в операторе `SELECT` является одним из наиболее мощных и элегантных средств реляционных баз данных.

Существует небольшое количество вариантов соединений, `JOIN`. Соединения задаются в списке имен таблиц:

```
FROM <ссылка на таблицу> [, <ссылка на таблицу>]...
```

Здесь не очень удачно названная мною конструкция `<ссылка на таблицу>` имеет приблизительно такой синтаксис:

```
<ссылка на таблицу> ::= { <соединяемая таблица> |
  <имя таблицы> |
  <имя представления> |
  <имя процедуры> [(<значение> [,<значение>] ...)]}
[<псевдоним>]
```

Мы в этом предложении пока только использовали вариант указания имени одной-единственной таблицы. Сейчас начнем рассматривать соединяемые таблицы.

```
<соединяемая таблица> ::= <ссылка на таблицу> <тип соединения>
  <ссылка на таблицу> ON <условия поиска> | (<ссылка на таблицу>)

<тип соединения> ::= [INNER] JOIN
  | {LEFT | RIGHT | FULL} OUTER JOIN
```

Есть внешние (`OUTER`) левые, правые и полные соединения, а есть внутренние (`INNER`) соединения. Если вид соединения не указывается, то по умолчанию предполагается внутреннее соединение.

Синтаксис довольно запутанный. Кроме того, в документации по InterBase этот синтаксис представлен с ошибками. Рассмотрим на примерах использование всех видов соединений.

7.3.1. Внешние соединения

Пожалуй, чаще всего используются внешние соединения, которые действительно позволяют объединить разделенные в результате нормализации таблицы в единое целое.

Левое внешнее соединение

Левое внешнее соединение чаще всего используется в наших операторах по причине его естественности.

Вначале отбираются строки первой, "главной", таблицы на основании условий, заданных в предложении `WHERE`. Затем к выбранным строкам добавляются данные из второй, присоединяемой, таблицы в соответствии с условиями соединения, заданными в предложении `ON`.

Вернемся опять к таблице сотрудников. В наших вариантах отображения списка присутствовали коды людей, но не было фамилий. Сейчас мы устраним этот неприятный недостаток.

Выполните, например, следующий оператор. Он выбирает всех сотрудников из организации с кодом 11, у которых заработная плата не превышает величины средней заработной платы в этой организации. При этом вместо ничего не значащего для нас кода сотрудника мы путем соединения таблиц добавляем в результат выборки фамилию, имя и отчество каждого человека.

```
SELECT PEOPLE.NAME3 || ' ' ||
       PEOPLE.NAME1 || ' ' ||
       PEOPLE.NAME2 AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
    LEFT OUTER JOIN PEOPLE
      ON STAFF.CODPEOPLE = PEOPLE.COD
```

```
WHERE CODORG = 11 AND
      SALARY <= (SELECT AVG(SALARY)
                 FROM STAFF
                 WHERE CODORG = 11)
ORDER BY 1;
```

Результат получается как раз такой, как мы хотели (листинг 7.26). Сравните с листингом 7.13, почувствуйте разницу.

Листинг 7.26. Левое внешнее соединение таблиц сотрудников и людей организации 11

```
Сотрудник                 Должность                          Оклад
========================= ================================== =========
ПАНКОВ ИГОРЬ АЛЕКСАНДРОВИЧ Начальник отдела                    12000.00
ПАНКОВА ГАЛИНА АНАТОЛЬЕВНА Заместитель начальника отдела       11000.00
ПАРШИНА АНТОНИНА ПЕТРОВНА  Техник                               7300.00
ПИНТЕРА АНДРЕЙ НИКОЛАЕВИЧ  Главный бухгалтер                   15000.00
ПИНТЕРА ЮЛИЯ НИКОЛАЕВНА    Финансовый директор                 12000.00
ШАФРАН НИНА СЕРГЕЕВНА      Заместитель директора по кадрам     13200.00
```

Давайте разберем выполненный нами оператор. Соединение таблиц задается в предложении `FROM`:

```
FROM STAFF
  LEFT OUTER JOIN PEOPLE
    ON STAFF.CODPEOPLE = PEOPLE.COD
```

Ключевое слово `JOIN` задает вид соединения и присоединяемую таблицу. Здесь мы используем левое внешнее соединение (`LEFT OUTER JOIN`) таблицы `STAFF` с таблицей `PEOPLE`.

Условие соединения задается после ключевого слова `ON`. "Главной" таблицей здесь, в левом соединении, является таблица `STAFF`. К каждой выбранной строке этой таблицы (выбираемые строки главной таблицы определяются как обычно, в предложении `WHERE`) присоединяются данные из таблицы `PEOPLE`, которые удовлетворяют условию в предложении `ON`. В нашем случае требуется равенство значения столбца `CODPEOPLE` из таблицы `STAFF`, который является внешним ключом, значению столбца `COD` из таблицы `PEOPLE`, который является первичным ключом; на него и ссылается внешний ключ таблицы `STAFF`. На самом деле не существует требования, чтобы в условиях соединения задавались только внешние и первичные (или уникальные) ключи.

Поскольку в нашем операторе присутствует более одной таблицы, мы используем для столбцов уточнения, задавая перед именем столбца имя таблицы, псевдоним или алиас — см. дальше. Конкретно в этом операторе, чтобы

избежать двусмысленности, мы обязаны использовать уточняющее имя для столбца с именем COD, т. к. это имя встречается в обеих таблицах.

Стоит сказать пару слов о списке выбора. Здесь в первом элементе мы используем операцию конкатенации нескольких строк. Мы соединяем фамилию, имя и отчество человека, полученные из таблицы PEOPLE, добавив между ними пробелы.

Коль скоро в таблице людей существует подходящий вычисляемый столбец, то мы можем записать предыдущий оператор и в следующем виде:

```
SELECT PEOPLE.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
    FULL OUTER JOIN PEOPLE
      ON STAFF.CODPEOPLE = PEOPLE.COD
  WHERE CODORG = 11
  ORDER BY 1;
```

При создании достаточно сложных операторов SELECT бывает утомительным набирать перед именами столбцов полные имена таблиц, особенно если эти имена достаточно длинные. По этой причине для нас, ленивых, существует возможность задавать псевдонимы (или алиасы, alias) для имен таблиц. Сравните следующий оператор:

```
SELECT P.NAME3 || ' ' ||
       P.NAME1 || ' ' ||
       P.NAME2 AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF S
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  WHERE CODORG = 11 AND
        SALARY <= (SELECT AVG(SALARY)
                   FROM STAFF
                   WHERE CODORG = 11)
  ORDER BY 1;
```

В предложении FROM мы для таблицы STAFF задали псевдоним S, а для таблицы PEOPLE — псевдоним P. Эти псевдонимы мы используем в списке выбора и в условии соединения. Однако во внутреннем операторе SELECT мы не можем использовать указанные во внешнем операторе SELECT псевдонимы, а всегда задаем полное имя таблицы.

Особенностью внешних соединений является то, что в выходной набор данных попадают и те строки, которые содержат пустые значения (NULL) в тех столбцах главной таблицы, которые присутствуют в условии соединения в предложении ON.

Посмотрите листинг 7.6. Там представлен список сотрудников организации с кодом 18. В списке присутствует три строки, в которых код сотрудника имеет пустое значение. Выполним для этой организации левое внешнее соединение таблицы STAFF (сотрудники) с таблицей PEOPLE (люди):

```
SELECT P.NAME3 || ' ' ||
       P.NAME1 || ' ' ||
       P.NAME2 AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF S
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  WHERE CODORG = 18
  ORDER BY 1;
```

Результат выполнения соединения — листинг 7.27.

Листинг 7.27. Левое внешнее соединение сотрудников и людей для организации 18

```
Сотрудник                 Должность                        Оклад
========================= ================================ =========
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ Исполнительный директор           38500.00
<null>                    Неизвестная должность1           <null>
<null>                    Неизвестная должность2           <null>
<null>                    Неизвестная должность3           <null>
```

Мы видим, что действительно в список попадают и строки, в которых столбцы, участвующие в условии соединения, имеют пустое значение. Забегая немного вперед, скажу, что при внутреннем соединении строки с пустыми значениями соответствующих столбцов игнорируются.

Напомню, что такой порядок мы получим при использовании Firebird 1.5. В Firebird 2.0 и InterBase 2007 пустые значения в столбцах, по которым проводится упорядочивание списка, будут находиться в начале списка.

Правое внешнее соединение

Правое внешнее соединение является зеркальным отражением левого внешнего соединения. При нем вначале отбираются все строки соединяемой таблицы (здесь она становится главной) на основании условий предложения

WHERE, затем к ним добавляются строки второй таблицы, указанной сразу после ключевого слова FROM с учетом условий, заданных в предложении ON.

Для иллюстрации этой зеркальности выполните оператор:

```
SELECT P.NAME3 || ' ' ||
       P.NAME1 || ' ' ||
       P.NAME2 AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM PEOPLE P
    RIGHT OUTER JOIN STAFF S
      ON S.CODPEOPLE = P.COD
  WHERE CODORG = 11 AND
        SALARY <= (SELECT AVG(SALARY)
                   FROM STAFF
                   WHERE CODORG = 11)
  ORDER BY 1;
```

Вы получите точно такой же список, как и в листинге 7.26.

Полное внешнее соединение

В случае полного внешнего соединения выбираются все соответствующие условию в предложении WHERE строки как левой, так и правой таблиц. Затем между ними устанавливается соответствие, заданное в предложении ON.

Проведем такое исследование. Вначале найдем все строки таблиц STAFF и PEOPLE. Потом выполним их полное соединение.

Выполним оператор

```
SELECT COD,
       CODPEOPLE
  FROM STAFF
  ORDER BY 1;
```

Получим 19 записей. Теперь выполним:

```
SELECT FULLNAME
  FROM PEOPLE
  ORDER BY 1;
```

Получим 112 записей. Наконец выполним полное внешнее соединение этих двух таблиц, задав условием соединения то же равенство кодов, что и в предыдущих операторах соединения.

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
```

```
  FROM STAFF S
    FULL OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

В результате мы получаем 118 записей. Такой результат несколько удивляет. Не слишком задумываясь, можно было бы предположить, что при полном внешнем объединении мы должны были бы получить количество записей либо равное количеству записей в большей соединяемой таблице, т. е. 112, либо равное, в худшем случае, сумме строк обеих таблиц — 131.

На самом деле все работает правильно. Вначале выполняется левое соединение таблицы `STAFF` с таблицей `PEOPLE`.

Посмотрите, какой моделируется оператор:

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF S
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Выполните такой оператор и посмотрите, что получится — листинг 7.28.

Листинг 7.28. Первый этап полного внешнего соединения

```
Сотрудник                    Должность                        Оклад
============================ ================================ =========
ЕРМИШИН АНТОН ИВАНОВИЧ       Системный администратор           31400.00
ЕРМИШИН АРТЕМ ИВАНОВИЧ       Программист                       23600.00
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ      Ночной сторож                      1500.00
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ      Исполнительный директор           38500.00
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ      Генеральный директор              56000.00
ЛОПАТНИКОВ ВАСИЛИЙ ГЕННАДИЕВИЧ Программист                     23900.00
ПАНКОВ ИГОРЬ АЛЕКСАНДРОВИЧ   Начальник отдела                  12000.00
ПАНКОВ РОМАН ИГОРЕВИЧ        Программист                       23700.00
ПАНКОВА ГАЛИНА АНАТОЛЬЕВНА   Заместитель начальника отдела     11000.00
ПАРШИНА АНТОНИНА ПЕТРОВНА    Техник                             7300.00
ПАРШИНА АНТОНИНА ПЕТРОВНА    Служба безопасности               12000.00
ПИНТЕРА АНДРЕЙ НИКОЛАЕВИЧ    Главный бухгалтер                 15000.00
ПИНТЕРА НИКОЛАЙ НИКОЛАЕВИЧ   Заместитель директора             42000.00
ПИНТЕРА ЮЛИЯ НИКОЛАЕВНА      Финансовый директор               12000.00
ЦВЕТКОВА ЕКАТЕРИНА АЛЕКСЕЕВНА Программист                      23800.00
ШАФРАН НИНА СЕРГЕЕВНА        Заместитель директора по кадрам   13200.00
<null>                       Неизвестная должность1           <null>
<null>                       Неизвестная должность2           <null>
<null>                       Неизвестная должность3           <null>
```

Результатом будет 19 строк, однако, два человека присутствуют здесь несколько раз. ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ представлен три раза (то есть на два раза больше, чем он существует в природе), ПАРШИНА АНТОНИНА ПЕТРОВНА представлена два раза (на один раз больше). Кроме того, в списке есть три сотрудника, не имеющих кодов. В сумме получаем тот самый перебор в шесть строк.

Кстати, несколько позже мы опять вернемся к этому листингу и выполним очень интересное его преобразование. Не заглядывайте сейчас в *разд. 7.5*.

После этого выполняется правое внешнее соединение тех строк таблицы PEOPLE, которые не вошли в левое соединение, с таблицей STAFF.

Общий итог — 112 записей таблицы PEOPLE плюс шесть "лишних" строк таблицы STAFF дают те самые 118 строк полного внешнего соединения.

Я в реальной жизни чаще всего использовал левое внешнее соединение. Давайте рассмотрим еще несколько примеров левого внешнего соединения.

7.3.2. Более сложные примеры соединений

Количество выполняемых соединений таблиц в одном операторе SELECT не имеет физических ограничений, однако в документации по InterBase рекомендуется использовать их не более 16.

Двойное соединение

Рассмотрим пример двойного внешнего соединения, т. е. тот случай, когда к первой таблице присоединяется не одна, а уже две таблицы. Мы только что отображали список сотрудников всех организаций. Там все было замечательно, мы получили фамилии, имена, отчества и должности людей, только мы не видели, в какой организации работает человек. Чтобы при отображении сотрудников видеть и их фамилии с именами и отчествами, и организации, в которых они работают, нужно выполнить два левых внешних соединения таблицы STAFF с таблицей PEOPLE и с таблицей ORGANIZATION.

Введите и выполните оператор:

```
SELECT P.FULLNAME AS "Сотрудник",
       S.DUTIES AS "Должность",
       O.NAME AS "Организация"
  FROM STAFF S
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
  ORDER BY 1;
```

Здесь выбираются все сотрудники всех организаций из таблицы STAFF. В выбранные строки добавляются полные имена сотрудников из таблицы PEOPLE. Условием соединения является равенство кода в таблице PEOPLE коду человека в таблице STAFF. После чего к этим строкам добавляются названия организаций из таблицы ORGANIZATION. Здесь условием соединения является равенство кода, первичного ключа таблицы ORGANIZATION коду организации в начальной, "главной", таблице STAFF.

> ***Замечание***
>
> Последовательность выполняемых действий при соединении таблиц вовсе не должна соответствовать описанной только что последовательности. Конкретный порядок действий определяет сервер базы данных.

В результате выполнения двойного соединения мы получим те же 19 строк (листинг 7.29).

Листинг 7.29. Двойное левое внешнее соединение

```
Сотрудник                      Должность                      Организация
============================== ============================== ==============
ЕРМИШИН АНТОН ИВАНОВИЧ         Системный администратор         ЗАО "ИнфоТел"
ЕРМИШИН АРТЕМ ИВАНОВИЧ         Программист                    ЗАО "ИнфоТел"
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ        Ночной сторож                  БАЛЧУГ
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ        Исполнительный директор         ЗАО "ИнфоТел"
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ        Генеральный директор           АЛЬФА-БАНК
ЛОПАТНИКОВ ВАСИЛИЙ ГЕННАДИЕВИЧ Программист                    ЗАО "ИнфоТел"
ПАНКОВ ИГОРЬ АЛЕКСАНДРОВИЧ     Начальник отдела               ЗАО "ИнфоТел"
ПАНКОВ РОМАН ИГОРЕВИЧ          Программист                    ЗАО "ИнфоТел"
ПАНКОВА ГАЛИНА АНАТОЛЬЕВНА     Заместитель начальника отдела  ЗАО "ИнфоТел"
ПАРШИНА АНТОНИНА ПЕТРОВНА      Техник                         ЗАО "ИнфоТел"
ПАРШИНА АНТОНИНА ПЕТРОВНА      Служба безопасности            ЗАО "ИнфоТел"
ПИНТЕРА АНДРЕЙ НИКОЛАЕВИЧ      Главный бухгалтер              ЗАО "ИнфоТел"
ПИНТЕРА НИКОЛАЙ НИКОЛАЕВИЧ     Заместитель директора          ЗАО "ИнфоТел"
ПИНТЕРА ЮЛИЯ НИКОЛАЕВНА        Финансовый директор            ЗАО "ИнфоТел"
ЦВЕТКОВА ЕКАТЕРИНА АЛЕКСЕЕВНА  Программист                    ЗАО "ИнфоТел"
ШАФРАН НИНА СЕРГЕЕВНА          Заместитель директора по кадрам ЗАО "ИнфоТел"
<null>                         Неизвестная должность1         БАЛЧУГ
<null>                         Неизвестная должность2         БАЛЧУГ
<null>                         Неизвестная должность3         БАЛЧУГ
```

Необходимо отметить, что порядок указания соединяемых таблиц не имеет значения. Здесь важно задать первую, "главную", таблицу сразу после ключевого слова FROM, все остальные соединяемые таблицы "подключаются" к строкам, выбранным именно из этой таблицы, и ее строки выбираются на основании условия в предложении WHERE.

Для иллюстрации этого изменим предыдущий оператор SELECT, поменяв местами соединяемые таблицы. Выполните оператор:

```
SELECT P.FULLNAME AS "Сотрудник",
       S.DUTIES AS "Должность",
       O.NAME AS "Организация"
  FROM STAFF S
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Вы получите точно такой же список, что и представленный в листинге 7.29.

Чтобы убрать последние три строки, имеющие пустые значения кода сотрудника, нужно в предыдущий оператор добавить предложение WHERE, удаляющее из списка строки с пустыми значениями этого кода:

```
SELECT P.FULLNAME AS "Сотрудник",
       S.DUTIES AS "Должность",
       O.NAME AS "Организация"
  FROM STAFF S
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  WHERE S.CODPEOPLE IS NOT NULL
  ORDER BY 1;
```

Теперь список будет содержать 16 непустых строк.

К этому результату мы опять вернемся чуть позже, когда будем рассматривать варианты внутреннего соединения.

Еще один похожий пример соединения. Мы говорили, что в нашей базе данных есть одна нехорошая семейка. Давайте в таблице PEOPLEADMIN найдем все нарушения закона всеми людьми. При этом выполним двойное левое внешнее соединение с таблицами REFADMIN и PEOPLE, чтобы сразу получить осмысленный результат.

```
SELECT P.FULLNAME AS "Человек",
       R.NAME AS "Преступление",
       A.DATEADMIN "Дата"
  FROM PEOPLEADMIN A
    LEFT OUTER JOIN REFADMIN R
      ON A.CODADMIN = R.COD
    LEFT OUTER JOIN PEOPLE P
```

```
      ON A.CODPEOPLE = P.COD
  ORDER BY 1;
```

Мне очень неловко за этих людей, они все из одной семьи. Считаю, таким не место в нашем обществе. Их поведение настолько безобразное, что я даже постесняюсь показать в этой приличной книге результат нашей выборки. Если вам интересно, выполните оператор сами и посмотрите, как не надо себя вести.

Рефлексивное соединение, или самосоединение

Соединяемые таблицы не обязательно должны отличаться от главной таблицы в операторе `SELECT`. Если таблица соединяется сама с собой, то такое соединение называется рефлексивным или самосоединением. Есть еще один термин для такого соединения — реентерабельное.

Рассмотрим следующий пример. Пусть мы собираемся получить список людей, куда добавляются и фамилии их супругов. Выполните оператор:

```
SELECT PM.FULLNAME AS "Фамилия, имя, отчество",
       PH.NAME3 AS "Супруг / супруга"
  FROM PEOPLE PM
    LEFT OUTER JOIN PEOPLE PH
      ON PM.CODOTHERHALF = PH.COD
  ORDER BY PM.FULLNAME;
```

> ***Замечание***
>
> Обратите внимание на то, что если в других случаях применения оператора `SELECT`, когда в выборке присутствуют *разные* таблицы, мы иногда можем и не использовать псевдонимы таблиц, то в данном случае мы *обязаны* для таблиц указывать псевдонимы, чтобы точно указывать в операторе, к какой именно таблице относится тот или иной столбец.

Результатом выполнения оператора будет список, содержащий все 112 строк таблицы `PEOPLE`.

Подкорректируем немного наш оператор, задав устранение из списка тех людей, для которых не указан код супруга.

```
SELECT PM.FULLNAME AS "Фамилия, имя, отчество",
       PH.NAME3 AS "Супруг / супруга"
  FROM PEOPLE PM
    LEFT OUTER JOIN PEOPLE PH
      ON PM.CODOTHERHALF = PH.COD
  WHERE PM.CODOTHERHALF IS NOT NULL
  ORDER BY PM.FULLNAME;
```

При выполнении этого оператора мы получаем уже 72 строки (листинг 7.30).

Листинг 7.30. Выборка людей и их супругов

```
Фамилия, имя, отчество         Супруг / супруга
============================== ================
БРЮКОВ АЛЕКСАНДР АЛЕКСЕЕВИЧ    МОРОЗОВА
БРЮКОВ ИГОРЬ АЛЕКСАНДРОВИЧ     ЗАБРОДИНА
ВАЛЕНТИНОВ ГЕННАДИЙ ПАВЛОВИЧ   ПАКИНА
ВОЛОСОВА ВАСИЛИСА АНДРЕЕВНА    ЛОПАТНИКОВ
ГАЛКИН ВАЛЕРИЙ СЕРГЕЕВИЧ       ГАЛКИНА
ГАЛКИНА ЕКАТЕРИНА ВАЛЕРЬЕВНА   ШИГАЕВ
ГАЛКИНА ИРИНА ВАЛЕРЬЕВНА       ГАЛКИН
ЕРМИШИН АНТОН ИВАНОВИЧ         МУРАВЬЕВА
ЕРМИШИН АРТЕМ ИВАНОВИЧ         ПАРШИНА
ЕРМИШИН ИВАН ПЕТРОВИЧ          СЕРЕБРЯКОВА
...
```

Теперь сформируем оператор, в котором с таблицей PEOPLE трижды будет соединяться эта же самая таблица. Выберем из нашей базы данных тех людей, для которых указаны супруги, мать и отец. Выполним необходимое тройное соединение. Устраним в нем строки с пустыми кодами.

```
SELECT PG.FULLNAME AS "Фамилия, имя, отчество",
       PH.NAME3 AS "Супруг / супруга",
       PM.NAME3 AS "Мать",
       PF.NAME3 AS "Отец"
  FROM PEOPLE PG                      /* Главная таблица */
    LEFT OUTER JOIN PEOPLE PH         /* Супруг */
       ON PG.CODOTHERHALF = PH.COD
    LEFT OUTER JOIN PEOPLE PM         /* Мать */
       ON PG.CODMOTHER = PM.COD
    LEFT OUTER JOIN PEOPLE PF         /* Отец */
       ON PG.CODFATHER = PF.COD
  WHERE PG.CODOTHERHALF IS NOT NULL AND
        PG.CODMOTHER IS NOT NULL AND
        PG.CODFATHER IS NOT NULL
  ORDER BY PG.FULLNAME;
```

Оператор вернет 24 записи (листинг 7.31).

Листинг 7.31. Выборка людей, их супругов и родителей

```
Фамилия, имя, отчество         Супруг / супруга Мать          Отец
============================== ================ ============ ==========
БРЮКОВ ИГОРЬ АЛЕКСАНДРОВИЧ     ЗАБРОДИНА        МОРОЗОВА     БРЮКОВ
ВОЛОСОВА ВАСИЛИСА АНДРЕЕВНА    ЛОПАТНИКОВ       МИТРОФАНОВА  ВОЛОСОВ
ГАЛКИНА ЕКАТЕРИНА ВАЛЕРЬЕВНА   ШИГАЕВ           ГАЛКИНА      ГАЛКИН
```

```
ЕРМИШИН АНТОН ИВАНОВИЧ          МУРАВЬЕВА       СЕРЕБРЯКОВА  ЕРМИШИН
ЕРМИШИН АРТЕМ ИВАНОВИЧ          ПАРШИНА         СЕРЕБРЯКОВА  ЕРМИШИН
ЗАБРОДИНА ИРИНА НИКОЛАЕВНА      БРЮКОВ          ФРОЛОВА      ЗАБРОДИН
ЗАЙЦЕВА ДАРЬЯ ИГОРЬЕВНА         ЦВЕТКОВ         ЗАЙЦЕВА      ЗАЙЦЕВ
ЛЕВИН ВЛАДИМИР АЛЕКСАНДРОВИЧ    ЛЕВИНА          ЛЕВИНА       ЛЕВИН
ЛЕВИНА ВИКТОРИЯ МИХАЙЛОВНА      ЛЕВИН           ПУШКИНА      ПУШКИН
...
```

***

Мы весьма подробно рассмотрели возможности внешних соединений. Настала пора перейти к внутренним соединениям таблиц.

7.3.3. Внутреннее соединение

Внутреннее соединение похоже на внешнее с той разницей, что если во внешнем соединении в список попадают все строки главной таблицы, соответствующие условию поиска в предложении `WHERE`, то во внутреннем соединении список содержит обычно чуть меньше строк, а именно те строки, для которых в точности выполняется условие соединения (условие в предложении `ON`).

Еще раз выполните оператор:

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF S
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Вы получите список из 19 записей (см. как пример листинг 7.28), где три последние записи имеют пустое значение в столбце, указанном в условии соединения (напомню — это для Firebird 1.5). Если из данного оператора поиска простой подстановкой и с легким сокращением сформировать внутреннее соединение, то получится следующий оператор:

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность"
  FROM STAFF S
    INNER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Результатом здесь будет следующее — листинг 7.32.

Листинг 7.32. Внутреннее соединение

```
Сотрудник                      Должность
============================== ==============================
ЕРМИШИН АНТОН ИВАНОВИЧ         Системный администратор
ЕРМИШИН АРТЕМ ИВАНОВИЧ         Программист
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ        Ночной сторож
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ        Исполнительный директор
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ        Генеральный директор
ЛОПАТНИКОВ ВАСИЛИЙ ГЕННАДИЕВИЧ Программист
ПАНКОВ ИГОРЬ АЛЕКСАНДРОВИЧ    Начальник отдела
ПАНКОВ РОМАН ИГОРЕВИЧ          Программист
ПАНКОВА ГАЛИНА АНАТОЛЬЕВНА     Заместитель начальника отдела
ПАРШИНА АНТОНИНА ПЕТРОВНА      Техник
ПАРШИНА АНТОНИНА ПЕТРОВНА      Служба безопасности
ПИНТЕРА АНДРЕЙ НИКОЛАЕВИЧ      Главный бухгалтер
ПИНТЕРА НИКОЛАЙ НИКОЛАЕВИЧ     Заместитель директора
ПИНТЕРА ЮЛИЯ НИКОЛАЕВНА        Финансовый директор
ЦВЕТКОВА ЕКАТЕРИНА АЛЕКСЕЕВНА  Программист
ШАФРАН НИНА СЕРГЕЕВНА          Заместитель директора по кадрам
```

Этот список содержит 16 строк, в нем отсутствуют записи, для которых нет точного соответствия условиям соединения.

Для внутренних соединений не существует ни левых, ни правых вариантов, поскольку в результирующий список попадают только те строки, которые в точности соответствуют условию соединения.

Проверим это. Выполним оператор, зеркальный предыдущему оператору:

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность"
  FROM PEOPLE P
    INNER JOIN STAFF S
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Результат, как вы видите, точно такой же, как и в листинге 7.32.

Аналогичным образом можно выполнять многократные внутренние соединения. Например, чтобы получить данные из листинга 7.29, но без пустых строк, нужно выполнить оператор:

```
SELECT P.FULLNAME AS "Сотрудник",
       S.DUTIES AS "Должность",
       O.NAME AS "Организация"
  FROM STAFF S
```

```
    INNER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
    INNER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

В листинге 7.31 отображен список, полученный в результате выполнения тройного левого внешнего объединения:

```
SELECT PG.FULLNAME AS "Фамилия, имя, отчество",
       PH.NAME3 AS "Супруг / супруга",
       PM.NAME3 AS "Мать",
       PF.NAME3 AS "Отец"
  FROM PEOPLE PG                      /* Главная таблица */
    LEFT OUTER JOIN PEOPLE PH         /* Супруг */
       ON PG.CODOTHERHALF = PH.COD
    LEFT OUTER JOIN PEOPLE PM         /* Мать */
       ON PG.CODMOTHER = PM.COD
    LEFT OUTER JOIN PEOPLE PF         /* Отец */
       ON PG.CODFATHER = PF.COD
  WHERE PG.CODOTHERHALF IS NOT NULL AND
        PG.CODMOTHER IS NOT NULL AND
        PG.CODFATHER IS NOT NULL
  ORDER BY PG.FULLNAME;
```

Создадим на базе этого оператора оператор внутреннего соединения, убрав к тому же ставшее уже ненужным требование отсутствия пустого значения.

```
SELECT PG.FULLNAME AS "Фамилия, имя, отчество",
       PH.NAME3 AS "Супруг / супруга",
       PM.NAME3 AS "Мать",
       PF.NAME3 AS "Отец"
  FROM PEOPLE PG                /* Главная таблица */
    INNER JOIN PEOPLE PH        /* Супруг */
       ON PG.CODOTHERHALF = PH.COD
    INNER JOIN PEOPLE PM        /* Мать */
       ON PG.CODMOTHER = PM.COD
    INNER JOIN PEOPLE PF        /* Отец */
       ON PG.CODFATHER = PF.COD
  ORDER BY PG.FULLNAME;
```

Мы получили точно такой же список, только с чуть меньшими интеллектуальными затратами и меньшим ручным вводом операторов. Относительно того, насколько новый вариант эффективней предыдущего, мне трудно судить. Эффективность зависит в основном от реализации конкретной системы управления базами данных, которую мы используем. К тому же, увеличение

или уменьшение быстродействия при выполнении операций поиска (при одной и той же структуре базы данных) сильно зависит от объема хранимых данных и быстродействия сетевых ресурсов.

Свои слова, что внешнее соединение используется чаще всего, я забираю назад. Теперь буду пользоваться только (или большей частью) внутренним соединением.

> ***Замечание***
>
> Разумеется, для каждого средства, существующего в SQL, есть своя сфера эффективного применения. В своей программистской деятельности вы, скорее всего, будете использовать и внешние, и внутренние соединения таблиц. Главное — помнить, что в любом случае вы имеете право применять адекватные средства, которые позволят с меньшими усилиями, с высокой эффективностью и довольно наглядно (не забывайте и об этом!) получить нужный вам результат.

Теперь (важный момент!) выполните предыдущий оператор `SELECT`, но убрав ключевое слово `AS` из последних трех элементов списка выбора:

```
SELECT PG.FULLNAME AS "Фамилия, имя, отчество",
       PH.NAME3,
       PM.NAME3,
       PF.NAME3
  FROM PEOPLE PG                  /* Главная таблица */
    INNER JOIN PEOPLE PH          /* Супруг */
       ON PG.CODOTHERHALF = PH.COD
    INNER JOIN PEOPLE PM          /* Мать */
       ON PG.CODMOTHER = PM.COD
    INNER JOIN PEOPLE PF          /* Отец */
       ON PG.CODFATHER = PF.COD
  ORDER BY PG.FULLNAME;
```

Посмотрите на заголовок отображения полученного списка:

```
Фамилия, имя, отчество         Name3           Name31       Name32
============================== =============== ============ ==========
```

Мы видим, что система управления базами данных автоматически присваивает специфические имена столбцам, которые повторяют уже существующие столбцы, но из соединяемых таблиц — `Name31`, `Name32`. Этот факт мы используем в *главе 10*, когда будем писать для нашего пользователя программу, использующую такой или подобный вид соединения таблиц.

7.3.4. Замечания по синтаксису

Хочу обратить ваше внимание на то, что синтаксис соединений, используемый в системах InerBase и Firebird, очень естественный и понятный, чего не скажешь о некоторых других реляционных СУБД.

Вообще говоря, в нашем синтаксисе существует и другой вариант написания соединений, когда соединяемые таблицы перечисляются через запятую в предложении FROM.

Я не рекомендую вам использовать этот вариант. Тем не менее, рассмотрим один пример.

Посмотрите на листинг 7.32. Это результат выполнения оператора

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность"
  FROM STAFF S
    INNER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Здесь осуществляется внутреннее соединение таблицы сотрудников STAFF и таблицы людей PEOPLE.

Оператор, выполняющий в точности те же самые действия, можно записать и в виде, когда соединяемые таблицы перечисляются в предложении FROM, а условия соединения задаются в предложении WHERE:

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность"
  FROM STAFF S, PEOPLE P
  WHERE S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Соединение, при котором все таблицы записываются в предложении FROM, является внутренним соединением. Условие же соединения добавляется к предложению WHERE, которое содержит также и условия поиска (в нашем примере условия поиска отсутствуют).

Здесь возникает два неприятных момента. Один связан с потерей той самой наглядности синтаксической конструкции, о которой мы сказали три минуты назад. Вторая неприятность может возникнуть в случае достаточно сложного условия в предложении WHERE, когда трудно будет определить, к какой части оператора относится условие — к выбираемым строкам или к условию соединения.

Может быть и не так страшно, что нам сложно понять, что к чему относится (хотя лично я жутко не люблю не очень понятные конструкции). Хуже, когда полученное условие будет неправильно интерпретировано сервером базы данных. В лучшем случае будет выдано сообщение об ошибке, а может получиться и так, что результат выборки никак не будет соответствовать нашим потребностям.

Имеет смысл воздержаться от использования этого синтаксиса. Не известно, будет ли подобный синтаксис поддерживаться в следующих версиях СУБД.

7.4. Группировка результатов выборки

Мы с вами уже рассмотрели довольно интересные и сложные варианты выборки данных, используемые в реальной практической деятельности. Есть еще интересные и полезные средства в операторе `SELECT` — использование возможностей группировки результатов поиска при применении предложений `GROUP BY` и `HAVING`.

Прежде чем обратиться к средствам группировки выборки, еще раз посмотрим на так называемые агрегатные функции, которые позволяют получить некоторые полезные данные не из одной строки, а из группы строк таблицы базы данных. Агрегатные функции представлены в табл. 7.1.

Таблица 7.1. *Список используемых агрегатных функций*

Функция	Выполняемые действия
`AVG`	Возвращает среднее значение данных числовых столбцов
`COUNT`	Подсчитывает количество строк, удовлетворяющих заданному условию
`MIN`	Находит минимальное значение столбца в группе строк
`MAX`	Находит максимальное значение столбца в группе строк
`SUM`	Суммирует числовые значения

Функции `MIN` и `MAX`, в отличие от функций `AVG` и `SUM`, могут использоваться не только с числовыми столбцами. Они применимы к любым типам данных, кроме `BLOB`. Чтобы существовала возможность применения этих функций к конкретному типу данных, для него должны быть определены операции отношения > и <. Выполним, например, следующий оператор, который отыскивает человека с "максимальной" фамилией, т. е. с фамилией, являющейся последней в упорядоченном по алфавиту списке фамилий:

```
SELECT MAX(NAME3)
  FROM PEOPLE;
```

Мы получим фамилию `ШИГАЕВА`. Чтобы убедиться, что это последняя фамилия в упорядоченном списке, выполним следующий оператор:

```
SELECT NAME3
  FROM PEOPLE
  ORDER BY NAME3;
```

Действительно, в конце списка находятся две фамилии `ШИГАЕВА`.

Аналогичным образом работает и функция MIN, в смысле с точностью до наоборот. Можете это также проверить.

Большинство из этих функций мы с вами уже рассматривали достаточно подробно. Сейчас посмотрим на них с точки зрения группировки строк.

Группировка является обязательной, если в операторе SELECT вы используете и агрегатные функции, и обычные столбцы. При этом группировка должна выполняться по всем неагрегатным столбцам.

Существует **два основных правила группировки**.

- Каждый столбец, включенный в неагрегатный список оператора SELECT, должен появиться в предложении GROUP BY.
- Оператор SELECT может содержать только одно предложение GROUP BY.

Это важнейшие правила группировки. Мы имеем право их помнить и использовать в любом нашем операторе, выполняющем группировку. Иначе получим ошибку, борьба с которой может занять слишком много нашего драгоценного времени. Пример подобной неприятной ситуации мы рассмотрим через несколько минут.

> ***Замечание***
>
> В документации по InterBase можно найти еще и третье правило, которое требует, чтобы в предложении GROUP BY присутствовали имена только тех столбцов, которые указаны в списке выбора оператора SELECT. Это не соответствует действительности. Хотя присутствие таких столбцов в этом предложении и не кажется достаточно разумным.

Предложения GROUP BY и HAVING располагаются после предложения WHERE и перед предложением ORDER BY. Синтаксис:

```
[GROUP BY <список столбцов> [HAVING <условия поиска>]]
```

При выполнении такого оператора SELECT делается все то же самое, что и при обычном SELECT — выбираются указанные столбцы строк, соответствующие условию в предложении WHERE, вычисляются значения агрегатных функций. После этого выполняется группировка по столбцам, перечисленным в предложении GROUP BY. Затем осуществляется дополнительная "фильтрация" строк на основании условия в предложении HAVING.

Рассмотрим небольшое количество примеров использования средств группировки результатов запроса.

Мы хотим на законных основаниях получить сведения конфиденциального характера об обобщенных характеристиках окладов сотрудников всех организаций, представленных в нашей базе данных. Выполните следующий оператор:

```
SELECT AVG(SALARY) AS "Среднее",
```

```
       MAX(SALARY) AS "Максимум",
       MIN(SALARY) AS "Минимум",
       COUNT(COD) AS "Количество",
       CODORG AS "Организация"
  FROM STAFF
  GROUP BY CODORG
  ORDER BY 4;
```

В операторе мы используем четыре агрегатные функции для получения среднего значения заработной платы всех сотрудников, представленных в базе данных, максимального и минимального значения оклада и общего количества сотрудников в каждой организации, попавшей в нашу выборку данных. В функции `COUNT` в этом случае можно указать любой столбец таблицы `STAFF` или задать символ `*`. Это ни на что не повлияет.

Группировку мы выполняем по столбцу `CODORG`, что и требуется по правилам группировки, поскольку только этот столбец является неагрегатным в списке выбора нашего оператора.

Результат выполнения оператора представлен в листинге 7.33.

Листинг 7.33. Обобщенные сведения об окладах сотрудников организаций

```
Среднее    Максимум   Минимум    Количество Организация
========== ========== ========== ========== ==========
   6750.00   12000.00    1500.00          2         16
  38500.00   38500.00   38500.00          4         18
  22684.61   56000.00    7300.00         13         11
```

Мы получили все те данные, которые нам были нужны.

В процессе обработки запроса система выполняет такие действия.

1. Отбираются те строки, которые соответствуют условию поиска в предложении `WHERE`, если таковое имеется. В нашем примере это предложение отсутствует.

14. Выбранные строки объединяются в группы на основании значения кода организации (`CODORG`).

15. Для каждой группы рассчитываются значения агрегатных функций. При этом для каждой группы создается одна результирующая строка.

16. Полученные в результате строки упорядочиваются в соответствии с предложением `ORDER BY` (в нашем случае — по количеству строк в группе).

При применении этих средств мы, разумеется, можем использовать и другие возможности оператора `SELECT`. В полученном нами списке есть один стол-

бец, который не имеет особого смысла. Это код организации. Выполним в рамках данного оператора еще и операцию соединения, чтобы вместо кода организации получить осмысленное название этой организации.

В первом приближении, как мы могли бы в спешке предположить, в список выбора нашего оператора SELECT нужно только внести одно простое изменение (заменить CODORG на O.NAME), и он должен быть, как нам вначале показалось, следующим:

```
SELECT AVG(SALARY) AS "Среднее",
       MAX(SALARY) AS "Максимум",
       MIN(SALARY) AS "Минимум",
       COUNT(COD) AS "Количество",
       O.NAME AS "Организация"
  FROM STAFF S
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
  GROUP BY CODORG
  ORDER BY 4;
```

Однако это не работает. Нам сообщают, что существует столбец в списке выбора, который следовало бы использовать в предложении GROUP BY. Рекомендую еще раз посмотреть на два основных правила группировки.

Здесь, действительно, нужно ввести в предложение GROUP BY столбец, указанный в списке выбора как O.NAME. Столбец же CODORG в этом предложении совсем не нужен, хотя его присутствие, как выяснилось, ничему не мешает. Правильный оператор выборки данных будет таким:

```
SELECT AVG(SALARY) AS "Среднее",
       MAX(SALARY) AS "Максимум",
       MIN(SALARY) AS "Минимум",
       COUNT(*) AS "Количество",
       O.NAME AS "Организация"
  FROM STAFF S
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
  GROUP BY O.NAME
  ORDER BY 4;
```

Мы получаем, наконец, нужный результат — листинг 7.34.

Листинг 7.34. Улучшенный результат получения конфиденциальных данных

```
Среднее    Максимум   Минимум    Количество Организация
========== ========== ========== ========== ==========
   6750.00   12000.00    1500.00          2  АЛЬФА-БАНК
  38500.00   38500.00   38500.00          4  БАЛЧУГ
  22684.61   56000.00    7300.00         13  ЗАО "ИнфоТел"
```

Теперь проиллюстрируем использование предложения `HAVING`. Для нашего примера зададим условие, что в результирующий список должны помещаться только те строки, где минимальный оклад выше 1500. Для этого изменим наш последний оператор выборки данных:

```
SELECT AVG(SALARY) AS "Среднее",
       MAX(SALARY) AS "Максимум",
       MIN(SALARY) AS "Минимум",
       COUNT(*) AS "Количество",
       O.NAME AS "Организация"
  FROM STAFF S
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
  GROUP BY O.NAME
  HAVING MIN(SALARY) > 1500
  ORDER BY 4;
```

После предложения `GROUP BY` мы добавили предложение

```
  HAVING MIN(SALARY) > 1500
```

Результат, естественно, сократится на одну строку (листинг 7.35).

Листинг 7.35. Использование предложения `HAVING`

```
Среднее    Максимум   Минимум    Количество Организация
========== ========== ========== ========== ==========
  38500.00   38500.00   38500.00          4  БАЛЧУГ
  22684.61   56000.00    7300.00         13  ЗАО "ИнфоТел"
```

Основным требованием к составу предложения `HAVING` является то, что имена столбцов в этом предложении обязательно должны присутствовать в списке `GROUP BY` или быть параметрами агрегатной функции.

Где-то я вычитал, что любой запрос, использующий предложение `HAVING`, может быть преобразован в другую форму без этого предложения. Я как-то не очень в это верю. Тренировки ради, попробуйте преобразовать предыдущий запрос в другой, не содержащий предложения `HAVING`. Мне же, по правде сказать, сейчас лень это делать.

7.5. Использование подзапросов в операторах SQL

Мы с вами очень часто используем внутренние, вложенные запросы или, как их еще называют, подзапросы в операторах поиска и манипулирования данными — в операторах `SELECT`, `INSERT`, `UPDATE` и `DELETE`. Настала пора несколько систематизировать наши знания о подзапросах и привести дополнительные примеры их использования.

В некоторых реляционных СУБД запрещено использовать предложение `ORDER BY` в подзапросах. У нас с вами это можно, хотя и не имеет никакого смысла.

Опять же некоторые СУБД в операциях сравнения допускают по непонятной причине использование подзапросов только в левой части таких операций. В наших системах управления базами данных такого ограничения нет. Что, с моей точки зрения, весьма естественно.

В подзапросе мы можем использовать имена столбцов таблицы подзапроса, однако нам не запрещено использование и имен столбцов таблиц внешнего по отношению к нашему запроса. Эту возможность мы с вами прямо сейчас и используем.

Есть один простой совет по синтаксису написания подзапросов. Вы не ошибетесь, если каждый внутренний оператор `SELECT` будете заключать в круглые скобки. В некоторых конструкциях это является обязательным, в других нет. Я всегда помещаю любой внутренний `SELECT` в скобки. Не люблю лишней головной боли.

Напомню некоторые ранее использованные варианты подзапросов.

Посмотрите в *главе 5* листинг 5.4. Там при добавлении новой строки для создания бесконечного цикла мы использовали подзапрос:

```
INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, SEX)
  SELECT (GEN_ID(GEN_PEOPLE, 1), NAME1, NAME2, NAME3, SEX FROM PEOPLE);
```

Бесконечные циклы — не лучшее применение наших с вами творческих способностей. Давайте лучше глянем на действительно полезный оператор добавления новой строки, представленный в листинге 5.7 той же главы:

```
INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, BIRTHDAY, SEX,
  CODMOTHER, CODFATHER)
  VALUES (GEN_ID(GEN_PEOPLE, 1), 'СЕРГЕЙ', 'НИКОЛАЕВИЧ', 'ПИНТЕРА',
   '13.12.1988', '0',
   (SELECT COD FROM PEOPLE
     WHERE NAME1 = 'ИРИНА' AND
           NAME2 = 'ИВАНОВНА' AND
```

```
        NAME3 = 'ПИНТЕРА'),
  (SELECT COD FROM PEOPLE
    WHERE NAME1 = 'НИКОЛАЙ' AND
          NAME2 = 'НИКОЛАЕВИЧ' AND
          NAME3 = 'ПИНТЕРА' AND
          BIRTHDAY = '01.01.1967'));
```

Здесь, вместо того чтобы распечатывать список людей, отыскивать там необходимых родителей и запоминать их коды, мы использовали вложенный оператор SELECT, в котором указали известные нам характеристики родителей человека.

В этом случае каждый оператор SELECT должен вернуть ровно одно значение одного столбца (это значение используется для формирования значения одного конкретного столбца).

В первом операторе нам было достаточно указать только имя, фамилию и отчество матери человека. Это позволило однозначно идентифицировать конкретную запись. Во втором операторе для выбора кода отца помимо этих значений нам пришлось указать еще и дату рождения человека, поскольку людей с таким же именем, отчеством и фамилией в базе данных оказалось двое.

В этой же главе еще один оператор добавления содержал три подзапроса, отыскивающих нужные коды людей:

```
INSERT INTO PEOPLE (COD, NAME1, NAME2, NAME3, BIRTHDAY, SEX, CODMOTHER,
CODFATHER, CODOTHERHALF)
  VALUES (GEN_ID(GEN_PEOPLE, 1), 'НАТАЛЬЯ', 'ИЛЬИНИЧНА', 'МУРАВЬЕВА',
'25.01.1986', '1',
  (SELECT COD FROM PEOPLE
     WHERE NAME1 = 'ЕКАТЕРИНА' AND
     NAME2 = 'АНДРЕЕВНА' AND
     NAME3 = 'ЧЕРНЫШОВА'),
  (SELECT COD FROM PEOPLE
     WHERE NAME1 = 'ИЛЬЯ' AND
     NAME2 = 'НИКОЛАЕВИЧ' AND
     NAME3 = 'МУРАВЬЕВ'),
  (SELECT COD FROM PEOPLE
     WHERE NAME1 = 'АНТОН' AND
     NAME2 = 'ИВАНОВИЧ' AND
     NAME3 = 'ЕРМИШИН'));
```

Ничего нового для нас с вами здесь нет. Все внутренние операторы SELECT помещены в скобки.

Это было просто воспоминание о том, как мы получали новые значения для операторов добавления данных. Аналогичные действия мы можем выполнять и для операторов UPDATE и DELETE.

Не менее интересные варианты подзапросов мы можем использовать и при поиске данных. Вспомните, как мы отыскивали код страны России:

```
SELECT CODREG  AS "Код региона",
       NAMEREG AS "Регион",
       CENTER  AS "Центр региона"
  FROM REFREG
  WHERE CODCTR = (SELECT CODCTR
                    FROM REFCTR
                    WHERE NAME = 'РОССИЯ')
  ORDER BY CENTER;
```

Вместо того чтобы задавать код, мы применяли оператор SELECT, который по названию страны отыскивал нам необходимое значение кода.

А сейчас я хочу привести пример использования нескольких вложенных операторов SELECT, которым я очень горжусь.

Посмотрите на листинг 7.28. Там показан возможный результат выполнения следующего оператора поиска:

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF S
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  ORDER BY 1;
```

Этот оператор возвращает список всех сотрудников всех организаций с указанием фамилии, имени и отчества сотрудника, его должности и оклада.

Можете сейчас выполнить этот оператор и получить те же 19 строк.

Мы видим, что некоторые люди в списке встречаются более одного раза. Как бы нам поступить, если мы хотим получить список, содержащий только тех людей, которые встречаются несколько раз, т. е. более одного раза? Специалист сразу скажет: нужно использовать хранимую процедуру, поскольку вот так, навскидку, простого решения не видно.

Однако я после нескольких бессонных ночей нашел решение. Оператор выборки данных содержит два вложенных один в другой подзапроса. Это не считая еще основного запроса.

Основной задачей являлась формулировка условия в предложении WHERE. Добавим это предложение в предыдущий оператор и выполним поиск.

```
SELECT P.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       O.NAME AS "Организация"
  FROM STAFF S
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
  WHERE S.COD IN
       (SELECT N.COD FROM STAFF N
          WHERE (SELECT COUNT(*)
                   FROM STAFF M
                   WHERE N.CODPEOPLE = M.CODPEOPLE) > 1)
  ORDER BY 1;
```

Мы получаем тот самый долго отыскиваемый результат, содержащий ровно пять нужных строк (листинг 7.36).

Листинг 7.36. Выбор повторяющихся людей в списке персонала

```
Сотрудник                    Должность                       Оклад
============================ ============================== =========
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ      Ночной сторож                    1500.00
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ      Исполнительный директор         38500.00
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ      Генеральный директор            56000.00
ПАРШИНА АНТОНИНА ПЕТРОВНА    Техник                           7300.00
ПАРШИНА АНТОНИНА ПЕТРОВНА    Служба безопасности             12000.00
```

Посмотрим еще раз на предложение WHERE, которое и задает нужные условия выборки данных.

```
  WHERE S.COD IN
       (SELECT N.COD FROM STAFF N
          WHERE (SELECT COUNT(*)
                   FROM STAFF M
                   WHERE N.CODPEOPLE = M.CODPEOPLE) > 1)
```

Несмотря на кажущуюся сложность, это предложение достаточно простое (поскольку было выстрадано долгими ночами). Здесь важным моментом является то, что в SQL в подзапросах мы можем использовать псевдонимы *всех* таблиц, указанных в данном операторе SELECT.

Самый внутренний, третий, оператор SELECT подсчитывает количество строк исходной таблицы STAFF, имеющих одно и то же значение кода человека. Третье по счету в операторе предложение WHERE задает для выборки только те строки, где это количество больше единицы. Пожалуй, самый тонкий здесь момент — это то, что мы ссылаемся на коды людей в самом внутреннем операторе SELECT и на коды людей в операторе SELECT более высокого уровня. При этом используются соответствующие псевдонимы таблиц.

Оператор SELECT уровнем выше (стало быть, это второй уровень) формирует список кодов персонала, которые присутствуют в списке кодов, полученных с помощью самого внутреннего оператора SELECT.

Самый верхний уровень предложения WHERE отбирает для вывода записи с кодами, заданными в этом списке. Все!

7.6. Использование представлений

После того как мы рассмотрели возможности оператора SELECT, имеет смысл познакомиться с представлениями.

Представление (view) иногда переводится на русский язык как обзор или просмотр.

Представление — это объект реляционной базы данных, который описывает *виртуальную* таблицу. Определение представления содержит оператор SELECT любой сложности, при выполнении которого получается та самая виртуальная таблица, которая физически в базе данных не хранится.

Фактически представление — это заранее подготовленный и сохраненный в области метаданных оператор SELECT, выбирающий нужные нам столбцы и строки из любого количества таблиц базы данных.

> ***Замечание***
>
> Результаты выполнения представления в базе данных не хранятся. Там присутствует лишь определение представления. Результаты будут получены при обращении к представлению с помощью опять же оператора SELECT.

Для создания представлений, описанных в этом разделе, можно в IBExpert использовать как инструмент Script Executive, так и SQL Editor. В окне Script Executive после оператора создания представления помещайте оператор COMMIT для подтверждения транзакции, чтобы созданное представление было видно в других транзакциях. При использовании SQL Editor для подтверждения транзакции нужно на панели инструментов щелкнуть мышью по кнопке **Commit Transaction**.

7.6.1. Создание представлений

Синтаксис оператора создания представления:

```
CREATE VIEW <имя представления>
  [(<имя столбца> [, <имя столбца>] ...)]
AS <оператор SELECT>;
```

Имя представления должно быть уникальным среди имен представлений, таблиц и хранимых процедур базы данных.

В скобках можно указать необязательный список имен столбцов, возвращаемых представлением. Есть случаи — мы рассмотрим один такой, — когда этот список является обязательным.

Оператор `SELECT` — это обычный, сколь угодно сложный оператор выборки данных, который может содержать операции соединения таблиц, группировку результатов запроса и т. д. По-моему единственное ограничение для этого оператора — запрет на использование предложения `ORDER BY`, которое по большому счету совсем здесь ни к месту.

Обращение к представлению выполняется с помощью обычного оператора `SELECT`, который может как угодно упорядочивать, группировать полученный результат, выбирать отдельные строки на основании условий поиска.

7.6.2. Примеры представлений

Рассмотрим примеры создания и использования представлений.

Посмотрите на листинг 7.9. Такой результат мы можем получить, если выполним следующий запрос:

```
SELECT CODREG  AS "Код региона",
       NAMEREG AS "Регион",
       CENTER  AS "Центр региона"
  FROM REFREG
  WHERE CODCTR = (SELECT CODCTR
                    FROM REFCTR
                    WHERE NAME = 'РОССИЯ')
  ORDER BY CENTER;
```

Мы можем создать представление, выполняющее такие же действия:

```
CREATE VIEW VIEW_RUSSIA
            (CODREG,
             NAMEREG,
             CENTER)
AS
SELECT CODREG,
       NAMEREG,
```

```
      CENTER
  FROM REFREG
  WHERE CODCTR = (SELECT CODCTR
                  FROM REFCTR
                  WHERE NAME = 'РОССИЯ');
```

В операторе создания представления после имени представления мы указали список имен столбцов. В таком операторе в этом нет необходимости.

Обращение к данному представлению может быть следующим:

```
SELECT * FROM VIEW_RUSSIA
  ORDER BY CENTER;
```

Здесь в качестве списка выбора можно указать символ *, выражение или список выбираемых столбцов. Для добавления осмысленных заголовков можно выполнить:

```
SELECT CODREG  AS "Код региона",
       NAMEREG AS "Регион",
       CENTER  AS "Центр региона"
  FROM VIEW_RUSSIA
  ORDER BY CENTER;
```

Теперь посмотрите на листинг 7.26. Похожий список мы можем получить, если выполним оператор:

```
SELECT PEOPLE.FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM STAFF
    FULL OUTER JOIN PEOPLE
      ON STAFF.CODPEOPLE = PEOPLE.COD
  ORDER BY 1;
```

Создадим подходящее представление:

```
CREATE VIEW VIEW_PEOPLE1 AS
SELECT PEOPLE.FULLNAME,
       DUTIES,
       SALARY,
       STAFF.CODORG
  FROM STAFF
    LEFT OUTER JOIN PEOPLE
      ON STAFF.CODPEOPLE = PEOPLE.COD;
```

Чтобы получить список всех сотрудников всех организаций, можно обратиться к этому представлению:

```
SELECT FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM VIEW_PEOPLE1
  ORDER BY FULLNAME;
```

Результат будет похожим на то, что можно увидеть в листинге 7.28.

В этом представлении в список выбора я поместил и код организации. Это будет полезным, если нам понадобится получать не весь список сотрудников, а только сотрудников конкретной организации.

Чтобы выбрать всех сотрудников организации с кодом 11, нужно выполнить оператор:

```
SELECT FULLNAME AS "Сотрудник",
       DUTIES AS "Должность",
       SALARY AS "Оклад"
  FROM VIEW_PEOPLE1
  WHERE CODORG = 11
  ORDER BY FULLNAME;
```

В листинге 7.29 показан результат двойного левого внешнего соединения таблиц. Для получения такого результата можно создать следующее представление.

```
CREATE VIEW VIEW_PEOPLE2 AS
SELECT P.FULLNAME,
       S.DUTIES,
       O.NAME
  FROM STAFF S
    LEFT OUTER JOIN PEOPLE P
      ON S.CODPEOPLE = P.COD
    LEFT OUTER JOIN ORGANIZATION O
      ON O.COD = S.CODORG;
```

Самый сложный из рассмотренных здесь операторов `SELECT` осуществляет выборку людей, их супругов и родителей, как показано в листинге 7.31. В операторе таблица людей трижды соединяется сама с собой.

Напишем соответствующее представление.

```
CREATE VIEW VIEW_PEOPLE3
       (FULLNAME,
        OTHERNAME,
        MOTHERNAME,
        FATHERNAME)
```

```
AS
SELECT PG.FULLNAME,
       PH.NAME3,
       PM.NAME3,
       PF.NAME3
  FROM PEOPLE PG                        /* Главная таблица */
    LEFT OUTER JOIN PEOPLE PH           /* Супруг */
       ON PG.CODOTHERHALF = PH.COD
    LEFT OUTER JOIN PEOPLE PM           /* Мать */
       ON PG.CODMOTHER = PM.COD
    LEFT OUTER JOIN PEOPLE PF           /* Отец */
       ON PG.CODFATHER = PF.COD
  WHERE PG.CODOTHERHALF IS NOT NULL AND
        PG.CODMOTHER IS NOT NULL AND
        PG.CODFATHER IS NOT NULL;
```

В этом случае задание списка имен столбцов после имени представления обязательно, потому что в операторе `SELECT` трижды упоминается столбец `NAME3` из разных строк одной и той же таблицы, что создает неопределенность.

> ***Замечание***
>
> По правде сказать, я вначале считал, что в подобном случае сервер базы данных сформирует уникальные имена, как это было сделано в обычном операторе `SELECT`, однако я ошибся.

Обращение к этому представлению осуществляется так же обычным образом через оператор `SELECT`.

Задание списка имен столбцов в определении представления также необходимо, когда оператор `SELECT` помимо столбцов возвращает выражение. Рассмотрим пример. Если бы в таблице персонала у нас не было вычисляемого столбца `NET_SALARY`, мы могли бы через представление сформировать соответствующее выражение.

Можно создать, например, такое представление.

```
CREATE VIEW VIEW_PEOPLE4
       (FULLNAME,
        DUTIES,
        NAME,
        NEW_SALARY)
AS
SELECT P.FULLNAME,
       S.DUTIES,
       O.NAME,
       SALARY * .87
  FROM STAFF S
```

```
LEFT OUTER JOIN PEOPLE P
  ON S.CODPEOPLE = P.COD
LEFT OUTER JOIN ORGANIZATION O
  ON O.COD = S.CODORG;
```

Здесь используется выражение (`SALARY * .87`). По этой причине мы обязаны указать список имен столбцов представления. Проверьте работу представления, выполнив следующий оператор:

```
SELECT * FROM VIEW_PEOPLE4;
```

7.6.3. Типы представлений: только для чтения или изменяемые

Полученные представления могут позволять изменять таблицы, лежащие в их основе. Такие представления называются изменяемыми. Другой тип представления, представление только для чтения, не позволяет изменять соответствующие таблицы.

Чтобы представление было изменяемым, должны быть выполнены три условия:

- представление должно использовать одну таблицу или другое изменяемое представление;
- все столбцы базовой таблицы, не включенные в определение представления, допускают пустое значение (`NULL`);
- оператор `SELECT` в представлении не содержит подзапросов, ключевого слова `DISTINCT`, предложения `HAVING`, агрегатных функций, соединяемых таблиц, хранимых процедур.

Если будет нарушено хотя бы одно из этих условий, то представление будет только для чтения. Однако в таблицы и такого представления могут быть внесены изменения при работе с этим представлением. Это возможно, в первую очередь, при использовании соответствующих триггеров.

Не думаю, что нам с вами часто придется пользоваться изменяемыми представлениями.

7.6.4. Удаление представлений

Вы можете удалить ненужное представление. Для этого используется простой оператор:

```
DROP VIEW <имя представления>;
```

Оператор удаляет определение представления из базы данных. На таблицах, используемых в представлениях, это никак не сказывается.

Не существует по непонятным причинам оператора изменения представления. Если вам нужно изменить существующее представление, вы должны его удалить и заново создать представление с тем же именем и нужными характеристиками.

Что там за перевалом?

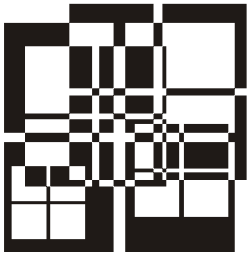
В этой главе мы довольно подробно рассмотрели оператор `SELECT`. Познакомились с представлениями.

Надеюсь, вам понравились наши исследования оператора `SELECT`, его возможностей. Они действительно впечатляют. Лично я получил большой объем новых знаний по этому оператору. Многое из рассмотренного я раньше в своей деятельности не использовал. Теперь мой арсенал средств существенно расширился. Надеюсь, ваш тоже.

В реляционных базах данных существует такой механизм, как транзакции. При кажущейся сложности транзакции достаточно просты и удобны в работе. От правильных вариантов задания свойств используемым в программах транзакциям зависит комфортность работы группы пользователей с одной базой данных в архитектуре "клиент-сервер".

Всю следующую главу мы посвятим транзакциям, их характеристикам и влиянию этих характеристик на взаимодействие параллельных процессов. Как обычно, мы будем рассматривать каждое свойство, каждую возможность и тут же проверять на практике, как это работает. Мы также напишем ряд исследовательских программ, которые в дальнейшем при несложных доработках сможем использовать в реальных промышленных программных системах.

ГЛАВА 8



Транзакции

Все действия, выполняемые с базой данных — любые изменения, как данных, так и метаданных, а также любая выборка данных — осуществляются в контексте (под управлением) какой-либо транзакции. Изменения, выполненные в контексте одной транзакции, можно либо все подтвердить (при отсутствии ошибок базы данных), либо все отменить. Если в любой операции, выполняемой в контексте транзакции, произошла ошибка, то подтвердить такую транзакцию нельзя. Можно только отменить все действия.

В этой главе мы будем использовать и некоторые сведения из интерфейса прикладного программирования (API), поскольку в процессе проведения исследовательских работ при формировании характеристик транзакций нам придется использовать некоторые константы, определенные в API.

При использовании операторов языка SQL для работы с базой данных для запуска транзакции и задания ее характеристик используется оператор `SET TRANSACTION`. В API серверов InterBase/Firebird используется функция `isc_start_transaction()` и буфер параметров транзакции Transaction Parameter Buffer (TPB). При проведении наших исследований мы будем сами формировать состав этого буфера TPB.

Для подтверждения транзакции используется оператор SQL `COMMIT` или эквивалентная функция API `isc_commit_transaction()`, для отмены всех действий транзакции используется оператор `ROLLBACK` или функция API `isc_rollback_transaction()`.

В InterBase/Firebird также поддерживаются замечательные средства создания контрольных точек сохранения транзакции и отката на любую из существующих точек. Эти средства мы рассмотрим несколько позже.

Для задания характеристик через буфер параметров транзакции используются мнемонические константы, чьи имена, к сожалению, не всегда соответствуют элементам синтаксиса оператора `SET TRANSACTION`.

8.1. Синтаксис оператора *SET TRANSACTION*

Вот несколько упрощенный синтаксис оператора `SET TRANSACTION`:

```
SET TRANSACTION
   [READ WRITE | READ ONLY] /* режим доступа */
   [WAIT | NO WAIT]         /* режим разрешения блокировок */
[[ISOLATION LEVEL]          /* уровень изоляции */
   {SNAPSHOT |
    SNAPSHOT TABLE STABILITY |
    READ COMMITTED [{RECORD_VERSION |
                     NO RECORD_VERSION}]]
[RESERVING <предложение резервирования>]
```

Предложение `RESERVING` задает необязательное резервирование таблиц. Синтаксис предложения резервирования следующий:

```
<таблица> [, <таблица> ...]
  [FOR [SHARED | PROTECTED] {READ | WRITE}]
  [, <предложение резервирования> ...]
```

Значением по умолчанию для транзакции (когда не заданы характеристики в операторе `SET TRANSACTION` или буфер параметров транзакции пустой) явля-
ется:

```
SET TRANSACTION READ WRITE WAIT SNAPSHOT;
```

Этому набору характеристик по умолчанию соответствует следующий состав параметров TPB:

```
isc_tpb_concurrency
isc_tpb_write
isc_tpb_wait
```

Что означают эти параметры, мы сейчас с вами рассмотрим.

8.2. Характеристики транзакций

Существует четыре основные характеристики транзакций:

- режим доступа;

- уровень изоляции;
- режим разрешения блокировок:
- средства резервирования.

Средства резервирования мы рассмотрим несколько позже, после исследования других характеристик.

8.2.1. Режим доступа

Самая простая характеристика — *режим доступа*. Может принимать значения `READ WRITE` и `READ ONLY`.

`READ WRITE` (параметр `isc_tpb_write` в TPB) позволяет в рамках данной транзакции не только читать, но и изменять данные базы данных.

В случае `READ ONLY` (параметр `isc_tpb_read` в TPB) в контексте данной транзакции допустимы только операции чтения данных.

8.2.2. Уровень изоляции

Важнейшая характеристика транзакции — *уровень изоляции*. В табл. 8.1 представлены три существующих в наших СУБД уровня изоляции транзакции.

Таблица 8.1. *Уровни изоляции транзакции в InterBase/Firebird*

SQL	Константа TPB	Значение
`READ COMMITTED`	`isc_tpb_read_committed`	Чтение подтвержденных изменений. Транзакция может видеть самые последние подтвержденные изменения базы данных, выполненные другими транзакциями. При этом уровне изоляции используются еще два взаимоисключающих параметра: По умолчанию `NO RECORD_VERSION` (`isc_tpb_no_rec_version` в TPB) требует, чтобы было выполнено подтверждение всех измененных другими транзакциями данных. `RECORD_VERSION` (`isc_tpb_rec_version` в TPB) позволяет читать самую последнюю подтвержденную версию изменений, даже если существуют другие неподтвержденные версии

Таблица 8.1 *(окончание)*

SQL	Константа TPB	Значение
SNAPSHOT	isc_tpb_concurrency	Мгновенный снимок (образ). Значение по умолчанию. В литературе можно найти и другое название — повторяемое чтение (Repeatable Read). Дает состояние базы данных на момент старта транзакции. Изменения, выполненные другими транзакциями, в данной транзакции не видны. Естественно, транзакция "видит" все изменения, выполненные в контексте этой транзакции
SNAPSHOT TABLE STABILITY	isc_tpb_consistency	Изолированный образ или упорядочиваемый, сериализуемый (Serializable) образ. Аналогичен уровню SNAPSHOT с тем отличием, что другим транзакциям разрешено чтение данных из таблиц данной транзакции, однако они не могут вносить в них никаких изменений

Помимо перечисленных в таблице уровней изоляции в литературе по базам данных описывается еще один уровень — `DIRTY READ`, грязное чтение, или, другими словами, неподтвержденное чтение, `READ UNCOMMITTED`. Этот уровень позволяет транзакции читать неподтвержденные изменения, выполненные другими транзакциями. В наших СУБД InterBase и Firebird такой уровень изоляции не поддерживается.

8.2.3. Режим разрешения блокировок

Еще одной характеристикой транзакции является *режим разрешения блокировок*. Может принимать значения `WAIT` и `NO WAIT`.

Если установлено `WAIT` (параметр `isc_tpb_wait` в блоке параметров транзакции TPB), то при появлении конфликтов обновления данная транзакция будет ожидать разрешения блокировки со стороны других транзакций путем выдачи ими оператора подтверждения или отмены транзакции.

Если же задано `NO WAIT` (параметр `isc_tpb_nowait` в TPB), то при появлении блокировки данная транзакция немедленно вызывает исключение и формирует значения кодов ошибки.

Напомню, что `WAIT` устанавливается по умолчанию.

8.3. Для особо одаренных. Написание исследовательской программы

Сейчас мы с вами напишем программы с использованием компонентов FIBPlus и IBX. Это программы, реализующие связь между главной и подчиненной таблицами (master/detail) *(см. разд. 6.3)*. В программах мы будем динамически, в процессе их выполнения, формировать различные характеристики транзакций. Любую из этих программ мы будем использовать для исследования характеристик транзакций, для проверки взаимодействия параллельных процессов друг на друга. Мы будем создавать конфликтные ситуации и смотреть, как программы с разными характеристиками транзакций реагируют на эти ситуации.

8.3.1. Использование компонентов FIBPlus

В Delphi 7 создайте новый проект. Создайте на форме панель инструментов с кнопками `TButton`, положите два компонента `TDBDrid`: `DBGridCountry` и `DBGridRegion`. Добавьте два компонента `TDataSource`: `DataSourceCountry` и `DataSourceRegion`.

С вкладки **FIBPlus** поместите на форму следующие компоненты: `Database1` типа `TpFIBDatabase`, компонент `Transaction1` типа `TpFIBTransaction`, два компонента типа `TpFIBDataSet`: `DataSetCountry` и `DataSetRegion`. Поместите на форму также компонент `ErrorHandler` (рис. 8.1).

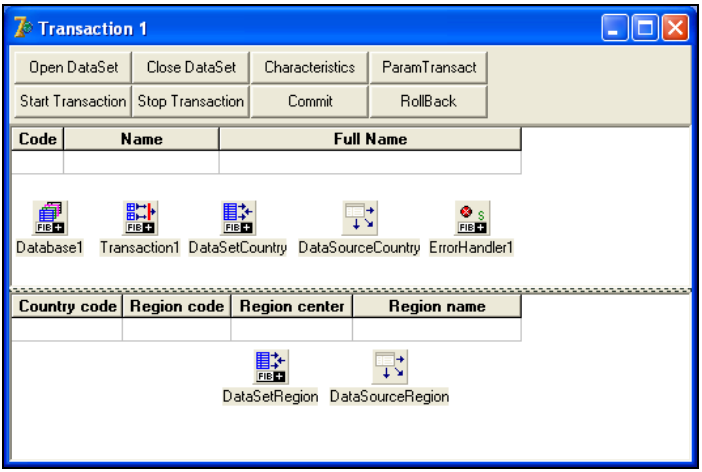


Рис. 8.1. Главная форма программы исследования характеристик транзакций

> **Замечание**
>
> В этой программе опять использованы английские тексты. По причине *моей* лености я здесь просто использовал написанную мною ранее программу, с помощью которой исследовал свойства транзакций, не потрудившись перевести все на русский язык. Надеюсь, вас это не очень огорчает.

Для объектов базы данных зададим необходимые значения.

- Имя базы данных у компонента `Database1` (свойство `DBName`) — `Work.fdb`. Как вы помните, файл базы данных располагается в каталоге D:\BestDatabase.
- Имя пользователя (`UserName`) — `WIZARD`.
- Пароль (`Password`) — `master`.
- Набор символов (`CharSet`) — `WIN1251`.
- Диалект базы данных (`SQLDialect`) — `3`.

В качестве транзакции по умолчанию (`DefaultTransaction`), которая используется при чтении данных, и транзакции, в контексте которой будут выполняться изменения данных (`DefaultUpdateTransaction`), выберем `Transaction1`.

Свойства компонента базы данных в Инспекторе объектов будут выглядеть так — рис. 8.2.

Для компонента транзакции `Transaction1` установим следующие значения (рис. 8.3).

- Выберем из выпадающего списка имени базы данных (`DefaultDatabase`) `Database1`.
- Для уровня изоляции транзакции `TPBMode` из выпадающего списка выберем значение `tpbDefault` — только в этом случае мы сможем изменять содержимое буфера параметров транзакции (Transaction Parameter Buffer, TPB); при выборе значения `tpbReadCommitted` или `tpbRepeatableRead` в момент запуска транзакции в буфер параметров будут помещаться соответствующие константы, и изменить эту ситуацию никакими силами нельзя. Для изменения характеристик транзакции `Transaction1` мы напишем специальный модуль.

В дальнейшем мы будем динамически, в процессе выполнения программы, создавать характеристики транзакции `Transaction1`.

Для компонента набора данных `DataSetCountry` большинство свойств можно оставить в том виде, как они задаются по умолчанию. Установим следующие значения.

- База данных (`Database`) — `Database1`.

- Транзакция (свойство `Transaction`) и транзакция для изменений (`UpdateTransaction`) — `Transaction1`. Скорее всего, значения этих свойств уже будут установлены автоматически.

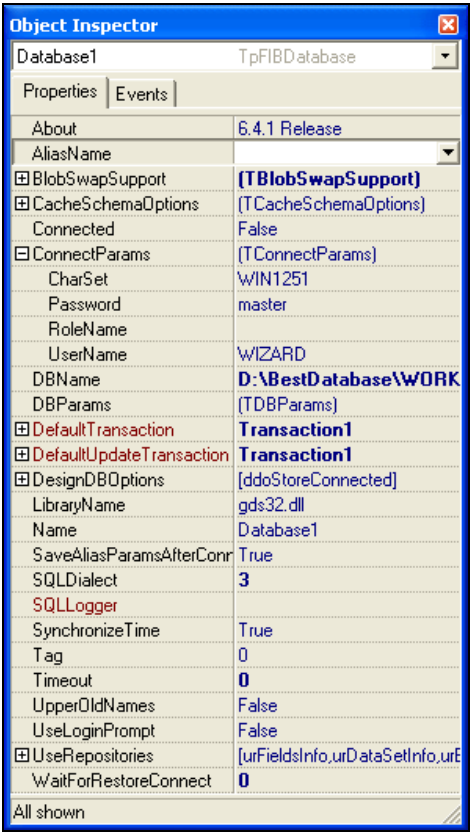


Рис. 8.2. Свойства компонента базы данных

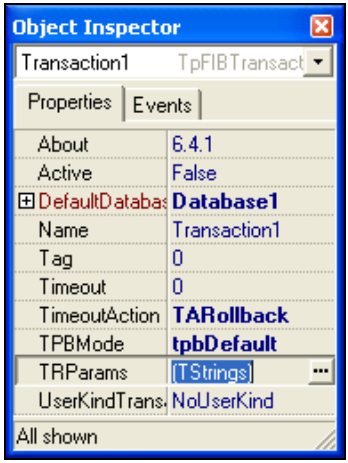


Рис. 8.3. Свойства транзакции `Transaction1`

- Проследите за тем, чтобы свойство `AutoCommit` было установлено в `False`. Это запрещает автоматически подтверждать транзакцию после внесения изменений в набор данных.
- В списке режимов (`Options`) для подсвойства `poStartTransaction` установим значение `False`. Это важно, поскольку мы собираемся явно управлять запуском и подтверждением (или откатом) транзакции. Подсвойство `poKeepSorting` установим в `True`. Для целей исследования поведения транзакций это особой роли не играет, однако бывает полезным для сохранения упорядоченности набора данных в случае внесения изменений в столбцы, по которым осуществляется упорядочивание набора данных (предложение `ORDER BY` в операторе `SELECT`).

- В списке `PrepareOptions` можно (в нашем случае, конечно, необязательно) установить `psAskRecordCount` в `True`. Это бывает полезным, если вам после открытия набора данных нужно в строке состояния указать количество полученных записей.

Вызовем генератор SQL (щелчок правой кнопкой мыши на компоненте и выбор в контекстном меню строки **SQL Generator** — все это относительно компонента `DataSetCountry`). В списке таблиц выберем `REFCTR` и дважды щелкнем мышью по этой строке. Будет сгенерирован оператор `SELECT`. В конец оператора добавим предложение `ORDER BY NAME`, чтобы упорядочить получаемый набор данных по названиям стран (рис. 8.4).

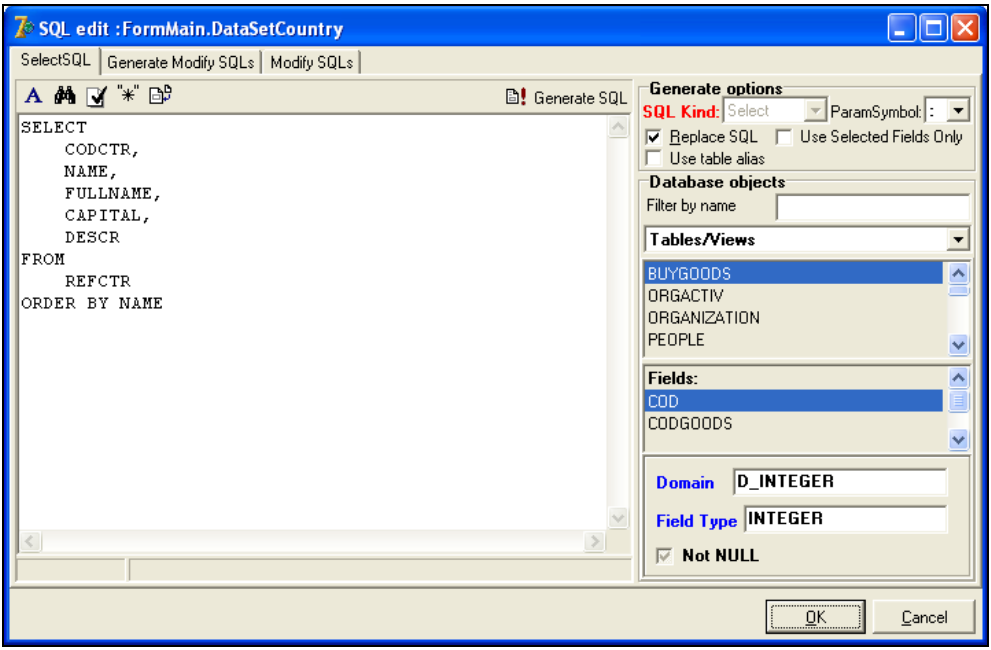


Рис. 8.4. Генерация оператора `SELECT`

Перейдем на закладку **Generate Modify SQLs** и щелкнем мышью по кнопке **Generate SQLs**. Будут сгенерированы операторы для добавления (`Insert`), изменения (`Update`), удаления (`Delete`) и повторного чтения текущей строки (`Refresh`). Не забудьте снять флажок с поля **Non update primary key** (Не изменять первичный ключ) (рис. 8.5).

Компонент `DataSourceCountry` свяжем с набором данных `DataSetCountry` (свойство `DataSet`).

Аналогичные действия нужно выполнить и с компонентом набора данных регионов `DataSetRegion`. Поскольку этот компонент является детальным (дочерним, подчиненным) набором данных в связке master-detail (главная-подчиненная), нужно установить его свойство `DataSource` в `DataSourceCountry`, а в свойстве `DetailConditions` следует установить в `True` подсвойства `dcForceOpen` и `dcWaitEndMasterScroll`. При генерации оператора `SELECT` в генераторе SQL нужно скорректировать оператор следующим образом:

```
SELECT
    CODCTR,
    CODREG,
    NAMEREG,
    CENTER
FROM
    REFREG
WHERE CODCTR = :MAS_CODCTR
ORDER BY CENTER
```

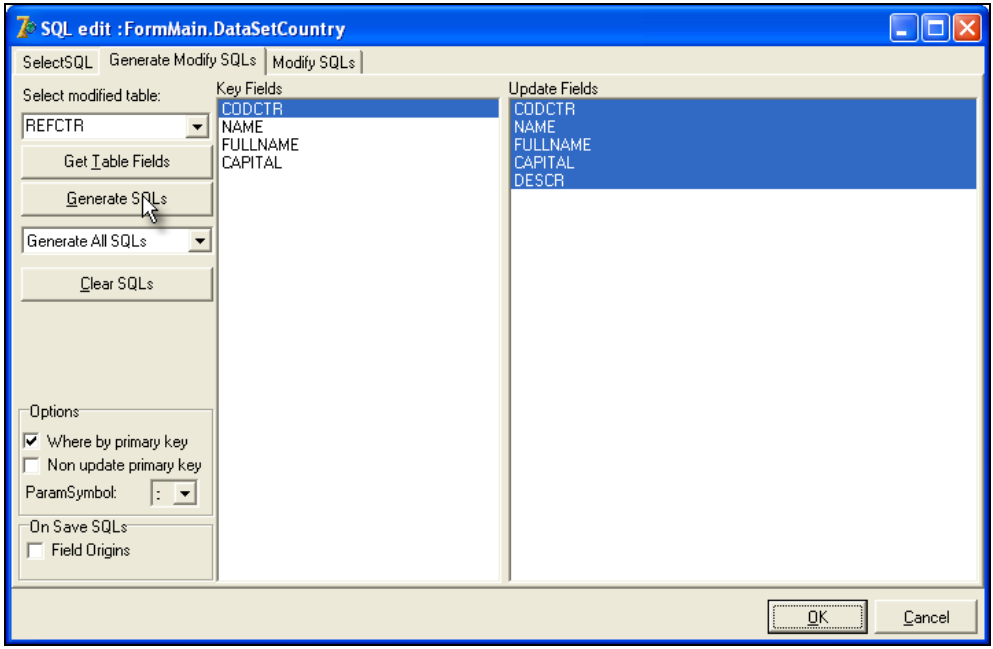


Рис. 8.5. Генерация модифицирующих операторов

Здесь предложение `WHERE CODCTR = :MAS_CODCTR` задает выборку только тех строк таблицы регионов, которые относятся к текущей стране. Более подробно про связь "главная-подчиненная" и ее реализацию с помощью компонентов FIBPlus читайте в соответствующей статье на сайте **www.devrace.com**.

Далее создадим еще две формы: одна, `Transaction's Characteristics`, будет использоваться для формирования списка параметров транзакции `TRParams`, другая, `Transaction's Parameters`, позволит отображать содержимое буфера параметров транзакции (TPB).

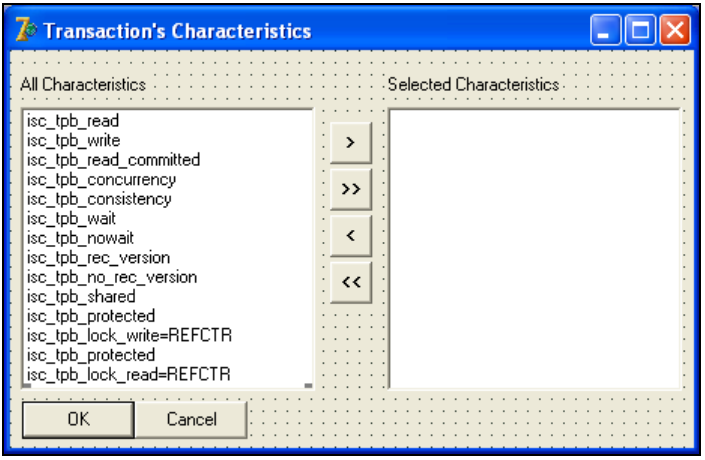


Рис. 8.6. Форма `Transaction's Characteristics`

Форма `Transaction's Characteristics` (рис. 8.6) нам понадобится для создания списка характеристик транзакции. Левому компоненту `ListBox` присвоим имя `AllParameters`, правому — `SelectedParameters`. Для обоих компонентов установим в `True` значение свойства `MultiSelect`, чтобы дать возможность пользователю выбирать в списке несколько строк. Для этого нужно, удерживая нажатой клавишу <Ctrl>, щелкать мышью по нужным строкам.

Напишем следующие обработчики событий щелчка по кнопкам. При щелчке по кнопке ">" выполняется перемещение выбранных строк из левого компонента `ListBox` в правый (листинг 8.1).

Листинг 8.1. Перемещение выбранных строк в правый список

```
procedure TFormTrans.BSelectOneClick(Sender: TObject);
var I: Integer;
begin
  I := 0;
  while (I <= AllParameters.Items.Count – 1) do
  begin
    if AllParameters.Selected[I] then
    begin
      SelectedParameters.Items.Add(AllParameters.Items[I]);
      AllParameters.Items.Delete(I);
```

```
      I := I – 1;
    end;
    I := I + 1;
  end;
end;
```

При щелчке по кнопке ">>" выполняется перемещение всех строк из левого компонента `ListBox` в правый. По правде сказать, эта функция для наших целей не нужна и приводится здесь для сохранения принятого порядка (листинг 8.2).

Листинг 8.2. Перемещение всех строк в правый список

```
procedure TFormTrans.BSelectAllClick(Sender: TObject);
var I: Integer;
begin
  for I := 0 to AllParameters.Items.Count – 1 do
    SelectedParameters.Items.Add(AllParameters.Items[I]);
  AllParameters.Items.Clear;
end;
```

Похожим образом реализуется и перемещение строк из правого списка в левый.

При щелчке по кнопке "<" выполняется перемещение выбранных строк из правого компонента `ListBox` в левый, т. е. осуществляется удаление отмеченных строк из правого списка (листинг 8.3).

Листинг 8.3. Перемещение выбранных строк в левый список

```
procedure TFormTrans.BRemoveOneClick(Sender: TObject);
var I: Integer;
begin
  I := 0;
  while (I <= SelectedParameters.Items.Count – 1) do
  begin
    if SelectedParameters.Selected[I] then
    begin
      AllParameters.Items.Add(SelectedParameters.Items[I]);
      SelectedParameters.Items.Delete(I);
      I := I – 1;
    end;
    I := I + 1;
  end;
end;
```

При щелчке по кнопке "<<" выполняется перемещение всех строк из правого компонента `ListBox` в левый (листинг 8.4).

Листинг 8.4. Перемещение всех строк в левый список

```
procedure TFormTrans.BRemoveAllClick(Sender: TObject);
var I: Integer;
begin
  for I := 0 to SelectedParameters.Items.Count – 1 do
    AllParameters.Items.Add(SelectedParameters.Items[I]);
  SelectedParameters.Items.Clear;
end;
```

Форма `Transaction's Parameters` (рис. 8.7) позволяет отобразить содержимое списка параметров транзакции (`TRParams`) и вектора буфера параметров транзакции (TPB). При вызове формы в поле `Memo` отображается список параметров и числовые значения из TPB.

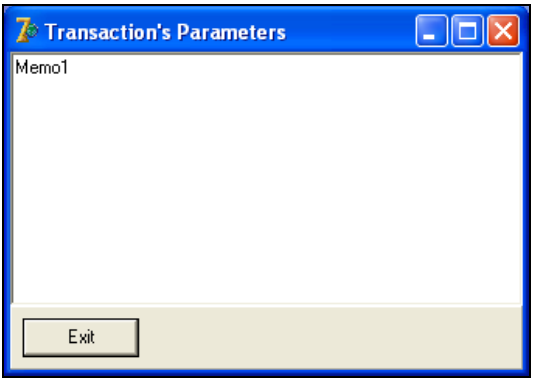


Рис. 8.7. Форма `Transaction's Parameters`

Все действия выполняются при отображении формы (листинг 8.5).

Листинг 8.5. Отображение характеристик транзакции

```
procedure TFormTransactionParam.FormShow(Sender: TObject);
var I: Integer;
begin
  Memo1.Clear;
  for I := 0 to FormMain.Transaction1.TRParams.Count – 1 do
    Memo1.Lines.Add(FormMain.Transaction1.TRParams.Strings[I]);
  Memo1.Lines.Add('============================');
  for I := 0 to FormMain.Transaction1.TPBLength – 1 do
```

```
    Memo1.Lines.Add(IntToStr(Integer(FormMain.Transaction1.TPB[I])));
end;
```

Результат отображения состоит из двух частей. В первой части воспроизводятся имена параметров транзакции, во второй части — их числовые значения.

Вернемся в главный модуль. Напишем обработчики событий формы. В событии формы `OnShow` выполним подключение к базе данных, в событии `OnClose` отключимся от базы данных. Соответственно, это будут операторы:

```
Database1.Open;
```

и

```
Database1.Close;
```

Щелчок по кнопке **Open DataSet** приводит к открытию главного набора данных `DataSetCountry`. При этом автоматически открывается и набор данных `DataSetRegion` (листинг 8.6).

Листинг 8.6. Открытие набора данных

```
procedure TFormMain.BOpenDataSetClick(Sender: TObject);
begin
  DataSetCountry.Open;
end;
```

Щелчок по кнопке **Close DataSet** закрывает оба набора данных:

```
DataSetCountry.Close;
```

Щелчок по кнопкам `BStartTransaction` и `BStopTransaction` приводит, соответственно, к запуску и останову транзакции:

```
Transaction1.StartTransaction;
```

и

```
Transaction1.Active := False;
```

Обработка события щелчка по кнопке **Commit** подтверждает транзакцию. После подтверждения транзакции она запускается заново, после чего открывается набор данных (он автоматически будет закрыт при завершении транзакции — по `COMMIT` или `ROLL BACK`) — листинг 8.7.

Листинг 8.7. Подтверждение транзакции

```
procedure TFormMain.BCommitClick(Sender: TObject);
begin
  Transaction1.Commit;
  Transaction1.StartTransaction;
```

```
  DataSetCountry.Open;
end;
```

Похожим образом обрабатывается событие щелчка по кнопке **RollBack** (листинг 8.8).

Листинг 8.8. Откат транзакции

```
procedure TFormMain.BRollBackClick(Sender: TObject);
begin
  Transaction1.Rollback;
  Transaction1.StartTransaction;
  DataSetCountry.Open;
end;
```

Щелчок по кнопке **Characteristics** приводит к обращению к форме `Transaction's Characteristics` для формирования пользователем списка характеристик транзакции (листинг 8.9).

Листинг 8.9. Формирование характеристик транзакции

```
procedure TFormMain.BCharactTransactClick(Sender: TObject);
var I: Integer;
begin
  if FormTrans.ShowModal <> IDOK then exit;
  if Transaction1.Active then
    Transaction1.Active := False;
  Transaction1.TRParams.Clear;
  for I := 0 to FormTrans.SelectedParameters.Items.Count – 1 do
    Transaction1.TRParams.Add(
      FormTrans.SelectedParameters.Items[I]);
end;
```

Щелчок по кнопке **ParamTransact** приводит просто лишь к вызову формы `Transaction's Parameters`, которая отобразит текущие характеристики транзакции (листинг 8.10).

Листинг 8.10. Отображение характеристик транзакции

```
procedure TFormMain.BParamTransactClick(Sender: TObject);
begin
  FormTransactionParam.ShowModal;
end;
```

И, наконец, напишем обработчик ошибок базы данных (листинг 8.11). Это позволит нам в случае ошибок увидеть не только сообщение сервера базы данных, но и значения кодов SQLCODE и GDSCODE. Для этих целей используем событие OnFIBErrorEvent компонента ErrorHandler. Коды ошибок см. в *приложении 3*.

Листинг 8.11. Обработчик ошибок базы данных

```
procedure TFormMain.ErrorHandler1FIBErrorEvent(Sender: TObject;
  ErrorValue: EFIBError; KindIBError: TKindIBError;
  var DoRaise: Boolean);
var S: String;
begin
  S := S + 'SQLCode = ' + IntToStr(ErrorValue.SQLCode) + #10#13;
  S := S + 'IBErrorCode = ' + IntToStr(ErrorValue.IBErrorCode) + #10#13;
  S := S + 'IBMessage = ' + ErrorValue.IBMessage + #10#13;
  Application.MessageBox(PAnsiChar(S), 'Database Error',
    MB_OK + MB_ICONSTOP);
  DoRaise := False;
end;
```

Для того чтобы трансляция такого текста прошла без ошибок, в список используемых модулей программы (предложение **uses**) необходимо добавить fib.

Текст программы находится в каталоге Chapter08\Transaction\FIBPlus.

> ***Замечание***
>
> Этот обработчик ошибок только выдает сообщение и не выполняет никаких других действий. После появления ошибки вам нужно при использовании этой учебной программы вручную остановить транзакцию, заново ее запустить и открыть набор данных, чтобы продолжить работу. В реальной жизни вы, конечно же, проведете в программе грамотный анализ ошибки и выполните все необходимые действия по ее нейтрализации и восстановлению работоспособности программы.

8.3.2. Использование компонентов IBX

Самый простой вариант получения программы с использованием компонентов IBX — это скопировать предыдущий проект в новый каталог. С формы нужно удалить компоненты FIBPlus (база данных, транзакция, наборы данных и обработчик ошибок) и из главной формы программы MasterDetail,

в которой использовались компоненты IBX, скопировать компоненты базы данных, транзакции и наборов данных.

После этого нужно связать компоненты `DataSource` с соответствующими наборами данных. В завершение корректировок нужно в текстах программы заменить имя свойства транзакции `TRParams` на `Params`. Эти имена встречаются в модулях Main и TransParam.

Необходимо также внести изменения в обработчики щелчка по кнопке подтверждения транзакции (**Commit**) и отмены, отката транзакции (**RollBack**). И там, и там нужно после открытия набора данных стран добавить оператор открытия набора данных регионов. При использовании компонентов FIBPlus открытие подчиненного набора данных происходит автоматически. Здесь же это нужно выполнить явно.

Удалите из программы текст существовавшего обработчика ошибок. Вы должны только удалить строки между операторами **`begin`** и **`end`**, а также строку описания переменной `S` перед оператором **`begin`**:

```
var S: String;
```

Не удаляйте здесь остальные строки процедуры, иначе получите ошибку, с которой придется некоторое время бороться.

> ***Замечание***
> Отсутствие в этой программе централизованного обработчика ошибок приведет лишь к тому, что в случае возникновения конфликта мы будем не сами формировать текст сообщения, а получим стандартное сообщение об ошибке.

Текст программы находится в каталоге Chapter08\Transaction\IBX.

На этом изменения заканчиваются. Можно приступать к исследованию характеристик транзакций.

8.3.3. Числовые значения параметров TPB

Ниже приводятся числовые значения параметров, используемых при формировании буфера параметров транзакции. Некоторые из них мы здесь не рассматриваем. Параметры определены в файле ibase.pas.

```
isc_tpb_version1                =         1;
isc_tpb_version3                =         3;
isc_tpb_consistency             =         1;
isc_tpb_concurrency             =         2;
isc_tpb_shared                  =         3;
isc_tpb_protected               =         4;
```

```
isc_tpb_exclusive               =         5;
isc_tpb_wait                    =         6;
isc_tpb_nowait                  =         7;
isc_tpb_read                    =         8;
isc_tpb_write                   =         9;
isc_tpb_lock_read               =        10;
isc_tpb_lock_write              =        11;
isc_tpb_verb_time               =        12;
isc_tpb_commit_time             =        13;
isc_tpb_ignore_limbo            =        14;
isc_tpb_read_committed          =        15;
isc_tpb_autocommit              =        16;
isc_tpb_rec_version             =        17;
isc_tpb_no_rec_version          =        18;
isc_tpb_restart_requests        =        19;
isc_tpb_no_auto_undo            =        20;
isc_tpb_last_tpb_constant       =        isc_tpb_no_auto_undo;
```

8.4. Исследование характеристик транзакций

Начнем эксперименты. Заметьте, что в наших программах мы имеем так называемую "длинную" транзакцию — пользователь вручную запускает транзакцию, выполняет различные изменения данных и подтверждает или отменяет транзакцию, когда ему заблагорассудится. Это может приводить к блокировкам.

Пример "короткой" транзакции. Пользователь в программе щелкает по кнопке **OK** на форме добавления или изменения данных в базе данных. После этого программа вызывает метод `Insert` или `Edit` для соответствующего набора данных, устанавливает значения столбцов и вызывает метод `Post`, который отправляет изменения в базу данных. Транзакция обновления запускается в момент вызова метода `Post`; сразу же после `Post` выполняется подтверждение транзакции. Как правило, время жизни такой короткой транзакции исчисляется долями секунды, что уменьшает вероятность блокировок при многопользовательской работе с базой данных.

Нашей задачей является выяснение условий появления блокировок, поэтому с исследовательскими целями мы будем использовать длинные транзакции.

В первую очередь нас будет интересовать взаимодействие программ в зависимости от уровня изоляции транзакций.

8.4.1. Уровень изоляции *READ COMMITTED*

Пожалуй, наиболее часто используемым является уровень изоляции `READ COMMITTED`. Он позволяет транзакции видеть подтвержденные изменения, выполненные другими транзакциями. Рассмотрим его использование.

Запустите два экземпляра созданной программы. Для целей исследования можно использовать любую из двух написанных нами программ, поскольку их функциональные возможности одинаковы. Программы лучше запускать не из среды Delphi, а выполнив EXE-файл. В этом случае при возникновении ошибок нам не потребуется вначале отвечать на сообщения среды разработки, а затем уж смотреть на текст сообщения об ошибке в базе данных.

В одной программе задайте следующие характеристики транзакции, щелкнув по кнопке **Characteristics**:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_rec_version
isc_tpb_nowait
```

Это означает, что транзакция может читать и писать данные (isc_tpb_write), имеет уровень изоляции `READ COMMITTED` (`isc_tpb_read_committed`), т. е. транзакция видит подтвержденные изменения, выполненные в других транзакциях. При этом транзакцию "устроит" любая последняя подтвержденная версия записи, даже если существуют другие неподтвержденные версии той же
записи (`isc_tpb_rec_version`). При возникновении конфликтов блокировки транзакция не будет ожидать разрешения конфликта со стороны другой транзакции, а сразу выдаст исключение (`isc_tpb_nowait`).

Для транзакции второй программы задайте следующие характеристики:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_rec_version
isc_tpb_wait
```

Отличие от первой транзакции только в последней строке. В случае возникновения конфликта блокировки вторая транзакция будет ожидать разрешение конфликта со стороны другой транзакции.

> ***ВНИМАНИЕ!***
> При любых уровнях изоляции транзакций конфликт блокировки всегда возникнет в том случае, когда две транзакции пытаются одновременно изменить одну и ту же строку таблицы.Вперед. Начнем создавать конфликтные ситуации и героически их разрешать.

В первой программе запустите транзакцию, откройте набор данных и на сетке `DBGridCountry` измените значение какого-либо столбца в справочнике

стран. Обязательно после изменения щелкните мышью по любой другой строке таблицы или клавишами перемещения курсора сделайте текущей другую строку. В этот момент выполняется операция `Post`, которая отправляет изменения в базу данных.

Аналогичным образом во второй программе запустите транзакцию, откройте набор данных, измените ту же самую строку, что и в первой программе (менять можно значение любого столбца, не обязательно того же самого, который был изменен в первой программе). Сделайте текущей другую строку таблицы, чтобы отправить изменения в базу данных.

Что произошло? Похоже, нашу вторую программу заклинило. Хорошо, если еще виден курсор мыши в виде песочных часов или привычной стрелочки. В некоторых случаях и курсора не видно.

Дело в том, что для второй программы вы задали режим ожидания разрешения конфликта блокировки (`isc_tpb_wait`). Щелкните по кнопке подтверждения транзакции в первой программе. Вторая программа сразу оживет и выдаст соответствующее сообщение об ошибке (это сработал наш обработчик ошибок).

Повторите те же действия — изменение строки в первой программе, изменение той же строки во второй программе. Вторая программа перейдет в режим ожидания. В это время отмените транзакцию в первой программе. Во второй программе изменения будут отправлены в базу данных без каких-либо ошибок.

Вывод первый. При использовании длинных транзакций не следует задавать режим ожидания конфликта блокировки.

Измените режим ожидания во второй программе: в характеристиках транзакции уберите параметр `isc_tpb_wait` и добавьте `isc_tpb_nowait`.

ВНИМАНИЕ!

Параметр `isc_tpb_wait` применяется по умолчанию. Вам всегда нужно явно задавать `isc_tpb_nowait`.

Замечание не для прессы

Готовя простенький демонстрационный пример, показывающий, как *короткие* транзакции хорошо себя ведут при одновременной работе нескольких клиентов с одной базой данных в сети, я допустил примитивную ошибку, сделав транзакцию *длинной* (не установил значения двух свойств). Одиннадцать моих студентов с клиентских компьютеров изменили одну и ту же запись и по моей команде перешли к другой строке, чтобы отправить изменения на сервер. В случае короткой транзакции блокировка могла возникнуть на небольшом количестве компьютеров. Однако при длинной транзакции на десяти компьютерах появилось сообщение об ошибке, и только на одном, разумеется, были нормально выполнены обновления.

Смоделируем еще один конфликт. В первой программе перейдите к записи USA и удалите ее, нажав клавиши <Ctrl>+<Del> и подтвердив удаление в появившемся стандартном диалоговом окне. Не подтверждайте транзакцию. Во второй программе также перейдите к строке USA и удалите ее. Возникнет конфликт. Отмените транзакции в обеих программах. Удаленная запись опять появится в списке. Снова удалите ее в первой программе. Во второй программе попытайтесь изменить или удалить любую запись в таблице регионов (нижняя сетка DBGridRegion) этой же страны. Здесь также возникнет конфликт. Конфликт возникнет и в том случае, когда вы измените в первой программе ключевой столбец (код страны), а во второй программе попытаетесь изменить или удалить подчиненную запись региона. В таблице регионов столбец CODCTR является внешним ключом, ссылающимся на код страны в таблице стран. В описании внешнего ключа было указано:

```
CONSTRAINT FK_REFREG
    FOREIGN KEY (CODCTR) REFERENCES REFCTR (CODCTR)
      ON DELETE CASCADE
      ON UPDATE CASCADE
```

Это значит, в частности, что при изменении значения первичного ключа родительской таблицы такое же изменение будет выполнено и для значения внешнего ключа в дочерней таблице.

Иными словами, конфликты будут возникать всегда, когда в рамках транзакции производятся попытки изменения (редактирования или удаления) любой строки любой таблицы (главной или подчиненной), вовлеченной в операцию изменения в другой транзакции.

Теперь проделаем такую штуку. Закроем набор данных во второй программе. В первой программе изменим какую-нибудь запись, не подтверждая транзакции. Откроем набор данных во второй программе. Все правильно — вторая транзакция видит старую, неизмененную версию этой записи. Теперь измените характеристики транзакции во второй программе. Уберите из ее характеристик параметр isc_tpb_rec_version и замените его на isc_tpb_no_rec_version. Запустите транзакцию во второй программе и попытайтесь открыть набор данных. Вы тут же получите исключение.

Все дело в параметре isc_tpb_no_rec_version. Он требует, чтобы было выполнено подтверждение всех измененных другими транзакциями данных в используемых нашей транзакцией таблицах. На самом деле все гораздо хуже — если при нормально выполняющейся второй программе вы измените любую строку в первой программе, не подтверждая транзакции, а во второй программе выполните откат вашей транзакции, а затем попытаетесь переоткрыть набор данных, вы получите то же исключение.

Вывод второй. При использовании длинных транзакций или при наличии большого количества клиентов, работающих с вашей базой данных, не следует задавать параметр `isc_tpb_no_rec_version` совместно с `isc_tpb_nowait`.

> ***ВНИМАНИЕ!***
> Параметр `isc_tpb_no_rec_version` применяется по умолчанию. Вам нужно явно задавать в списке параметров `isc_tpb_rec_version`.

Интересные результаты можно получить, задавая параметр `isc_tpb_no_rec_version` вместе с `isc_tpb_wait`.

Вы помните, когда для транзакции задается параметр `isc_tpb_rec_version` вместе с `isc_tpb_wait`, то при возникновении конфликта вторая программа перейдет в состояние ожидания, а при подтверждении транзакции в первой программе вторая программа выдаст исключение.

Однако в случае задания для транзакции параметров `isc_tpb_no_rec_version` и `isc_tpb_wait` и после отката, и после подтверждения транзакции в первой программе вторая программа не выдает исключения. Эта особенность позволяет резко сократить или даже свести к нулю вероятность конфликта блокировок при многопользовательской работе с базой данных. Подробнее использование таких характеристик транзакций мы рассмотрим далее при обсуждении разделенных транзакций.

Проверим результат задания параметра `isc_tpb_read`. Установите для транзакции этот параметр, убрав `isc_tpb_write`, и попытайтесь выполнить какое-либо изменение данных. Вы тут же получите исключение. Транзакция с таким параметром, как и следовало ожидать, действительно не позволяет изменять данные.

Теперь одновременно заклиним обе программы — создадим настоящую "смертельную блокировку" (deadlock, или взаимную блокировку), когда первая программа ожидает подтверждения или отмены транзакции второй программы, а вторая программа ожидает того же от первой программы.

В обеих программах установите следующие характеристики транзакций:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_wait
isc_tpb_rec_version
```

Запустите транзакции и откройте наборы данных. Во второй программе выполните изменение в одной строке. В первой программе — изменения в *другой* строке. После этого во второй программе попытайтесь изменить ту же строку, которая была изменена в первой программе. В этот момент вторая программа перейдет в режим ожидания, поскольку среди параметров ее тран-

закции присутствует `isc_tpb_wait`, задающий режим ожидания разрешения конфликта блокировки с другой транзакцией. Затем в первой программе измените строку, которая была изменена второй программой. Первая программа также перейдет в режим ожидания. В этот момент и происходит действительно "смертельная" взаимная блокировка, deadlock, когда обе программы ожидают друг друга.

Однако на самом деле ничего страшного не происходит. Через несколько секунд первая программа выдает исключение со следующим сообщением: `deadlock. update conflicts with concurrent update` (взаимная блокировка, изменение конфликтует с параллельным изменением). Сервер базы данных имеет средства для выявления ситуаций взаимных блокировок. Анализ блокировок осуществляет Менеджер блокировок (Lock Manager). Для уменьшения накладных расходов и повышения производительности Менеджер не постоянно отслеживает ситуацию, а периодически запускается через определенное количество секунд. Интервал времени запуска Менеджера блокировок задается параметром `DeadlockTimeout` в файле конфигурации firebird.conf для Firebird 1.5 или параметром `DEADLOCK_TIMEOUT` в файле конфигурации ibconfig для InterBase. Значением по умолчанию является 10 секунд.

Теперь проделайте последний эксперимент. Отмените обе транзакции и переоткройте наборы данных в обеих программах. В первой программе измените какую-нибудь запись и подтвердите транзакцию. Во второй программе измените *ту же* запись и также подтвердите транзакцию. Эта операция, естественно, пройдет успешно. Переоткройте набор данных в обеих программах. Как и следовало ожидать, в базе данных сохранились изменения *только последней операции*.

> ***Напоминание***
>
> Для того чтобы транзакция с уровнем изоляции `READ COMMITTED` увидела подтвержденные изменения других транзакций, достаточно выполнить только переоткрытие набора данных (в FIBPlus для этого обычно используется метод набора данных `FullRefresh`). Останавливать и вновь запускать транзакцию не нужно.

8.4.2. Уровень изоляции *SNAPSHOT*

Это уровень изоляции транзакции по умолчанию. Он позволяет видеть неизменное состояние базы данных на момент старта транзакции. Изменения, выполненные другими транзакциями, в этой транзакции не видны. Свои изменения транзакция, разумеется, видит.

В первой программе сохраните уровень изоляции `READ COMMITTED`, задав для транзакции параметры:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_rec_version
isc_tpb_nowait
```

Во второй программе установите следующие параметры транзакции:

```
isc_tpb_write
isc_tpb_concurrency
isc_tpb_nowait
```

Здесь транзакция будет иметь уровень изоляции `SNAPSHOT`.

Измените в первой программе какую-нибудь строку, но не подтверждайте транзакцию. Во второй программе попытайтесь изменить ту же самую строку. Как и ожидалось, такое изменение вызывает ошибку блокировки.

Теперь измените в первой программе любую строку и подтвердите транзакцию. Во второй программе закройте набор данных и снова его откройте. Видно, что ничего не видно. То есть вторая программа с транзакцией `SNAPSHOT` не видит выполненных и подтвержденных в другой транзакции изменений. Так и должно быть, поскольку этот уровень изоляции дает нам мгновенный снимок базы данных на момент старта транзакции, этот снимок не может быть изменен другими параллельными транзакциями. Чтобы увидеть во второй программе изменения, выполненные другими транзакциями, следует остановить транзакцию, заново ее запустить и открыть набор данных.

Еще одна интересная проверка. Измените в первой программе любую строку и подтвердите транзакцию. Во второй программе (она не видит изменений, выполненных первой программой) попытайтесь изменить ту же (подтвержденную!) строку. Вы тут же получите исключение. Подозреваю, что для многих пользователей такое поведение программы будет, мягко сказать, несколько необычным.

Пожалуй, об этом уровне изоляции пока больше сказать нечего. Если вы не хотите ничего знать о внешнем мире, — какие были выполнены изменения в базе данных после старта вашей транзакции — или желаете получать исключения буквально на пустом месте, смело используйте этот уровень изоляции.

8.4.3. Уровень изоляции *SNAPSHOT TABLE STABILITY*

Изолированный, упорядочиваемый или сериализуемый образ. Аналогичен уровню `SNAPSHOT` с той лишь разницей, что в данном случае другие транзакции независимо от их уровня изоляции могут только читать данные таблиц, включенных в операции этой транзакции, но не могут их изменять.

Измените во второй программе характеристики транзакции. Создайте следующий список параметров:

```
isc_tpb_write
isc_tpb_nowait
isc_tpb_consistency
```

Запустите в ней транзакцию и откройте набор данных.

Попытайтесь в первой программе изменить или удалить любой элемент данных как в родительской, так и в дочерней таблице. Эта попытка приведет к исключению.

Такой уровень изоляции можно назвать приблизительно следующим образом: "по-настоящему работаю только я и больше никто". Однако здесь есть один подводный камень. Если при запуске транзакции с уровнем изоляции `SNAPSHOT TABLE STABILITY` и открытии в ее контексте набора данных другая параллельная транзакция уже изменила данные в том же наборе данных и еще не подтвердила эти изменения, то открытие набора данных в нашей транзакции приведет к исключению.

Проверим это на практике. Остановите транзакцию во второй программе. В первой программе измените любую строку в таблице стран. Сейчас это можно выполнить, поскольку вторая транзакция неактивна. После этого запустите транзакцию во второй программе и попытайтесь открыть набор данных. Вы тут же получите исключение `lock conflict on no wait transaction`. Здесь следует остановить транзакцию во второй программе, подтвердить или отменить изменения в первой программе и снова запустить транзакцию и открыть набор данных во второй программе.

Любопытная картина получится, если после появления этого сообщения об ошибке сначала подтвердить изменения в первой программе, а затем щелкнуть по кнопке **OK** в сообщении об ошибке во второй программе. Появившийся в сетке набор данных не будет содержать изменений, выполненных и не подтвержденных первой программой на момент старта второй программы! Да, при написании обработчиков событий следует проявлять большую осторожность.

8.4.4. Средства резервирования таблиц

Уровень `SNAPSHOT TABLE STABILITY` задает резервирование используемых в такой транзакции таблиц. Существуют дополнительные средства, позволяющие в транзакции задать резервирование отдельных таблиц или, наоборот, при уровне `SNAPSHOT TABLE STABILITY` разрешать изменения отдельных таблиц другими транзакциями. В SQL это предложение резервирования (`RESERVING`) — см. синтаксис оператора `SET TRANSACTION` в начале этой гла-

вы. В буфере параметров транзакции — это константы `isc_tpb_lock_read`, `isc_tpb_lock_write`, `isc_tpb_exclusive`, `isc_tpb_shared`, `isc_tpb_protected`. Рассмотрим возможности их использования на практике.

Резервирование таблиц на уровне изоляции *READ COMMITTED*

Запустите два экземпляра программы. В первой установите следующие параметры транзакции:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_nowait
isc_tpb_rec_version
isc_tpb_lock_write=REFCTR
isc_tpb_protected
```

> ***ВНИМАНИЕ!***
>
> Параметр `isc_tpb_protected` должен располагаться сразу после имени резервируемой таблицы.

В языке SQL такому набору параметров соответствует следующий оператор:

```
SET TRANSACTION
  READ COMMITTED
  READ WRITE
  NO WAIT
  RECORD_VERSION
  RESERVING REFCTR FOR PROTECTED WRITE;
```

Само резервирование задается в последней строке этого оператора.

Для второй программы задайте стандартный набор параметров для транзакции `READ COMMITTED`:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_nowait
isc_tpb_rec_version
```

Запустите транзакции и откройте наборы данных в обеих программах. После этого попытайтесь изменить любую строку во второй программе. Вы получите исключение по блокировке, поскольку таблица `REFCTR` зарезервирована.

Теперь остановите транзакции в программах и закройте наборы данных. Запустите транзакцию в первой программе, но не открывайте набор данных. Во второй программе запустите транзакцию, откройте набор данных и попытай-

тесь выполнить изменения. Вы опять получите исключение. Мы видим, что резервирование таблиц выполняется не в момент открытия набора данных, как это происходит при уровне изоляции `SNAPSHOT TABLE STABILITY`, а в момент запуска транзакции.

Замените в первой транзакции параметр `isc_tpb_protected` на `isc_tpb_exclusive`. Результат будет таким же.

Если заменить в первой транзакции `isc_tpb_lock_write=REFCTR` на `isc_tpb_lock_read=REFCTR`, то получим точно такой же результат. Если же установить `isc_tpb_shared`, то фактически никакого резервирования при использовании уровня изоляции `READ COMMITTED` не будет.

Можно убедиться, что такие же результаты будут получены, если во второй программе выбрать уровни изоляции `SNAPSHOT` и `SNAPSHOT TABLE STABILITY`. В последнем случае будет невозможным корректировать данные и в первой программе.

Таким образом, средства резервирования для транзакции `READ COMMITTED` позволяют предотвратить изменение зарезервированной таблицы другой транзакцией с любым уровнем изоляции.

Резервирование таблиц на уровне изоляции *SNAPSHOT*

Установите для первой программы следующие характеристики транзакции:

```
isc_tpb_write
isc_tpb_concurrency
isc_tpb_nowait
isc_tpb_lock_read=REFCTR
isc_tpb_exclusive
```

Проведите те же эксперименты, что и в случае `READ COMMITTED`. Результаты будут в точности такими же.

Резервирование таблиц на уровне изоляции *SNAPSHOT TABLE STABILITY*

Уровень изоляции `SNAPSHOT TABLE STABILITY` не позволяет другим транзакциям изменять данные в используемых таблицах. Однако средства резервирования дают возможность разрешить такие изменения для отдельных таблиц.

В первой программе задайте характеристики транзакции:

```
isc_tpb_write
isc_tpb_nowait
isc_tpb_lock_write=REFCTR
isc_tpb_shared
isc_tpb_consistency
```

Если вторая транзакция имеет уровень изоляции READ COMMITTED или SNAPSHOT, то ей предоставляется возможность совместного изменения данных указанной таблицы. Одновременное изменение данных одной и той же строки и в этом случае, разумеется, также невозможно.

Запустите транзакции и откройте наборы данных. Во второй программе вы можете выполнять изменения в таблице, для которой указано совместное использование (isc_tpb_shared).

> ***Напоминание***
> Параметры isc_tpb_shared, isc_tpb_exclusive и isc_tpb_protected должны следовать сразу же за параметром isc_tpb_lock_write (isc_tpb_lock_read). Имя любой таблицы не должно появляться более одного раза в конструкциях резервирования.

8.4.5. Использование разделенных транзакций

Компоненты FIBPlus позволяют при работе с наборами данных использовать одновременно две транзакции — одну для чтения, другую для изменения данных. В компонентах IBX это невозможно. Посмотрим, какие характеристики этих транзакций допустимы для использования в программах.

Создайте на базе существующего проекта новый проект Transaction2. Внесите изменения в программу. Добавьте еще один компонент транзакции с именем Transaction2. Ее мы будем использовать как транзакцию для обновления. В компоненте базы данных и в обоих наборах данных задайте транзакцию для обновления Transaction2 (в Database это свойство DefaultUpdateTransaction, в наборах данных — свойство UpdateTransaction). В наборах данных установите в True свойство AutoCommit и подсвойство poStartTransaction свойства Options. Этим мы создаем короткую транзакцию обновления. Она будет запускаться каждый раз, когда программа будет отправлять в базу данных изменения (операция Post) и тут же будет автоматически подтверждаться при отсутствии ошибок базы данных.

Для второй транзакции установите свойство TPBMode в значение tpbDefault. Установите следующие значения в окне TRParams:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_nowait
isc_tpb_rec_version
```

Запустите программу и задайте характеристики первой транзакции:

```
isc_tpb_read
isc_tpb_concurrency
isc_tpb_nowait
```

Это соответствует уровню изоляции SNAPSHOT. Запустите транзакцию, откройте набор данных и измените любую строку. Щелкните мышью по любой другой строке, чтобы выполнить операции Post и Commit для измененной записи. В строке появится старое, неизмененное значение столбца! Все правильно. Наша транзакция для чтения, Transaction1, не видит изменений, выполненных в той же программе транзакцией для обновления Transaction2. Даже если вы закроете набор данных и заново его откроете, ничего не изменится. Нужно остановить транзакцию, запустить ее заново и открыть набор данных. Только тогда в программе мы увидим новое, измененное значение столбца. Транзакция для чтения не видит изменений другой транзакции, пусть даже они обе находятся в одной программе.

Это еще не самый плохой вариант. Можно сделать хуже. Создайте следующие характеристики транзакции для чтения:

```
isc_tpb_read
isc_tpb_nowait
isc_tpb_consistency
```

Теперь наша транзакция для чтения имеет уровень изоляции SNAPSHOT TABLE STABILITY, что не позволяет другим транзакциям изменять используемые нашей транзакцией таблицы. Запустите транзакцию, переоткройте набор данных и попытайтесь изменить какую-нибудь строку. Вы тут же получите конфликт блокировки.

Конечно же, нормальным для читающей транзакции является набор следующих параметров:

```
isc_tpb_read
isc_tpb_nowait
isc_tpb_read_committed
isc_tpb_rec_version
```

Кроме того, она не должна выполнять никакого резервирования используемых в программе таблиц.

А для пишущей транзакции можно выбирать уже любой подходящий для поставленной задачи уровень изоляции и дополнительные характеристики.

Наличие отдельной долгой читающей транзакции, имеющей режим только для чтения (READ ONLY или isc_tpb_read), позволяет существенно экономить ресурсы сервера базы данных.

8.4.6. Преимущества использования компонентов FIBPlus

Мы потратили много сил и времени, чтобы получить себе неприятности в виде блокировок в программах, одновременно работающих с одной и той же базой данных. Теперь настала пора воспользоваться всеми преимуществами компонентов FIBPlus для создания "правильных" программ, уменьшающих вероятность блокировок.

Правильное использование разделенных транзакций

Запустите на выполнение проект Transaction2. Установите следующие характеристики транзакции чтения (`Transaction1`):

```
isc_tpb_read
isc_tpb_read_committed
isc_tpb_nowait
isc_tpb_rec_version
```

Это длинная транзакция только для чтения с уровнем изоляции `READ COMMITTED`. Одно из достоинств такой транзакции — минимальное использование ресурсов сервера.

Транзакция обновления (`Transaction2`) является короткой транзакцией. Она запускается при вызове метода `Post`, отправляющего изменения на сервер, и автоматически подтверждается (при отсутствии ошибок обновления) после завершения выполнения метода `Post`. Момент запуска и подтверждения транзакции можно уловить, написав обработчики событий у компонента транзакции `Transaction2`: `AfterStart` или `BeforeStart` для фиксации старта и `AfterEnd` (`BeforeEnd`) для завершения транзакции. В нашей программе при этих событиях выдаются соответствующие сообщения (вы можете убрать комментарии для реальной выдачи сообщений).

Запустите второй экземпляр программы, задав те же характеристики читающей транзакции. Теперь попробуйте получить блокировку, выполняя одновременное изменение одной и той же записи. На одном компьютере этого вам точно не удастся сделать. Использование в такой конфигурации короткой транзакции для обновления не оставляет никаких шансов получить на локальном компьютере ошибок блокировки. В локальной сети при многократных попытках одновременного изменения одиннадцатью клиентами одной и той же записи в базе данных, расположенной на сервере, при большом старании и тренировке в одновременности перехода к новой строке для подтверждения транзакции мы получали от нуля до трех блокировок.

Есть возможность полностью исключить конфликты обновления при использовании разделенных транзакций. Для этого обновляющей транзакции следует задать следующие характеристики:

```
isc_tpb_write
isc_tpb_read_committed
isc_tpb_wait
isc_tpb_no_rec_version
```

При возникновении конфликта блокировки с параллельным процессом транзакция перейдет в режим ожидания. Ожидание не будет слишком долгим, поскольку у конкурирующего процесса обновляющая транзакция короткая. После того как параллельный процесс отменит или подтвердит свою транзакцию, наш ожидающий процесс выполнит свою работу. В базе данных, естественно, сохранится только последнее изменение данных.

Подобную ситуацию мы проверили на тех же одиннадцати компьютерах в сети. Конфликта мы не получили. Хорошо это или плохо, зависит от конкретной задачи обработки данных.

> ***Замечание***
>
> Разумеется, если конфликт произошел, когда один процесс удалил запись, а другой пытается изменять удаленную запись, то неизбежно будет выдано исключение.

Таким образом, грамотное использование разделенных транзакций резко сокращает вероятность получения ошибок блокировки.

Использование защищенного режима в компонентах FIBPlus

Вернемся к первому проекту, созданному с использованием компонентов FIBPlus. В наборе данных `DataSetCountry` в свойстве `Option` установим в `True` значение подсвойства `poProtectedEdit`. Это позволяет задавать блокировку одной записи, для которой начинается операция изменения.

Запустим на выполнение два экземпляра программы. Установим в обоих экземплярах для транзакции уровень изоляции `READ COMMITTED`. Начнем изменять запись в одной программе, не подтверждая транзакцию. Затем начнем изменять ту же запись во второй программе.

Естественно, мы получим блокировку, однако ошибка в этом случае появится не при попытке во второй программе подтвердить транзакцию (как было в наших первых экспериментах), а сразу же, как только вы попытаетесь изменить хотя бы один символ в этой записи. Благодаря этому мы можем избежать такой неприятной ситуации, когда пользователь в течение достаточно длительного времени корректирует какую-то запись в базе данных, содержа-

щую большое количество столбцов, а при попытке поместить изменения в базу данных узнает, что запись нельзя было корректировать, поскольку другой пользователь занят тем же самым.

Здесь в компонентах FIBPlus реализован механизм так называемых фиктивных изменений. Смысл его в следующем. Чтобы заблокировать ровно одну запись для изменения параллельными процессами, программа выдает оператор `UPDATE`, который ничего не изменяет в записи (точнее, одному столбцу присваивает то же самое значение). Сервер создает новую версию записи, в которой нет отличий от последней подтвержденной версии, и задает блокировку этой строки. В этом случае другие процессы не могут изменять эту же запись, пока не будет подтверждена транзакция.

> ***Замечание***
>
> Следует отметить, что количество защищаемых таким образом записей не ограничено — как только вы приступаете к изменению новой записи в рамках одной транзакции, эта запись тут же становится защищенной. Не существует никаких средств снятия блокировок записей, кроме как подтвердив или отменив транзакцию.

8.4.7. Вложенные транзакции

В Firebird существует средство, которое называется "вложенными транзакциями" (nested transactions) или, более правильно, пользовательскими точками сохранения транзакции (savepoint). Такое же средство появилось только в InterBase 2007. В более ранних версиях это средство отсутствовало.

Это средство позволяет для длинной транзакции последовательно создавать промежуточные точки, в которых фиксируется текущее состояние базы данных. Этим точкам сохранения программа присваивает имена. В любой момент времени до завершения транзакции (ее подтверждения или отката) можно восстановить состояние базы данных на любую из созданных точек сохранения. При этом предыдущие точки сохранения остаются, а текущая, на которую выполнен откат, и все последующие аннулируются.

Вообще говоря, с точки зрения программиста здесь все очень просто. Для создания точки сохранения используется оператор SQL `SAVEPOINT`:

```
SAVEPOINT <идентификатор>;
```

`<Идентификатор>` — любое правильное имя объекта базы данных, длиной до 31 символа (в InterBase, наверное, до 67 символов, в документации я подробностей не нашел). Если точка сохранения с тем же именем уже существует, то она заменяется на новую. То есть старая точка сохранения удаляется, и создается новая с тем же именем.

В FIBPlus для этого используется метод компонента транзакции `SetSavePoint`:

```
SetSavePoint(<идентификатор>);
```

Для возврата (отката) на конкретную точку сохранения в SQL используется следующий оператор:

```
ROLLBACK [WORK] TO [SAVEPOINT] <идентификатор>;
```

В FIBPlus используется метод компонента транзакции `RollBackToSavePoint`:

```
RollBackToSavePoint(<идентификатор>);
```

Для освобождения (удаления) существующей точки сохранения используется SQL-оператор `RELEASE SAVEPOINT`:

```
RELEASE SAVEPOINT <идентификатор> [ONLY];
```

Если не указано ключевое слово `ONLY`, то будут освобождены (удалены) и потеряны все точки сохранения, начиная с указанной.

В FIBPlus используется метод компонента транзакции `ReleaseSavePoint`:

```
ReleaseSavePoint(<идентификатор>);
```

Этот метод всегда освобождает все точки, начиная с указанной.

Для иллюстрации работы с точками сохранения создадим программу с использованием компонентов FIBPlus, компоненты IBX не поддерживают средства создания точек сохранения. Напомню, что такая программа может работать с Firebird версии не ниже 1.5 или InterBase 2007.

Написание программы

Создадим новый проект (рис. 8.8). Положим на форму панель и выровняем ее по верхнему краю. Это будет панель инструментов. Разместим на ней кнопку завершения работы, выпадающий список `ComboBox` для хранения имен созданных точек сохранения и пять кнопок `TButton` — создания точки сохранения (текст на кнопке **Add**), отката транзакции на точку сохранения (**Rollback**), подтверждения транзакции (**Commit**), освобождения всех точек сохранения (**Release**) и создания точки с тем же именем (**AddExist**).

Положим (красоты ради) `StatusBar`, где будем указывать количество записей стран, `DBGrid` и `DataSource`. Добавим компоненты FIBPlus для работы с базой данных — `TpFIBDatabase`, `TpFIBTransaction`, `TpFIBDataSet`. Обычным образом для набора данных установим значения операторов SQL для работы с таблицей стран `REFCTR`, сделаем для него длинную транзакцию, т. е. не будем устанавливать в `True` значение свойства `AutoCommit`.

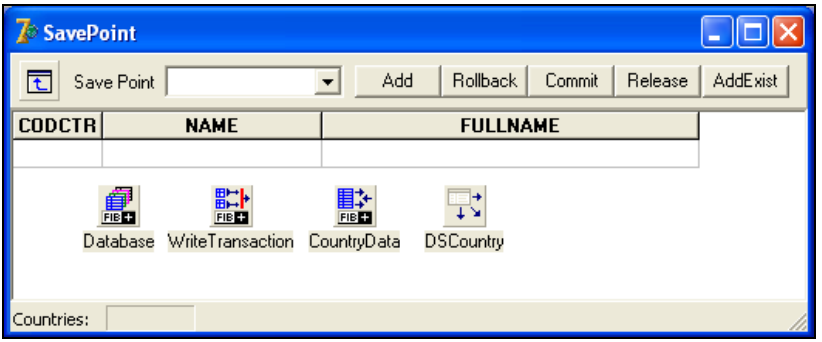


Рис. 8.8. Программа работы с точками сохранения транзакции

Главное в этом проекте — написание обработчиков. Для начала опишем в области **private** переменную SavePoint:

```
SavePoint: Integer;
```

В ней будет храниться текущий номер точки сохранения.

Обработчик события OnShow для формы выглядит следующим образом — листинг 8.12.

Листинг 8.12. Обработчик отображения формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin
  Database.Connected := True;
  WriteTransaction.StartTransaction;
  CountryData.Open;
  SavePoint := 0;
  StatusBar1.Panels.Items[1].Text := IntToStr(CountryData.RecordCount);
  DBGrid1.SetFocus;
end;
```

При завершении работы программы (событие OnClose) достаточно закрыть набор данных и отсоединиться от базы данных. По умолчанию для незавершенной транзакции будет выполнен откат.

Напишем следующий обработчик щелчка по кнопке добавления точки сохранения (**Add**) — листинг 8.13.

Листинг 8.13. Добавление новой точки сохранения

```
procedure TFormMain.BSAddClick(Sender: TObject);
var NewPoint: string;
begin
```

```
  SavePoint := SavePoint + 1;
  NewPoint := 'SavePoint' + IntToStr(SavePoint);
  WriteTransaction.SetSavePoint(NewPoint);
  CSavePoints.Items.Add(NewPoint);
  CSavePoints.ItemIndex := CSavePoints.Items.Count – 1;
  DBGrid1.SetFocus;
end;
```

Здесь создается имя точки сохранения (оно должно быть уникальным в рамках выполнения транзакции), это имя заносится в список `ComboBox`. Оператор создания точки сохранения:

```
WriteTransaction.SetSavePoint(NewPoint);
```

Выполнение отката транзакции на точку сохранения, выбранную пользователем в списке `ComboBox`, осуществляется при щелчке по кнопке отмены (**Rollback**) — листинг 8.14.

Листинг 8.14. Откат на точку сохранения

```
procedure TFormMain.BSRollbackClick(Sender: TObject);
var NewPoint: string;
    i, NewIndex: Integer;
begin
  if CSavePoints.ItemIndex < 0 then exit;
  NewIndex := CSavePoints.ItemIndex – 1;
  NewPoint := CSavePoints.Items.Strings[CSavePoints.ItemIndex];
  WriteTransaction.RollBackToSavePoint(NewPoint);
  CountryData.FullRefresh;
  for i := CSavePoints.Items.Count – 1 downto NewIndex + 1 do
    CSavePoints.Items.Delete(i);
  CSavePoints.ItemIndex := NewIndex;
  if NewIndex = -1 then CSavePoints.Clear;
  SavePoint := CSavePoints.Items.Count;
  StatusBar1.Panels.Items[1].Text := IntToStr(CountryData.RecordCount);
  DBGrid1.SetFocus;
end;
```

Собственно оператор отката следующий:

```
WriteTransaction.RollBackToSavePoint(NewPoint);
```

Остальные операторы лишь наводят порядок в списке имен точек сохранения.

При выполнении подтверждения транзакции (щелчок мышью по кнопке **Commit**) необходимо также выполнить и приведение в начальное состояние списка точек сохранения, т. е. нужно удалить все точки сохранения из списка,

поскольку после подтверждения транзакции они уже не действительны (листинг 8.15).

Листинг 8.15. Откат на точку сохранения

```
procedure TFormMain.BSCommitClick(Sender: TObject);
begin
  WriteTransaction.CommitRetaining;
  CountryData.FullRefresh;
  CSavePoints.Items.Clear;
  CSavePoints.ItemIndex := -1;
  SavePoint := 0;
  DBGrid1.SetFocus;
end;
```

Операция по освобождению точек сохранения (щелчок мышью по кнопке **Release**) похожа на операцию отката (листинг 8.16).

Листинг 8.16. Освобождение (удаление) всех точек сохранения

```
procedure TFormMain.BSReleaseClick(Sender: TObject);
var NewPoint: string;
    i, NewIndex: Integer;
begin
  if CSavePoints.ItemIndex < 0 then exit;
  NewIndex := CSavePoints.ItemIndex – 1;
  NewPoint := CSavePoints.Items.Strings[CSavePoints.ItemIndex];
  WriteTransaction.ReleaseSavePoint(NewPoint);
  CountryData.FullRefresh;
  for i := CSavePoints.Items.Count – 1 downto NewIndex + 1 do
    CSavePoints.Items.Delete(i);
  CSavePoints.ItemIndex := NewIndex;
  if NewIndex = -1 then CSavePoints.Clear;
  SavePoint := CSavePoints.Items.Count;
  DBGrid1.SetFocus;
end;
```

Поскольку в нашем списке `ComboBox` точки сохранения располагаются в порядке их создания, то при создании точки сохранения с тем же именем нужно удалить из списка соответствующую строку и в конец списка добавить имя точки сохранения. Это выполняется при щелчке по кнопке **AddExist** (листинг 8.17).

Листинг 8.17. Добавление в список точки сохранения с уже существующим именем

```
procedure TFormMain.BSAddExistClick(Sender: TObject);
var NewPoint: string;
begin
  if CSavePoints.ItemIndex < 0 then exit;
  NewPoint := CSavePoints.Items.Strings[CSavePoints.ItemIndex];
  CSavePoints.Items.Delete(CSavePoints.ItemIndex);
  WriteTransaction.SetSavePoint(NewPoint);
  CSavePoints.Items.Add(NewPoint);
  CSavePoints.ItemIndex := CSavePoints.Items.Count – 1;
  DBGrid1.SetFocus;
end;
```

Ну и в плане красоты необходимо написать обработчик события удаления строки, чтобы скорректировать отображаемое на панели `StatusBar` количество стран. Это можно выполнить в обработчике события `AfterDelete` (после удаления) — листинг 8.18.

Листинг 8.18. Обработка события после удаления строки таблицы

```
procedure TFormMain.CountryDataAfterDelete(DataSet: TDataSet);
begin
  StatusBar1.Panels.Items[1].Text := IntToStr(CountryData.RecordCount);
end;
```

Работа сделана. Запустите программу на выполнение. Создавайте точки сохранения в произвольных количествах, выполняйте изменение и удаление данных, возвращайтесь на любую созданную точку сохранения, освобождайте различные точки сохранения, создавайте точки с теми же именами, что уже существуют в списке. Все работает прекрасно.

Поскольку для пользовательских точек сохранения используется длинная обновляющая транзакция, применять это средство естественнее в монопольном режиме с уровнем изоляции транзакции `SNAPSHOT TABLE STABILITY`.

8.5. Краткие итоги

Здесь мы довольно подробно рассмотрели основные вопросы использования транзакций в базах данных InterBase/Firebird с учетом их различий.

Из трех уровней изоляции транзакций для одновременной многопользовательской работы больше всего подходит `READ COMMITTED`. При этом обновляющие транзакции должны быть короткими.

Уровень изоляции `SNAPSHOT` является довольно неудобным по причине высокой вероятности получения исключений по блокировкам.

Если же требуется получить монопольный доступ к отдельным таблицам базы данных, следует выбрать `SNAPSHOT TABLE STABILITY`. При этом необходимо помнить, что при запуске такой транзакции и попытке открыть наборы данных можно получить исключение по блокировке, если какая-либо параллельная транзакция выполнила изменения в таблице и не подтвердила транзакцию. При использовании коротких обновляющих транзакций вероятность такого исключения мала.

При использовании разделенных транзакций для чтения и для обновления читающая транзакция должна иметь уровень изоляции только `READ COMMITTED`, и никакой другой. Пишущая транзакция может иметь любой требуемый
условиями задачи уровень изоляции и соответствующий набор дополнительных характеристик.

Для обновляющих транзакций наиболее подходящими являются два варианта набора характеристик: использование режима `WAIT` совместно с `NO RECORD_VERSION` или `NO WAIT` совместно с `RECORD_VERSION`. В первом случае полностью исключаются ошибки блокировки после отмены или подтверждения блокирующей транзакции (ошибки будут, если транзакция пытается изменять запись, удаленную другим процессом). Во втором случае мы сразу получаем ошибку, после чего принимаем решение о дальнейших действиях.

Как правило, использование режима `NO RECORD_VERSION` не является хорошей идеей при достаточно большом количестве одновременно работающих пользователей. Если пользователи активно изменяют данные в базе данных, то может оказаться довольно затруднительным запустить на выполнение такую транзакцию. Если же вам действительно нужно получать самые свежие данные, самые последние изменения, этот режим будет самым подходящим.

При грамотном использовании в компонентах FIBPlus двух транзакций — на чтение и на обновление — можно повысить эффективность использования ресурсов сервера и практически свести к нулю вероятность осложнений в работе пользователя в случае появления блокировок.

Использование защищенного режима, предоставляемого компонентами FIBPlus, позволяет в особых случаях предотвратить попытки других пользователей изменять или удалять запись, которую начал корректировать один из клиентов.

Удобные средства "вложенных транзакций" позволяют раздробить длинную обновляющую транзакцию на фрагменты и выполнять откат не всей транзакции, а только некоторой ее части.

Здесь мы все время говорим об уменьшении вероятности блокировок. У кого-то может сложиться впечатление, что блокировки — это зло, с которым нужно бороться всеми доступными средствами. Конечно, это не так. Все зависит от конкретной решаемой задачи. Часто бывает нужным блокировать изменения в базе данных другими клиентами. В реальной жизни могут потребоваться как "жесткие" средства блокировки в виде уровня изоляции `SNAPSHOT TABLE STABILITY`, так и "мягкие" средства защищенного режима. Серверы InterBase/Firebird и компоненты обращения к базам данных FIBPlus и IBX имеют мощные гибкие средства решения любых задач обработки данных в архитектуре "клиент-сервер".

Замечание

На самом деле мы еще не все сказали про транзакции. В InterBase и Firebird существует удивительная способность транзакций работать с несколькими базами данных. Мы обязательно рассмотрим эти средства, но несколько позже, когда наберемся опыта в написании триггеров и хранимых процедур. Это будет *глава 10*, где мы в поте лица своего создадим несколько довольно сложных и очень полезных программ.

Что там за перевалом?

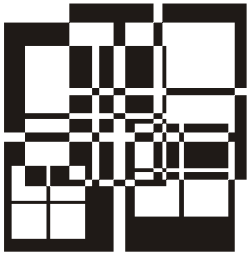
Транзакции — это действительно объемная и очень интересная часть реляционных баз данных InterBase и Firebird. Надеюсь, после проведенных в этой главе исследований у вас сформировались четкие знания о возможностях транзакций и сложились полезные практические навыки в их использовании.

Я, кончено, был неправ, когда сказал, что это все не так уж и сложно. Да, транзакции — это довольно сложный механизм. Однако сложность транзакций имеет исключительно количественный характер — уж очень много у них интересных возможностей. Содержательно, это очень естественная и понятная часть реляционных баз данных.

К транзакциям мы еще вернемся. А сейчас перейдем к изучению и практическому применению императивных процедурных возможностей реляционных баз данных. Мы займемся хранимыми процедурами и триггерами.

Это то, что позволит нам, программистам, реализовать наши императивные функциональные потребности по обработке данных.

ГЛАВА 9



Хранимые процедуры и триггеры

До этого момента мы использовали декларативные (то есть описательные) средства языка SQL. С помощью этих средств мы указывали, *что* должно быть сделано, но не сообщали системе, *как* это должно быть сделано. В языке SQL существуют и императивные, выполняемые, средства. Это языковые средства, используемые в хранимых процедурах и триггерах.

Для написания хранимых процедур и триггеров используется так называемое расширение языка SQL, язык хранимых процедур и триггеров.

Хранимые процедуры и триггеры являются программами. Они хранятся в области метаданных базы данных в системных таблицах и выполняются на стороне сервера, что во многих случаях может сильно сократить сетевой трафик, нагрузку сети.

Как правило, они выполняют какие-то действия с базой данных, в которой определены, однако это не является обязательным. Они могут выполнять и любые другие действия, никак не связанные с базой данных.

К хранимым процедурам могут обращаться любые программы, работающие с базой данных — хранимые же процедуры, триггеры, клиентские приложения.

К триггерам напрямую обращение невозможно. Они автоматически вызываются при наступлении одного из событий базы данных — добавление новой строки в таблицу, удаление строки или изменение существующей строки.

По порядку рассмотрим общие для этих программ языковые средства и сами программные элементы — триггеры и хранимые процедуры.

В литературе сейчас используется понятие бизнес-логики. Это набор правил, которым должны удовлетворять данные в одной или разных таблицах базы

данных. В первую очередь эти правила касаются формальной "чистоты" данных, например, соответствие внешних ключей первичным ключам. Однако существуют и содержательные, связанные с конкретной предметной областью правила, определяющие отношения между данными различных таблиц или различными строками одной и той же таблицы. Именно триггеры и хранимые процедуры позволяют автоматизировать процесс поддержания всех видов бизнес-логики.

9.1. Язык хранимых процедур и триггеров

Язык, используемый для написания процедур и триггеров, является полноценным языком программирования, он позволяет выполнить любые действия по обработке данных — как локальных, так и хранящихся в базе данных.

В языке допустимы практически любые действия с базой данных, за исключением создания и изменения метаданных. Можно использовать операторы работы с данными базы данных: `INSERT`, `UPDATE`, `DELETE` и `SELECT` (в двух вариантах). Только в триггерах и хранимых процедурах можно отправлять сообщения (events), вызывать пользовательские исключения (exception). Нельзя устанавливать и разрывать связь с базой данных, манипулировать транзакциями, открывать наборы данных.

При создании как процедур, так и триггеров в синтаксисе выделяются заголовок и тело (хранимой процедуры, триггера). Тело является *блоком операторов*.

Все действия описываются в блоке операторов между ключевыми словами `BEGIN` и `END`. Программы могут содержать произвольное количество блоков, как независимых, так и вложенных друг в друга.

В любое место программного кода мы можем поместить комментарий. Текст комментария располагается между символами `/*` и `*/`, как в языке C. Комментарий может занимать произвольное количество строк.

9.1.1. Использование оператора *SET TERM*

Как мы помним, любой оператор SQL заканчивается терминатором — символом точка с запятой (`;`). Все объявления и операции в хранимых процедурах и триггерах также заканчиваются этим символом.

Чтобы избежать двусмысленности, при создании триггера или хранимой процедуры используется оператор `SET TERM`, который устанавливает новое значение терминатора. После завершения описания программного компонента опять же с помощью оператора `SET TERM` вновь устанавливается старое значение терминатора точка с запятой.

Пример использования оператора:

```
SET TERM ^;
DECLARE PROCEDURE ...
...
END ^
SET TERM ;^
```

Первый оператор `SET TERM` устанавливает значение терминатора в `^`. После этого идут операторы создания процедуры (точно так же мы поступаем и когда создаем триггер), каждый оператор заканчивается как обычно точкой с запятой. Последний оператор процедуры `END` завершается символом `^`. Затем оператор `SET TERM` возвращает значение терминатора в символ точка с запятой.

> ***Замечание***
>
> Вообще говоря, `SET TERM` не является оператором языка SQL. Это средство, которое используется в программах, осуществляющих выполнение скриптов, например, isql, IBExpert и в других.

9.1.2. Использование переменных

Переменными в этом языке являются локальные переменные, которые могут быть описаны и в триггерах, и в процедурах, а также входные и выходные параметры, используемые только в хранимых процедурах. Назовем эти три группы переменных внутренними переменными.

При описании каждой переменной задается ее имя и тип данных. Имя каждой переменной должно быть уникальным среди имен внутренних переменных. Имена внутренних переменных могут совпадать с именами столбцов таблиц, с которыми работает триггер или хранимая процедура. Это не приведет к недоразумениям, поскольку имена столбцов используются только в операторах `SELECT`, `INSERT`, `UPDATE` и `DELETE`. В этих операторах именам внутренних переменных должен предшествовать символ двоеточия.

Типом данных при описании внутренней переменной может быть любой допустимый в базе данных тип, за исключением `BLOB`. Не допускается использование имен доменов для задания типа данных.

Локальные переменные описываются в заголовке триггера или процедуры. Синтаксис задания локальной переменной:

```
DECLARE VARIABLE <имя локальной переменной> <тип данных>;
```

Каждая локальная переменная задается отдельной строкой.

Можно задать любое количество локальных переменных для одного триггера или одной хранимой процедуры.

Внутренние переменные известны только в пределах того триггера или процедуры, где они определены. Другим программным объектам они не доступны.

Входные и выходные параметры используются только в хранимых процедурах. Они описываются в заголовке процедуры. Их использование аналогично локальным переменным. Синтаксис задания параметров мы рассмотрим чуть позже.

Переменные записываются в операторах как обычно. Исключение составляют лишь операторы обращения к базе данных: `INSERT`, `UPDATE`, `DELETE` и `SELECT`. В этих операторах для того, чтобы не спутать имена внутренних переменных с именами столбцов таблиц, перед именем переменной должно стоять двоеточие.

9.1.3. Выполняемые операции

Все действия в процедурах и триггерах выполняются в блоках операторов, которые заключены в так называемые операторные скобки — ключевые слова `BEGIN` и `END`. После этих слов не ставится терминатор точка с запятой.

В языке хранимых процедур и триггеров существует ряд выполняемых операций. После каждой операции ставится точка с запятой.

Присваивание значения

Важнейшей, фундаментальной, операцией в любом языке программирования является операция присваивания переменной конкретного значения. Здесь также используется эта операция. Ее синтаксис соответствует языку C:

```
<переменная> = <выражение>;
```

`<Переменная>` — локальная переменная, входной или выходной параметр.

`<Выражение>` — литерал или любое правильное выражение, содержащее литералы, имена переменных и знаки операции. Выражение возвращает значение, соответствующее типу данных переменной, которой присваивается значение. Может использовать арифметические, строковые и логические операции, встроенные в язык функции, например, `GEN_ID`, а также функции, определенные пользователем (UDF).

Простой пример присваивания:

```
variable1 = variable1 + 1;
```

Для строковых переменных мы можем использовать операцию конкатенации, соединения нескольких строк в одну, например:

```
FULLNAME = NAME3 || ' ' || NAME1 || ' ' || NAME2;
```

Для получения значения искусственного первичного ключа мы можем использовать встроенную функцию GEN_ID:

```
COD = GEN_ID(GENERATOR_1, 1);
```

В последних версиях InterBase и Firebird в большинстве случаев не выполняется неявное преобразование типов данных к нужному типу. По этой причине вам нужно использовать внутреннюю функцию CAST для выполнения соответствующих преобразований.

Например, нельзя выполнить такое присваивание числовой переменной:

```
I = I + '1';
```

Второй операнд нужно явно преобразовать к целому типу:

```
I = I + CAST('1' AS INTEGER);
```

> ***Замечание***
>
> Понятно, что последний оператор с точки зрения нормального программиста — это полный бред, хотя тоже работает. Это всего лишь иллюстрация применения преобразования типов функцией CAST. Операция становится осмысленной, если в качестве объекта преобразования типов мы выбираем строковый столбец, который может содержать только строку цифр.

Использование оператора *IF-THEN-ELSE*

Как и в любом нормальном языке программирования, в языке хранимых процедур и триггеров существуют средства *ветвления* процесса. Для этих целей используется только один оператор: IF-THEN-ELSE. Отсутствует, например, такой оператор, как привычный для нас по некоторым языкам программирования CASE. Синтаксис оператора ветвления:

```
IF (<условие>)
  THEN <составной оператор>
  [ELSE <составной оператор>];
```

Здесь проверяемое условие, как и в языке C, заключается в круглые скобки. Что лично меня радует.

Если условие является истинным (возвращает значение TRUE), то выполняется оператор или группа операторов (составной оператор, заключенный в операторные скобки BEGIN и END) после ключевого слова THEN. Иначе выполняется группа операторов после ключевого слова ELSE, если присутствует.

Опять хочу напомнить, что в реляционных базах данных столбцы могут иметь пустое значение. Если в операциях сравнения участвуют столбцы, которые могут принимать значение NULL, то результатом всегда будет неопределенное, неизвестное значение — NULL. В этом случае не будут выполняться операторы после ключевого слова THEN, а будет выполняться конструкция после ключевого слова ELSE (если присутствует).

Приведем пару примеров.

Пример проверки на равенство. Сравним значения двух числовых (целочисленных) переменных. Заодно опишем и все используемые здесь переменные.

```
...
DECLARE VARIABLE A INTEGER;
DECLARE VARIABLE B INTEGER;
DECLARE VARIABLE MESSAGE CHAR(20); //Можем использовать и VARCHAR
...
IF (A = B) THEN
MESSAGE = 'Блин! Они равны!';
ELSE
MESSAGE = 'Не-а. Не равны';
...
```

Это фрагмент хранимой процедуры, которую написал один мой студент. Формируемые сообщения я сохранил в том виде, как это было в оригинале.

Обратите внимание на то, что перед ключевым словом `ELSE` мы должны поставить точку с запятой, в отличие от языка Delphi (или Объектный Паскаль).

Пример посложнее и поинтереснее. В триггере, который мы с вами ранее рассматривали, есть такая проверка:

```
IF (NOT EXISTS(SELECT COD
        FROM PEOPLE
        WHERE COD = NEW.CODOTHERHALF)) THEN
  EXCEPTION NO_PEOPLE;
```

Здесь используется очень удобная и весьма полезная функция `EXISTS`. Точнее, ее отрицание. Если не существует соответствующей записи нужного нам человека, то выдается исключение (более подробно об исключениях чуть позже).

Оператор цикла *WHILE-DO*

Конечно же, здесь, в языке хранимых процедур и триггеров, присутствует и оператор цикла. Это оператор `WHILE-DO`. Его синтаксис:

```
WHILE (<условие>) DO
  <составной оператор>;
```

Опять напомню, `<составной оператор>` — это либо один оператор, либо группа операторов, заключенных в операторные скобки `BEGIN...END`.

Пример оператора цикла. Выполним расчет факториала целого числа. Пусть заданное число (это будет выполняться в хранимой процедуре) пе-

редается во входном параметре N. Результат вычисления должен помещаться в выходной параметр RESULT. Используется промежуточная целочисленная переменная I. Фрагмент программного кода:

```
RESULT = 1;
I = 1;
WHILE (I <= N) DO
BEGIN
  RESULT = RESULT * I;
  I = I + 1;
END
```

Этот простой алгоритм используется в выполняемой хранимой процедуре PROC_FACTORIAL, которую мы рассмотрим позже в этой главе.

Использование одиночного оператора *SELECT*

Мы можем использовать одиночный оператор SELECT для получения одной строки из одной или нескольких таблиц.

Такой оператор требует обязательного присутствия предложения INTO, которое должно быть последним в операторе. В предложении INTO указывается, в какие внутренние переменные должны помещаться считанные из таблицы данные. Всем именам внутренних переменных в этом операторе должен предшествовать символ двоеточия, чтобы отличить переменные от имен столбцов таблицы.

Для присваивания, например, локальным переменным (или входным или выходным параметрам) avg_salary и sum_salary средней зарплаты и суммы заработных плат всех сотрудников организации с кодом 11 мы можем использовать следующий оператор:

```
SELECT AVG(SALARY), SUM(SALARY)
  FROM STAFF
  WHERE CODORG = 11
INTO :avg_salary, :sum_salary;
```

Здесь оператор SELECT находит в базе данных среднюю и суммарную зарплату всех сотрудников организации 11 и помещает полученные значения в локальные переменные. Обратите внимание, что в этом операторе в предложении INTO перед именами локальных переменных стоит двоеточие.

Реально работающий пример из триггера, который мы с вами ранее рассматривали.

```
SELECT SEX, CODOTHERHALF
  FROM PEOPLE
  WHERE COD = NEW.CODOTHERHALF
```

```
INTO :SEXI, :CODI;
```

Здесь из конкретной одной строки таблицы (которая однозначно задается в предложении `WHERE`) читаются признак пола человека и код его супруга (супруги). Считанные значения помещаются в локальные переменные.

Опять же, перед именами локальных переменных в таком операторе мы обязательно должны поставить символ двоеточия, чего не нужно делать в обычных операторах присваивания значения переменным.

Оператор *FOR SELECT-DO*

Очень полезный оператор в хранимых процедурах выбора (см. дальше). Синтаксис:

```
FOR <оператор SELECT>
INTO <список переменных>
DO <составной оператор>;
```

С помощью этого оператора мы в хранимой процедуре можем выбирать произвольное количество строк из одной или более таблиц. Более подробно этот оператор мы рассмотрим несколько позже, после триггеров.

Оператор вызова хранимой процедуры *EXECUTE PROCEDURE*

Как уже было сказано, и триггеры, и хранимые процедуры могут вызывать хранимые процедуры. Для этого используется оператор `EXECUTE PROCEDURE`:

```
EXECUTE PROCEDURE <имя выполняемой хранимой процедуры>
[(<входной параметр> [, <входной параметр>] ...)]
[RETURNING_VALUES (<выходной параметр> [, <выходной параметр>] ...)];
```

В операторе задается имя вызываемой хранимой процедуры. Может задаваться список входных параметров.

Процедура может возвращать список выходных значений. В этом случае список выходных параметров следует задать в предложении `RETURNING_VALUES`.

> ***Замечания по синтаксису***
>
> Вообще говоря, по правилам синтаксиса список как входных, так и выходных параметров в этом операторе можно и не заключать в круглые скобки. Однако лично мне это очень не нравится. Как-то уж в нормальных языках программирования принято списки заключать в скобки и разделять элементы списка запятыми.

Оператор *EXIT*

Оператор `EXIT` вызывает передачу управления на конечный оператор `END`, т. е. приводит к завершению хранимой процедуры или триггера.

Радует, что среди операторов языка хранимых процедур и триггеров отсутствует оператор типа `GO TO`, который программисты часть называют "смерть структурному программированию".

Оператор *SUSPEND*

Этот оператор используется только в хранимых процедурах выбора. Он осуществляет временную приостановку выполнения хранимой процедуры и передачу управления вызвавшей программе. Подробнее `SUSPEND` мы рассмотрим чуть позже, когда будем говорить о хранимых процедурах выбора.

***

Помимо рассмотренных средств существуют средства обработки исключений базы данных и вызова событий. Каждому из этих средств мы посвятим следующие два небольших раздела.

9.2. Исключения базы данных

В процессе работы пользователя с реляционной базой данных могут, естественно, возникать ошибки, которые определяются сервером базы данных. Например, ошибками будут дублирование значения первичного или уникального ключа или несоответствие значения внешнего ключа первичному (уникальному) ключу родительской таблицы. Такие ситуации определяются самой системой управления базами данных. В результате выдаются соответствующие сообщения, формируются подходящие исключения и коды ошибок.

Существует другая группа ошибок, которая связана с *содержательным* аспектом использования базы данных. Эти ошибки связаны с конкретной предметной областью, в которой работает наша программная система.

Выявление таких ошибок возможно не на уровне формальных проверок использования реляционных баз данных, а на содержательном уровне, в процессе выполнения проблемных программ.

Для реализации подобного механизма реакции на нарушение содержательной целостности данных применяются пользовательские исключения базы данных. Это один из наиболее простых и в то же время элегантных механизмов, позволяющих программисту создавать необходимые содержательные формулировки исключений для ошибочных ситуаций.

Такие средства в полном объеме реализованы в системах управления базами данных InterBase и Firebird. Это пользовательские исключения (exception) базы данных.

9.2.1. Пользовательские исключения

Синтаксис создания пользовательского исключения:

```
CREATE EXCEPTION <имя исключения> '<текст сообщения>';
```

Вы можете создать произвольное количество исключений в вашей базе данных и использовать их по прямому назначению.

Вызов любого исключения в хранимой процедуре или в триггере приводит к прекращению выполнения текущей операции и, при возможности, к переходу к подпрограмме обработки ошибочной ситуации.

Хранимая процедура или триггер, обнаружив ошибочную ситуацию, может выдать исключение. Синтаксис оператора:

```
EXCEPTION <имя исключения>;
```

Если в самой процедуре или триггере, где было выдано исключение, не предусмотрена обработка ошибочных ситуаций, то ошибка возвращается вызвавшей программе или программе, инициировавшей вызов триггера. В результате чаще всего вызвавшая программа выдаст сообщение об ошибке, которое будет содержать текст сообщения, определенный при создании исключения.

Еще раз вернемся к рассмотренному ранее триггеру. В том же скрипте мы создали несколько исключений:

```
CREATE EXCEPTION SEX_PEOPLE_INSERT 'Неверный пол супруга при добавлении
записи';
CREATE EXCEPTION SEX_PEOPLE_UPDATE 'Неверный пол супруга при изменении
записи';
CREATE EXCEPTION NO_PEOPLE 'Запись человека отсутствует в базе данных';
CREATE EXCEPTION OTHER_PEOPLE 'У супруга установлена связь с другим
человеком';
```

А вот фрагмент текста триггера, выполняющий проверки и выдающий исключения в случае ошибок:

```
IF (NOT EXISTS(SELECT COD
                 FROM PEOPLE
                 WHERE COD = NEW.CODOTHERHALF)) THEN
   EXCEPTION NO_PEOPLE;
SELECT SEX, CODOTHERHALF FROM PEOPLE WHERE COD = NEW.CODOTHERHALF
  INTO :SEXI, :CODI;
IF (NEW.SEX = SEXI) THEN
  EXCEPTION SEX_PEOPLE_INSERT;
IF (CODI IS NOT NULL) THEN
  EXCEPTION OTHER_PEOPLE;
```

Полагаю, тут все понятно. Если не выполняются некоторые условия, то выдается соответствующее исключение.

Исключения хороши еще и тем, что в них мы можем задавать текст сообщения на языке, понятном нашему пользователю.

9.2.2. Обработка исключений и ошибок базы данных в триггерах и хранимых процедурах

Хранимые процедуры и триггеры имеют средства для обработки пользовательских исключений и ошибок базы данных. Это осуществляется при использовании оператора `WHEN`.

Синтаксис оператора:

```
WHEN {<ошибка> [, <ошибка>] ... | ANY}
DO <составной оператор>;
```

Здесь `<составной оператор>` обрабатывает указанную группу ошибок и пользовательских исключений.

Если после ключевого слова `WHEN` указано `ANY`, то обрабатывается любая возникшая ошибочная ситуация и пользовательское исключение.

Можно указать конкретный список ошибочных ситуаций и пользовательских исключений, которые будут обработаны этим оператором. Элементы в списке отделяются друг от друга запятыми.

В синтаксисе оператора `<ошибка>` может быть:

- `EXCEPTION <имя пользовательского исключения>;`
- `SQLCODE <код ошибки>;`
- `GDSCODE <номер>.`

Значения кодов `SQLCODE` и `GDSCODE` приведены в *приложении 3*.

> **Важный момент**
>
> Если ошибка обработана в хранимой процедуре или в триггере, то она не возвращается вызвавшей программе или программе, инициировавшей вызов триггера.

В триггере или хранимой процедуре может присутствовать произвольное количество операторов `WHEN`. Каждый должен располагаться в самом конце программного блока перед оператором `END`. Любое количество операторов `WHEN`, каждый из которых обрабатывает свою группу ошибок или пользовательских исключений, могут располагаться один за другим.

Примеры использования оператора.

Следующий оператор будет обрабатывать любую ошибку, связанную с ошибками авторизации доступа к базе данных:

```
WHEN SQLCODE -85
DO ...;
```

Коды ошибок SQLCODE описывают ошибочную ситуацию очень обобщенно. В частности, значение SQLCODE -85 имеет одиннадцать вариантов ошибок. Более детальную информацию об ошибке можно получить из значения кода GDSCODE.

В следующем примере будет обработана ошибка неверного указания имени пользователя:

```
WHEN GDSCODE 335544753
DO ...;
```

9.3. События базы данных

В базах данных InterBase и Firebird реализован механизм событий. Хранимые процедуры и триггеры (и только они) могут посылать сообщение о событии. Эти сообщения могут получать клиентские приложения, которые явно "прослушивают" соответствующие сообщения.

Синтаксис выдачи сообщения о событии:

```
POST_EVENT '<событие>';
```

Одно из применений событий — оповещение клиентов об изменении данных какой-либо таблицы. Триггер, который автоматически вызывается после выполнения изменения, выдает сообщение об этом событии. Здесь <событие> — произвольный текст. Примеры таких триггеров мы с вами рассмотрим в этой главе.

Сергей Востриков в этом месте сделал полезное замечание. События попадают в клиентское приложение только после подтверждения транзакции, в контексте которой выполняется соответствующий триггер или хранимая процедура.

9.4. Триггеры

Триггер — это программный объект базы данных, который хранится в области метаданных. Триггер выполняется на стороне сервера. Во многих случаях это позволяет повысить производительность системы.

Для поддержания ограничений CHECK и FOREING KEY система автоматически создает триггеры.

Напрямую обратиться к триггеру невозможно. Он вызывается автоматически при наступлении соответствующего события базы данных — добавление новой строки в таблицу, изменение или удаление строки. Триггер может срабатывать, когда соответствующее действие с базой данных выполняет клиент-

ское приложение, хранимая процедура или триггер (другой или тот же самый — здесь заложена прекрасная возможность получить бесконечный цикл).

Триггер выполняется в контексте той транзакции, под управлением которой выполняется и программа, инициировавшая вызов триггера.

Момент вызова триггера задается двумя элементами: событие базы данных и фаза события — до наступления события или после. В результате получается шесть вариантов вызова триггера:

- перед добавлением новой строки (`BEFORE INSERT`);
- сразу после добавления новой строки (`AFTER INSERT`);
- перед изменением строки (`BEFORE UPDATE`);
- после изменения строки (`AFTER UPDATE`);
- перед удалением строки (`BEFORE DELETE`);
- после удаления строки (`AFTER DELETE`).

9.4.1. Создание триггеров

Триггер создается оператором `CREATE TRIGGER`:

```
CREATE TRIGGER <имя триггера>
  FOR {<имя таблицы> | <имя представления>}
  [ACTIVE | INACTIVE]
  {BEFORE | AFTER} {INSERT | UPDATE | DELETE}
  [POSITION <целое>]
AS <тело триггера>;
```

Тело триггера определяется следующим образом:

```
<тело триггера> ::= [<список переменных>] <блок операторов>
```

Синтаксис описания списка локальных переменных:

```
<список переменных> ::=
  DECLARE VARIABLE <имя переменной> <тип данных>;
  [DECLARE VARIABLE <имя переменной> <тип данных>;] ...
```

Триггер может содержать произвольное количество локальных переменных. Каждая переменная должна быть определена в отдельной строке.

Последний элемент синтаксиса:

```
<блок операторов> ::=
BEGIN
  <составной оператор> [<составной оператор>] ...
END
```

```
<составной оператор> ::= {<блок операторов> | <оператор>}
```

В заголовке триггера указывается, для какой таблицы или представления создается триггер, задается, является ли триггер активным (`ACTIVE`), т. е. будет автоматически вызываться при наступлении события, или неактивным (`INACTIVE`). Задается событие и фаза вызова триггера, а также позиция (`POSITION`).

По умолчанию созданный триггер является активным.

Позиция указывается для случая, когда для одного и того же события задано несколько триггеров. По умолчанию устанавливается позиция 0. Триггеры будут вызываться в порядке возрастания номеров позиций. Если несколько триггеров для события имеют один и тот же номер позиции, то они будут выполняться еще и в алфавитном порядке имен триггеров.

Тело триггера содержит операторы языка хранимых процедур и триггеров. Все тело триггера заключается в операторные скобки `BEGIN` и `END`.

В расширении языка SQL при использовании в триггерах могут использоваться две так называемые контекстные переменные: `NEW` и `OLD`.

Контекстные переменные записываются в виде:

```
NEW.<имя столбца>
```

и

```
OLD.<имя столбца>
```

`NEW` ссылается на новое значение столбца. Может использоваться только в триггерах, реагирующих на операции добавления новой записи или изменения значения существующей. Не используется при удалении строки таблицы.

`OLD` ссылается на старое, до изменения, значение столбца. Используется при изменении и удалении записи. Не может использоваться при добавлении новых данных.

9.4.2. Удаление и изменение триггеров

Для удаления триггера используется оператор `DROP TRIGGER`:

```
DROP TRIGGER <имя триггера>;
```

Вы не можете удалить триггер, созданный системой для поддержания ограничений `CHECK` и `FOREIGN KEY`.

Для изменения триггера используется оператор `ALTER TRIGGER`, имеющий следующий синтаксис:

```
ALTER TRIGGER <имя триггера>
  [ACTIVE | INACTIVE]
  [{BEFORE | AFTER} {INSERT | UPDATE | DELETE}]
```

```
  [POSITION <целое>]
[AS <тело триггера>];
```

Этим оператором можно изменить любую часть заголовка триггера (кроме таблицы или представления, с которым связан триггер). Можно изменить тело триггера, точнее, полностью заменить все операторы на новые.

Лично мне такой оператор не нравится. Я предпочитаю создавать *все* объекты базы данных, используя скрипты, которые храню в отдельном каталоге. Если в триггере нужно что-то изменить, я вношу изменения в скрипт, удаляю существующий триггер и заново выполняю измененный скрипт.

9.4.3. Примеры триггеров

Рассмотрим примеры некоторых триггеров. Все тексты вы найдете в скрипте создания триггеров CreateTriggers.sql.

Автоинкрементные поля

Одним из назначений триггеров является автоматическое получение уникального значения числового искусственного первичного ключа из генератора. В литературе такие поля, которым автоматически (с помощью триггеров и генераторов или иными средствами системы) присваиваются уникальные числовые значения, называются автоинкрементными (auto increment).

Здесь мы будем использовать контекстную переменную `NEW`.

Листинг 9.1. Триггер формирования значения первичного ключа для таблицы STAFF

```
/* Формирование значения первичного ключа для таблицы STAFF */
CREATE TRIGGER TBI_STAFF FOR STAFF
  BEFORE INSERT
AS
BEGIN
  IF (NEW.COD IS NULL) THEN
    NEW.COD = GEN_ID(GEN_STAFF, 1);
END ^
```

Этот триггер, представленный в листинге 9.1, будет автоматически вызываться системой перед (`BEFORE`) добавлением (`INSERT`) новой строки в таблицу `STAFF`. Триггер должен вызываться именно *перед* добавлением по той причине, что новая строка должна отправляться в базу данных с правильным, непустым значением первичного ключа.

Контекстная переменная `NEW.COD` ссылается на созданное значение первичного ключа. Если пользователь не задал значения первичного ключа (`NEW.COD`

IS NULL), то ключевому полю обычным образом присваивается значение, полученное из генератора при обращении к функции GEN_ID.

Хочу обратить ваше внимание на важность проверки поля на пустое значение. Если пользователь по какой-либо причине забыл задать значение первичного ключа, триггер исправит подобную оплошность. Однако если пользователь уже присвоил нужное значение и использует его в своей программе (например, для формирования значений внешних ключей в дочерних, подчиненных, таблицах), триггер в этом варианте ничего не испортит.

Замечание

При разработке программ с использованием компонентов FIBPlus или IBX у нас есть более удобные средства для автоматического формирования значения искусственного первичного ключа. Лично я не использую триггеры с подобной целью. В любом случае такой триггер не помешает.

Замечание по формированию имен триггеров

Для имен триггеров здесь мы с вами используем такую систему. Имя начинается с буквы "T" (понятно — от trigger). За ней идет буква "B", если это триггер "до", или "A", если триггер "после". Затем буква "I" (добавление), "U" (изменение) или "D" (удаление). Если на данную фазу этого события базы данных создается несколько триггеров, то сюда добавляется еще и цифра, задающая позицию выполнения триггера. После символа подчеркивания мы записываем имя таблицы, для которой создается триггер.

Триггер по поддержанию значений внешних ключей

Это очень простой триггер, который создается системой, если в описании внешнего ключа вы указали предложение ON UPDATE CASCADE. Его задачей является изменить значения всех соответствующих ключей подчиненной таблицы при изменении первичного ключа в родительской таблице.

Такой триггер не представлен в наших скриптах, он создается системой автоматически. Этот текст мы приводим только с целью иллюстрации.

Если бы мы решили вручную поддерживать связь "главная-подчиненная" для таблиц стран (главная, родительская) и регионов (подчиненная, дочерняя), то мы написали бы следующий триггер, который должен вызываться после изменения кода страны (листинг 9.2).

Кстати, такой триггер будет действительно необходим, если в описании внешнего ключа вы укажете предложение ON UPDATE NO ACTION или вообще не зададите это предложение.

Листинг 9.2. Триггер реализации отношения страна — регион

```
/* Гипотетическое поддержание отношения страна – регион */
CREATE TRIGGER COUNTRY_REGION_UPDATE FOR REFCTR
  AFTER UPDATE
```

```
AS
BEGIN
  IF (NEW.CODCTR <> OLD.CODCTR) THEN
    UPDATE REFREG
      SET REFREG.CODCTR = NEW.CODCTR
      WHERE REFREG.CODCTR = OLD.CODCTR;
END ^
```

При изменении кода страны все регионы, относящиеся к этой стране, получают новое измененное значение кода этой страны.

Реально в нашей базе данных система создала триггер с именем `CHECK_1`, который выполняет эти изменения в таблице регионов.

Удаление строк подчиненной таблицы

Коль скоро мы стали писать триггеры, осуществляющие системные функции, давайте уж напишем и триггер, который будет удалять соответствующие строки из таблицы регионов при удалении строки из таблицы стран. Это такой же триггер, который создала система для реализации каскадного удаления, описанного в предложении `ON DELETE CASCADE` в описании внешнего ключа таблицы регионов. Текст триггера представлен в листинге 9.3.

Такой триггер может реально потребоваться, если в описании внешнего ключа не задать предложение `ON DELETE` или указать `ON DELETE NO ACTION`.

Листинг 9.3. Триггер, выполняющий каскадное удаление строк подчиненной таблицы

```
CREATE TRIGGER COUNTRY_REGION_DELETE FOR REFCTR
  AFTER DELETE
AS
BEGIN
  DELETE FROM REFREG
    WHERE REFREG.CODCTR = OLD.CODCTR;
END ^
```

При удалении страны также удаляются все регионы, относящиеся к этой стране.

В нашей базе данных система создала триггер с именем `CHECK_2`, выполняющий удаление строк в таблице регионов.

Поддержание значения кода района в организациях

Когда в *главе 4* мы проектировали наши таблицы, то столкнулись с досадной ситуацией. В таблице организаций код района, явно являясь внешним ключом (вместе с кодом страны и кодом региона), ссылающимся на таблицу рай-

онов, не мог быть описан как внешний ключ со всеми вытекающими отсюда прелестями автоматического приведения в соответствие всех значений внешнего ключа, если менялось значение первичного ключа в родительской таблице — справочнике районов.

Причина здесь в том, что такой внешний ключ в нашем случае допускал случаи, когда код страны и код региона имеют "нормальные" значения, а код района (если организация находится не в районе, а в областном центре) имеет значение NULL. Это недопустимо для внешнего ключа. Либо все его столбцы имеют значения, которым соответствует строка в родительской таблице, либо все имеют пустое значение.

> ***Замечание***
>
> Есть еще одна более глубокая причина сложившейся ситуации. Справочники стран, регионов, районов имеют чисто иерархическую структуру. Соответственно, и ключи у них имеют точно такую же иерархическую структуру. Отсюда и возникшая у нас проблема с внешним ключом.

На самом деле, ничего сложного для нас с вами здесь нет. С учетом наших глубоких знаний по использованию триггеров, мы легко напишем два триггера, реагирующих на изменение значения первичного ключа в таблице районов и на удаление строки из этой таблицы.

Когда меняется первичный ключ в любой строке таблицы районов и если меняется именно код района, нам нужно во всех записях организаций, где код района равнялся старому, до изменения, значению кода района в таблице районов, присвоить новое, измененное, значение. Фраза получилась достаточно тяжелая, но она точно описывает, что должен выполнить наш триггер, вызываемый после изменения (AFTER UPDATE) записи района. Имя ему мы присвоим, разумеется, TAU_REFAREA. Это будет первая версия нашего триггера. Текст представлен в листинге 9.4.

Листинг 9.4. Триггер корректировки кодов района в записях организаций

```
/* Корректировка кодов района в записях организаций
   после изменения кода района */
CREATE TRIGGER TAU_REFAREA FOR REFAREA
  AFTER UPDATE
AS
BEGIN
  IF (NEW.CODAREA <> OLD.CODAREA) THEN
    UPDATE ORGANIZATION
      SET CODAREA = NEW.CODAREA
    WHERE ((CODAREA = OLD.CODAREA) AND
           (CODCTR = NEW.CODCTR) AND
           (CODREG = NEW.CODREG));
END ^
```

В первую очередь мы проверяем, изменялось ли значение кода района. Наши изменения в записях организаций имеют смысл только в этом случае. Затем в организациях, соответствующих условиям поиска в предложении WHERE, изменяется значение кода региона.

Несколько слов относительно условий поиска. В предложении WHERE не имеет значения, какие, старые или новые, коды страны и коды региона мы используем в условии поиска (контекстные переменные OLD или NEW). Потому как в строке района не могут одновременно изменяться все три поля, входящие в состав внешнего ключа. Код страны и код региона может изменяться автоматически при изменении справочника стран и/или справочника регионов.

Это высказывание довольно спорное. Ведь ничто не мешает нам, используя средства SQL, поменять в строке района значения сразу всех трех ключевых полей. Содержательно, с учетом семантики нашей базы данных, такое изменение означало бы, что мы, используя оператор UPDATE, берем район одной страны и переносим в другую страну. Такое действие попахивает политическим скандалом.

Тем не менее, отвлекаясь от политики, приведем наш триггер к состоянию формальной чистоты и учтем все возможности нашей непростой жизни.

Листинг 9.5. Полная корректировка всех кодов в записях организаций

```
/* Корректировка кодов в записях организаций
   после изменения первичного ключа в записи района */
CREATE TRIGGER TAU_REFAREA FOR REFAREA
  AFTER UPDATE
AS
BEGIN
  IF ((NEW.CODCTR <> OLD.CODCTR) OR
      (NEW.CODREG <> OLD.CODREG) OR
      (NEW.CODAREA <> OLD.CODAREA)) THEN
    UPDATE ORGANIZATION
      SET CODCTR = NEW.CODCTR,
          CODREG = NEW.CODREG,
          CODAREA = NEW.CODAREA
    WHERE ((CODCTR = OLD.CODCTR) AND
           (CODREG = OLD.CODREG) AND
           (CODAREA = OLD.CODAREA));
END ^
```

Здесь триггер, текст которого представлен в листинге 9.5, выполняется при изменении *любого* поля, входящего в состав первичного ключа. Кроме этого,

в соответствующих строках таблицы организаций изменяются *все* столбцы, входящие в состав нашего слегка запутанного внешнего ключа. В условиях поиска предложения WHERE мы должны теперь указывать только старые значения столбцов, используя контекстные переменные OLD.

Теперь мы уже легко напишем триггер установки в NULL кода района в записи организаций после удаления строки района (листинг 9.6). Триггер будет вызываться после удаления (AFTER DELETE).

Листинг 9.6. Установка в NULL кодов района в записях организаций

```
/* Корректировка кодов района в записях организаций
   после удаления района */
CREATE TRIGGER TAD_REFAREA FOR REFAREA
  AFTER DELETE
AS
BEGIN
  UPDATE ORGANIZATION
      SET CODAREA = NULL
    WHERE ((CODCTR = OLD.CODCTR) AND
           (CODREG = OLD.CODREG) AND
           (CODAREA = OLD.CODAREA));
END ^
```

Здесь нам все понятно. Мы присваиваем пустое значение коду района для всех записей, чей внешний ключ равен первичному ключу удаленной записи района.

Созданные нами два триггера (самый первый был только пробным) позволяют реализовать поведение, как если бы у нас был описан внешний ключ:

```
CONSTRAINT FK3_ORGANIZATION
    FOREIGN KEY (CODCTR, CODREG, CODAREA)
      REFERENCES REFAREA (CODCTR, CODREG, CODAREA)
      ON DELETE SET NULL
      ON UPDATE CASCADE,
```

Разница только в том, что у нас более "интеллектуальная" работа с таким внешним ключом.

Создание истории

В нашей базе данных есть таблица STAFFHISTORY, которая хранит историю изменения окладов сотрудников всех организаций. Чтобы строка этой таблицы формировалась автоматически, мы напишем триггер, который будет вызываться после изменения строки сотрудника и станет создавать новую стро-

ку в таблице истории только в том случае, если было изменено значение оклада. Текст триггера см. в листинге 9.7.

Листинг 9.7. Триггер создания истории изменения окладов

```
/* Формирование истории изменения окладов сотрудников */
CREATE TRIGGER TAU_STAFF FOR STAFF
  AFTER UPDATE
AS
BEGIN
  IF (NEW.SALARY <> OLD.SALARY) THEN
    INSERT INTO STAFFHISTORY
      (COD, CODSTAFF, SALARY, DATESALARY) VALUES
      (GEN_ID(GEN_STAFFHISTORY, 1), OLD.COD, OLD.SALARY, CURRENT_DATE);
END ^
```

После изменения пользователем строки сотрудника (оператор `UPDATE`) автоматически вызывается наш триггер. Если величина оклада не изменялась, то триггер ничего не делает.

Если же новое измененное значение оклада отличается от старого (`NEW.SALARY <> OLD.SALARY`), то оператором `INSERT` создается новая строка в таблице истории. Для первичного ключа выбирается новое значение из генератора, из таблицы `STAFF` помещаются значения кода сотрудника и старое значение оклада, формируется дата, до которой действовал старый оклад; ей присваивает значение текущей даты из переменной `CURRENT_DATE`.

Здесь, пожалуй, пора посмотреть на результаты работы триггера и еще раз потренироваться в создании оператора `SELECT`.

Вызовите IBExpert, соединитесь с нашей базой данных Work.fdb (меню **Database | Connect to Database**). Вызовите окно редактирования и выполнения скриптов — меню **Tools | Script Executive**.

Сейчас мы проверим работу триггера, текст которого представлен в листинге 9.7. Этот текст несколько отличается от того, который представлен в скрипте создания триггеров CreateTriggers.sql. В чем отличие, мы чуть позже рассмотрим.

Нам нужно удалить триггер, созданный в скрипте, и создать новый текст. В окне **Script Executive** вызовите и выполните скрипт CreateWrongTrigger.sql. Он содержит операторы соединения с базой данных, удаления ранее созданного триггера и создание нового варианта с тем же именем — листинг 9.8.

Листинг 9.8. Пересоздание триггера

```
SET SQL DIALECT 3;
SET NAMES WIN1251;
CONNECT 'D:\BestDatabase\work.fdb'
USER 'wizard' PASSWORD 'master';

DROP TRIGGER TAU_STAFF;

SET TERM ^;

/* Формирование истории изменения окладов сотрудников (неполное) */
CREATE TRIGGER TAU_STAFF FOR STAFF
  AFTER UPDATE
AS
BEGIN
  IF (NEW.SALARY <> OLD.SALARY) THEN
    INSERT INTO STAFFHISTORY
      (COD, CODSTAFF, SALARY, DATESALARY) VALUES
      (GEN_ID(GEN_STAFFHISTORY, 1), OLD.COD, OLD.SALARY, CURRENT_DATE);
END ^

SET TERM ;^

COMMIT;

EXIT;
```

Проверим работу этого триггера.

Вызовите окно выполнения операторов SQL (меню **Tools | SQL Editor**). В этом окне выполните изменение оклада у сотрудника с кодом, скажем, 1:

```
UPDATE STAFF SET SALARY = 9999
  WHERE COD = 1;
```

Подтвердите транзакцию. Отобразите содержимое таблицы истории окладов:

```
SELECT * FROM STAFFHISTORY;
```

Там появилась одна запись, показанная в листинге 9.9.

Листинг 9.9. История окладов

```
COD     CODSTAFF SALARY    DATESALARY
======= ======== ========= ==========
      1        1  56000.00 14.01.2007
```

Все правильно. Запись содержит коды, старое значение оклада и дату, до которой действовал этот оклад.

Продолжим исследования. Как поведет себя наш триггер, если оклад до изменения имел пустое значение?

Отобразив список всех сотрудников, мы видим, что, например, сотрудник с кодом 16 имеет значение оклада NULL. Установим ему какое-то значение оклада, потому что нехорошо получается, когда человек работает, а денег ему не платят.

```
UPDATE STAFF SET SALARY = 3333
  WHERE COD = 16;
```

Подтвердите транзакцию. Опять отобразите содержимое таблицы истории окладов. Что это? Мы видим опять только одну запись, результат первого изменения оклада. Результат последнего изменения отсутствует.

Дело в том, что в этой версии триггера мы не учли возможность существования пустого значения оклада. Результат проверки на неравенство старого и нового значения оклада не даст нам значения TRUE (хотя они и не равны), поскольку сравнение столбцов, допускающих пустые значения, как мы с вами хорошо знаем (но часто забываем), никогда не даст значения "истина". Наш триггер в подобной ситуации ничего не делает.

Исправим триггер. Не торопитесь. Здесь недостаточно просто добавить в условие дизъюнкцию проверки старого значения оклада на NULL. Чтобы триггер выполнялся при правильных условиях, нужно также проверить отсутствие пустого значения и у нового значения столбца. Правильный текст триггера теперь выглядит, как показано в листинге 9.10.

Листинг 9.10. Правильный триггер создания истории изменения окладов

```
/* Формирование истории изменения окладов сотрудников */
CREATE TRIGGER TAU_STAFF FOR STAFF
  AFTER UPDATE
AS
BEGIN
  IF ((NEW.SALARY <> OLD.SALARY) OR
      ((OLD.SALARY IS NULL) AND (NEW.SALARY IS NOT NULL))) THEN
    INSERT INTO STAFFHISTORY
      (COD, CODSTAFF, SALARY, DATESALARY) VALUES
      (GEN_ID(GEN_STAFFHISTORY, 1), OLD.COD, OLD.SALARY, CURRENT_DATE);
END ^
```

В условие мы добавили дизъюнкцию (логическое ИЛИ) с проверкой условия на наличие пустого значения предыдущего значения оклада и отсутствие при

этом значения NULL у нового значения оклада. В данном случае именно это гарантирует, что оклад был изменен.

Чтобы проверить правильность работы новой версии триггера, вам нужно пересоздать базу данных, сформировать все объекты и заполнить базу данных справочными и переменными данными.

Для этого в программе IBExpert отключаемся от базы данных и последовательно выполняем следующие скрипты в указанном порядке:

17. RecreateDatabase.sql
18. CreateDomains.sql
19. CreateTables.sql
20. CreateTriggers.sql
21. CreateViews.sql
22. InsertReference.sql
23. InsertOperative.sql
24. UpdateOperative.sql

Соединившись в IBExpert с базой данных, опять последовательно делаем изменения окладов тех же сотрудников, выполняя отдельно каждый из следующих операторов и подтверждая каждый раз транзакцию после выполнения оператора. Сначала оператор:

```
UPDATE STAFF SET SALARY = 9999
  WHERE COD = 1;
```

Затем:

```
UPDATE STAFF SET SALARY = 3333
  WHERE COD = 16;
```

Отобразим содержимое таблицы истории окладов:

```
SELECT * FROM STAFFHISTORY;
```

Мы получаем тот результат, который нам был нужен — листинг 9.11.

Листинг 9.11. История окладов

```
COD     CODSTAFF SALARY    DATESALARY
======= ======== ========= ==========
      1        1  56000.00 14.01.2007
      2       16 NULL      14.01.2007
```

Теперь мы правильно учитываем и пустые оклады.

Совет-напоминание

Давайте остановимся и еще раз себе напомним про важный момент при работе с реляционными базами данных. Если столбец, участвующий в операциях сравнения, может принимать пустое значение, то в соответствующее условие мы должны добавлять и проверку на значение NULL. Причем не всегда такое добавление может быть простым делом. Скажу по секрету, что первую версию триггера я таки написал без подобной проверки.

Замечание

Если у вас сложилось впечатление, что последняя версия нашего триггера абсолютно безупречна, то вы ошибаетесь. На самом деле в условии мы не учли еще и такую довольно странную с человеческой точки зрения ситуацию, когда у сотрудника был задан нормальный оклад, а в результате изменения ему присваивается пустое значение. Поскольку операторы SQL позволяют нам выполнить и такое не очень осмысленное действие, необходимо несколько изменить условие в триггере. Это довольно просто. Предлагаю вам выполнить соответствующее изменение самим.

Однако чего-то в полученном списке все-таки не хватает — красоты. Результат отображения не соответствует нашим эстетическим потребностям.

Изменим оператор SELECT следующим образом:

```
SELECT (SELECT FULLNAME
          FROM PEOPLE
          WHERE COD = (SELECT CODPEOPLE
                         FROM STAFF
                         WHERE COD = H.CODSTAFF)) AS "Фамилия",
       H.SALARY AS "Оклад",
       H.DATESALARY AS "Дата"
  FROM STAFFHISTORY H;
```

Помимо добавления заголовков мы отыскиваем фамилию, имя и отчество сотрудника. Это делается путем использования двух вложенных операторов SELECT на месте первого выбираемого столбца в главном операторе SELECT.

Самый внутренний SELECT в таблице персонала STAFF находит код соответствующего человека. Оператор SELECT более высокого уровня на основании этого кода отыскивает соответствующую строку в таблице людей PEOPLE и выбирает оттуда полное имя человека.

Напомню, что при подобном использовании вложенные операторы SELECT обязательно должны заключаться в скобки.

Результат выборки будет таким, как показано в листинге 9.12.

Листинг 9.12. История окладов в улучшенном варианте

```
Фамилия                   Оклад     Дата
========================= ========= =========
ЛЕВИН МИХАИЛ МИХАЙЛОВИЧ    56000.00 14.01.2007
NULL                      NULL      14.01.2007
```

Замечания по поводу триггера создания истории

Теперь, когда мы проделали такую тяжелую работу по формированию истории окладов, совершили обидные ошибки, наткнулись на проблемы с пустыми значениями, скажем несколько правдивых слов по поводу такого триггера.

Подобный пример (без проверки на пустое значение) существует в документации по InterBase. Мне же такой подход к автоматическому формированию строки истории кажется порочным.

Дело в том, что изменения оклада в базе данных в реальной жизни могут выполняться в двух случаях. Во-первых, когда человеку действительно изменяют величину оклада. Тогда, верно, имеет смысл создать запись истории, если этого хочет наш заказчик. Однако существует и вариант, когда при помещении записи в базу данных было введено ошибочное значение оклада. В этом случае, при исправлении допущенной ошибки создание строки истории совершенно бессмысленно.

Подобные изменения я выполняю в клиентской программе, где можно реализовать более интеллектуальное поведение и дать возможность пользователю решить, следует ли создавать запись истории. Очень часто бизнес-логику имеет смысл переносить из базы данных (из триггеров, хранимых процедур) в клиентское приложение. Об этом я еще скажу несколько слов чуть позже. Кроме того, структура записи таблицы истории, конечно же, должна быть несколько иной.

Давайте рассматривать этот триггер как хороший пример для тренировки в написании кодов на языке хранимых процедур и триггеров.

Помещение в базу данных нового человека

В *главе 5* мы кратко рассмотрели фрагменты триггеров, которые использовались при добавлении нового человека для проверки правильности задания кода супруга. Сейчас более подробно рассмотрим эти триггеры.

Прежде чем поместить новую запись человека в базу данных, если пользователь указывает код супруга, имеет смысл проверить, существует ли этот супруг в базе данных и имеет ли он противоположный признак пола. Эти про-

верки выполняются в триггере TBI_PEOPLE (листинг 9.13), который автоматически вызывается до добавления новой записи.

Листинг 9.13. Проверка характеристик супруга до добавления записи в базу

```
CREATE TRIGGER TBI_PEOPLE FOR PEOPLE
  BEFORE INSERT
AS
DECLARE VARIABLE CODI INTEGER;
DECLARE VARIABLE SEXI CHAR(1);
BEGIN
  IF (NEW.CODOTHERHALF IS NOT NULL) THEN
  BEGIN
    IF (NOT EXISTS(SELECT COD
                   FROM PEOPLE
                   WHERE COD = NEW.CODOTHERHALF)) THEN
       EXCEPTION NO_PEOPLE;
    SELECT SEX, CODOTHERHALF FROM PEOPLE WHERE COD = NEW.CODOTHERHALF
      INTO :SEXI, :CODI;
    IF (NEW.SEX = SEXI) THEN
      EXCEPTION SEX_PEOPLE_INSERT;
    IF (CODI IS NOT NULL) THEN
      EXCEPTION OTHER_PEOPLE;
  END
END ^
```

Триггер выполняется только в случае непустого значения кода супруга. Проверяется, присутствует ли в базе данных соответствующая строка. Для такой проверки используется удобная функция EXISTS.

Альтернативным вариантом может быть, например, подсчет количества строк, имеющих соответствующее значение первичного ключа. Понятно, что таких строк может быть или одна, или ни одной. В тексте триггера такой вариант помещен в комментарий (в другом триггере я использую именно этот не очень умный вариант, только для примера):

```
SELECT COUNT(COD) FROM PEOPLE WHERE COD = NEW.CODOTHERHALF
    INTO :CODI;
  IF (CODI = 0) THEN ...
```

Если записи нет, выдается исключение NO_PEOPLE с текстом "Запись человека отсутствует в базе данных" и работа прекращается. Напомню, все исключения, используемые в наших триггерах, описаны в *разд. 9.2*.

Далее выбирается запись супруга. Если супруг имеет тот же пол, то выдается исключение SEX_PEOPLE_INSERT с текстом "Неверный пол супруга при добавлении записи".

Последняя проверка — является ли этот человек уже мужем (женой) другого человека. Если да, то выдается исключение OTHER_PEOPLE "У супруга установлена связь с другим человеком".

Следующий триггер TAI_PEOPLE (листинг 9.14) вызывается сразу после добавления. Он выполняет корректировку кода супруга во второй записи, записи супруга. Скорректировать этот код в первом триггере нельзя, потому что первая запись (родительская для второй записи) еще не помещена в базу данных, и такая корректировка вызовет исключение неверного значения внешнего ключа.

Листинг 9.14. Корректировка кода супруга во второй строке после добавления записи

```
CREATE TRIGGER TAI_PEOPLE FOR PEOPLE
  AFTER INSERT
AS
BEGIN
  IF (NEW.CODOTHERHALF IS NOT NULL) THEN
    UPDATE PEOPLE SET CODOTHERHALF = NEW.COD
      WHERE COD = NEW.CODOTHERHALF;
END ^
```

Для корректной работы триггера здесь также проверяется, был ли задан в добавляемой записи код супруга.

С помощью этих двух триггеров мы перекрыли возможность помещения в базу данных неверных сведений о супругах. Это работает только для вновь помещаемых записей. Теперь рассмотрим возможности триггеров при изменении кодов супругов в существующих записях.

Корректировка данных человека

В этом случае нам также понадобятся два триггера — один до изменения (BEFORE UPDATE), другой после изменения (AFTER UPDATE). Они похожи на уже рассмотренные триггеры, однако условия их выполнения пришлось несколько усложнить.

Первый триггер вызывается *до* выполнения изменений — листинг 9.15.

Листинг 9.15. Проверка характеристик супруга до изменения записи человека

```
CREATE TRIGGER TBU_PEOPLE FOR PEOPLE
```

```
  BEFORE UPDATE
AS
DECLARE VARIABLE CODI INTEGER;
DECLARE VARIABLE SEXI CHAR(1);
BEGIN
  IF ((NEW.CODOTHERHALF IS NOT NULL) AND
      ((OLD.CODOTHERHALF IS NULL) OR
      (NEW.CODOTHERHALF <> OLD.CODOTHERHALF))) THEN
  BEGIN
    SELECT COUNT(COD) FROM PEOPLE WHERE COD = NEW.CODOTHERHALF
      INTO :CODI;
    IF (CODI = 0) THEN
      EXCEPTION NO_PEOPLE;
    SELECT SEX, CODOTHERHALF FROM PEOPLE WHERE COD = NEW.CODOTHERHALF
      INTO :SEXI, :CODI;
    IF (NEW.SEX = SEXI) THEN
      EXCEPTION SEX_PEOPLE_UPDATE;
    IF ((CODI IS NOT NULL) AND (CODI <> NEW.COD)) THEN
      EXCEPTION OTHER_PEOPLE;
  END
END ^
```

Посмотрим на условие выполнения триггера:

```
IF ((NEW.CODOTHERHALF IS NOT NULL) AND
    ((OLD.CODOTHERHALF IS NULL) OR
    (NEW.CODOTHERHALF <> OLD.CODOTHERHALF))) THEN ...
```

Здесь помимо того, что мы проверяем на `NULL` новое значение кода супруга (что само по себе тоже не очень правильно; об этом см. дальше), мы также проверяем, изменился ли этот код. В процессе проверки мы не забыли учесть, что старое значение кода может быть и пустым.

Дальше выполняются действия, похожие на те, что мы делали в триггерах, вызываемых при добавлении новых данных с учетом небольших различий в вызываемых исключениях и формируемых ими текстах сообщений об ошибках. Только здесь я решил продемонстрировать принципиальную допустимость проверки существования нужной строки в более глупом коде, о котором говорил ранее:

```
SELECT COUNT(COD) FROM PEOPLE WHERE COD = NEW.CODOTHERHALF
    INTO :CODI;
  IF (CODI = 0) THEN ...
```

Второй триггер, представленный в листинге 9.16, вызывается *после* выполнения изменения записи человека:

Листинг 9.16. Корректировка кода супруга во второй строке после изменения записи

```
CREATE TRIGGER TAU_PEOPLE FOR PEOPLE
  AFTER UPDATE
AS
BEGIN
  IF ((NEW.CODOTHERHALF IS NOT NULL) AND
      ((OLD.CODOTHERHALF IS NULL) OR
      (NEW.CODOTHERHALF <> OLD.CODOTHERHALF))) THEN
  BEGIN
    UPDATE PEOPLE SET CODOTHERHALF = OLD.COD
      WHERE COD = NEW.CODOTHERHALF;
  END
END ^
```

Условие выполнения триггера точно такое же, как и в первом триггере. При его выполнении осуществляется корректировка кода супруга во второй записи.

Замечания относительно триггеров, связанных с созданием и изменением записей по людям

Наши триггеры для изменения записи человека не являются полнофункциональными. Если пользователь убирает сведения о супруге (устанавливает значение его кода в `NULL`), то наши триггеры не смогут на это адекватно отреагировать. В этом случае следовало бы у второго супруга, если он присутствует в базе данных, также установить в `NULL` код супруга. Мне, честно скажу, лень, а точнее, не хватает времени дальше думать на эту тему, потому что хочу быстрее предоставить вам эту книгу. Наверняка у вас появятся светлые мысли по этому поводу. Сообщите мне результаты ваших раздумий.

Основная загвоздка здесь в том, что при решении полной задачи можно получить бесконечный цикл обновлений, когда триггеры будут постоянно вызываться при корректировке кодов супруга — сначала в первой записи, которую скорректировал пользователь, затем во второй записи. Корректировка второй записи автоматически опять вызовет триггер, который начнет корректировать первую запись (уже не нужно), эта корректировка автоматически вызовет триггер для изменения второй записи и т. д. до бесконечности.

Здесь так же, как и в случае с триггером создания истории (см. *разд. "Замечания по поводу триггера создания истории"* несколькими страницами ранее), я предпочитаю переносить эту бизнес-логику в клиентское приложение. В программе проще и удобнее предложить пользователю выполнить корректировку, а затем программа сделает все необходимые изменения в базе данных.

Используемые нами возможности триггеров предполагают, скорее всего, что написание программ будет выполнять не высокопрофессиональный программист, а какой-нибудь кодировщик. Однако чаще всего мы с вами и создаем базу данных, и пишем программы, использующие эту базу данных. По этой причине мы можем рассчитывать на то, что созданную нами базу данных будет использовать человек с сильно развитым интеллектом и высоким профессиональным уровнем.

Отправка событий базы данных

Последний вариант триггера, который мы с вами сейчас рассмотрим, связан с событиями базы данных. Термин "событие" (event) не очень удачен. Здесь речь идет только о передаче клиентским программам, работающим в настоящий момент с базой данных, некоторого сообщения, сигнала.

Посмотрите еще раз *разд. 9.3*.

Сейчас мы с вами напишем простенький триггер (листинг 9.17), который будет посылать сообщения клиентским программам о том, что произошло изменение данных в конкретной таблице, в данном случае — в справочнике стран.

Листинг 9.17. Сообщение о событии добавления строки справочника стран

```
CREATE TRIGGER TAI_REFCTR FOR REFCTR
  AFTER INSERT
AS
BEGIN
  POST_EVENT 'NEW_REFCTR';
END ^
```

Проще трудно придумать. Здесь отправляется сообщение о событии NEW_REFCTR, которое получит то клиентское приложение, которое "прослушивает" именно это событие, оно перехватит это событие и выполнит соответствующие действия.

Для полноты ощущений напишем еще и триггер, который сообщает всем, что данные в справочнике стран изменились. Этот триггер также не отличается большой сложностью — листинг 9.18.

Листинг 9.18. Сообщение о событии изменения строки справочника стран

```
CREATE TRIGGER TAU_REFCTR FOR REFCTR
  AFTER UPDATE
AS
BEGIN
```

```
  POST_EVENT 'UPDATE_REFCTR';
END ^
```

Чтобы быть совсем логичными, напишем и триггер, сообщающий об удалении данных из справочника стран (листинг 9.19).

Листинг 9.19. Сообщение о событии удаления строки справочника стран

```
CREATE TRIGGER TAD_REFCTR FOR REFCTR
  AFTER DELETE
AS
BEGIN
  POST_EVENT 'DELETE_REFCTR';
END ^
```

Реакцией нашего клиентского приложения на каждое из этих событий будет переоткрытие справочника стран.

Замечание по поводу триггеров отправки событий

Если вам действительно нужно использовать средства отправки событий только для переоткрытия используемого в программе набора данных в случае любого изменения записей другими клиентами, то во всех трех триггерах имеет смысл выдавать (посылать) ровно одно событие, которое можно назвать, например, `REFCTR_CHANGED`.

Если вы используете Firebird версии не ниже 1.5 (но не InterBase), вы для этих целей можете использовать один триггер. В этой версии можно написать один триггер, который выполняется и после добавления, и после изменения, и после удаления строки. Это может быть такой триггер — листинг 9.20.

Листинг 9.20. Сообщение о группе событий строки справочника стран

```
CREATE TRIGGER TAC_REFCTR FOR REFCTR
  AFTER INSERT OR UPDATE OR DELETE
AS
BEGIN
  POST_EVENT 'REFCTR_CHANGED';
END ^
```

В имени триггера присутствует буква "C", т. е. change, изменение.

Если у вас установлена СУБД InterBase любой версии или Firebird версии 1.0, не создавайте этот триггер. В скрипте CreateTriggers.sql следует удалить описание данного триггера. Он самый последний в тексте. Впрочем, если вы и не удалите этот триггер, то ничего страшного не произойдет. Вы просто получите сообщение об ошибке. Остальные же триггеры будут уже созданы.

9.5. Хранимые процедуры

Триггеры являются действительно полезным средством реляционных баз данных. При этом хранимые процедуры несколько более сложны и, пожалуй, чаще используются в реальной программистской практике.

Хранимые процедуры могут вызываться из клиентских приложений, из триггеров и из хранимых же процедур. Выполняются хранимые процедуры на стороне сервера в контексте той транзакции, в которой выполняется вызвавший их программный компонент.

Для написания хранимых процедур так же, как и для написания триггеров, используется расширение языка SQL. Это расширение, как вы помните, называется языком хранимых процедур и триггеров. В отличие от триггеров здесь нельзя по понятным причинам использовать контекстные переменные `NEW` и `OLD`. Помимо остальных операторов языка в хранимых процедурах мы можем использовать и специфический оператор `SUSPEND`. Хранимые процедуры в отличие от триггеров могут получать входные параметры и могут возвращать выходные значения.

Хранимые процедуры бывают двух видов, типов. Это выполняемые хранимые процедуры (executed procedures) и хранимые процедуры выбора (selected procedures).

Выполняемые хранимые процедуры могут получать входные параметры и выполняют какие-то действия с данными базы данных или с любыми другими данными. Вызов такой процедуры из триггера или хранимой процедуры осуществляется оператором `EXECUTE PROCEDURE` или любым другим соответствующим средством клиентского приложения. Выполняемые хранимые процедуры могут возвращать выходные значения вызвавшему программному компоненту (клиентскому приложению, триггеру, хранимой процедуре).

Хранимые процедуры выбора также могут получать входные параметры. Основное назначение таких процедур — чтение строк из таблицы (группы таблиц) или из представления. В основе таких процедур чаще всего лежит оператор `SELECT`, возвращающий множество строк. После выборки и возможной предварительной обработки очередной строки процедура выдает оператор `SUSPEND`, который временно приостанавливает выполнение хранимой процедуры и возвращает эту строку вызвавшей программе. Затем процесс повторяется, пока не будут считаны все записи, соответствующие условиям поиска. Хранимая процедура выбора обязательно возвращает выходные значения вызвавшей программе. Обращение к такой хранимой процедуре осуществляется при использовании оператора `SELECT`, в предложении `FROM` которого указывается не таблица или представление, а имя хранимой процедуры выбора и, при необходимости, список входных параметров. Такие процедуры явля-

ются мощным средством получения данных из таблиц, которые могут быть подвергнуты "интеллектуальной" обработке.

При создании хранимых процедур никак синтаксически не указывается, является ли процедура процедурой выбора или выполняемой процедурой. Тип процедуры определяется в ее теле.

Примеры хранимых процедур мы с вами рассмотрим в этой главе.

9.5.1. Создание, изменение и удаление хранимых процедур

Хранимая процедура создается оператором CREATE PROCEDURE:

```
CREATE PROCEDURE <имя хранимой процедуры>
  [(<входной параметр> <тип данных>
  [, <входной параметр> <тип данных>] ...)]
[RETURNS [(<выходной параметр> <тип данных>
         [, <выходной параметр> <тип данных>] ...)]
AS
<тело процедуры>;

<тело процедуры> ::=
  [<список локальных переменных>]
  <блок операторов>

<список локальных переменных> ::=
  DECLARE VARIABLE <имя переменной> <тип данных>;
  [DECLARE VARIABLE <имя переменной> <тип данных>;] ...
```

Поскольку мы с вами внимательно изучили синтаксис оператора создания триггера, то и основные синтаксические конструкции в операторе создания процедуры нам понятны. По этой причине не станем уточнять такие термины, как "тип данных", "блок операторов".

Для удаления хранимой процедуры используется оператор DROP PROCEDURE:

```
DROP PROCEDURE <имя хранимой процедуры>;
```

Оператор изменения существующей хранимой процедуры похож на оператор создания процедуры:

```
ALTER PROCEDURE <имя хранимой процедуры>
  [(<входной параметр> <тип данных>
  [, <входной параметр> <тип данных>] ...)]
[RETURNS [(<выходной параметр> <тип данных>
         [, <выходной параметр> <тип данных>] ...)]
[AS <тело процедуры>];
```

С помощью этого оператора можно изменить входные параметры, выходные параметры и/или тело существующей хранимой процедуры.

Опять сообщу свое мнение на изменение хранимых процедур. Я предпочитаю изменить текст хранимой процедуры в скрипте, удалить существующую процедуру и заново создать ее с уже измененным текстом. Правда, это не всегда бывает просто сделать. Вы не сможете удалить процедуру, на которую есть ссылки из других процедур или триггеров. Порядок действий нужно выбирать в каждом отдельном случае.

Замечание по именованию хранимых процедур

У меня нет никаких разумных правил именования хранимых процедур. Обычно имя хранимой процедуры я начинаю с символов PROC_, за которыми идут произвольные тексты, более или менее объясняющие (с моей точки зрения) назначение этой процедуры. В примере из документации по InterBase, который я привожу чуть позже, я сохраняю оригинальную систему имен.

Для помещения в нашу базу данных всех рассматриваемых здесь хранимых процедур (как выполняемых процедур, так и процедур выбора) нужно в IBExpert выполнить скрипт CreateProcedures.sql. Там содержатся тексты и некоторых других процедур, которые мы рассмотрим несколько позже, в следующей главе.

9.5.2. Выполняемые хранимые процедуры

Выполняемые хранимые процедуры обычно выполняют какие-то действия с данными базы данных. Они могут получать входные параметры и могут возвращать выходные параметры — результаты своей работы. Рассмотрим несколько простых примеров выполняемых хранимых процедур.

Получение значения первичного ключа

Чуть раньше мы написали триггер, который получает из генератора уникальное числовое значение и присваивает его первичному ключу новой строки, помещаемой в таблицу сотрудников STAFF. Основным недостатком такого способа формирования значения искусственного первичного ключа является то, что это значение неизвестно клиентской программе.

Другой способ получения значения такого ключа состоит в написании хранимой процедуры, которая вернет нужное значение вызвавшей программе. Напишем такую процедуру (листинг 9.21).

Листинг 9.21. Хранимая процедура для формирования значения первичного ключа

```
/* Формирование значение первичного ключа для таблицы STAFF */
CREATE PROCEDURE PROC_STAFF
  RETURNS (COD INTEGER)
```

```
AS
BEGIN
  COD = GEN_ID(GEN_STAFF, 1);
END ^
```

Ничего нового для нас с вами здесь нет. В процедуре используется хорошо знакомая нам функция `GEN_ID` для получения из генератора нового значения. Это значение присваивается выходному параметру `COD`.

Для обращения к этой процедуре из другой процедуры или триггера мы можем использовать, например, такой оператор:

```
EXECUTE PROCEDURE PROC_STAFF RETURNING_VALUES (OUR_COD);
```

При написании программ работы с базами данных в современных интегрированных средах разработки, таких как Delphi или C++Builder, для получения значений искусственных первичных ключей нам не понадобятся ни триггеры, ни хранимые процедуры. Такие значения можно получить автоматически при использовании компонентов FIBPlus или IBX. Пример подобной программы мы напишем в следующей главе.

Вычисление факториала целого числа

Напишем еще одну простую хранимую процедуру, вычисляющую факториал числа. Алгоритм такого расчета мы рассматривали ранее в этой главе. Процедура может выглядеть следующим образом — листинг 9.22.

Листинг 9.22. Хранимая процедура для вычисления факториала целого числа

```
CREATE PROCEDURE PROC_FACTORIAL (N INTEGER)
  RETURNS (RESULT INTEGER)
AS
DECLARE VARIABLE I INTEGER;
BEGIN
  RESULT = 1;
  I = 1;
  WHILE (I <= N) DO
  BEGIN
    RESULT = RESULT * I;
    I = I + 1;
  END
  SUSPEND;
END ^
```

Исходное значение передается в виде целого числа во входном параметре `N`. Факториал возвращается в выходном параметре `RESULT`.

Кстати, все входные параметры хранимым процедурам передаются по значению, т. е. процедуре передается временная копия параметра. Процедура может как угодно изменять значение входного параметра. Это никак не скажется на значении параметра в вызвавшей программе.

Несмотря на то, что в этом подразделе мы рассматриваем только выполняемые хранимые процедуры, в данном тексте мы видим оператор SUSPEND. Его присутствие допустимо и в выполняемых процедурах. В них он осуществляет функцию оператора EXIT. Я его поместил перед последним оператором END только для того, чтобы выполнить проверку работы процедуры в IBExpert с использованием оператора SELECT.

Для обращения к процедуре выполним оператор SELECT:

```
SELECT RESULT
FROM PROC_FACTORIAL(5);
```

Результатом будет число 120. Вместо имени параметра в операторе можно указать и символ *. Результаты будут такими же.

Можете поэкспериментировать с другими значениями.

Рекурсивное вычисление факториала целого числа

Приведу в листинге 9.23 почти без комментариев текст рекурсивной процедуры вычисления факториала числа. Пример взят из документации по InterBase.

Листинг 9.23. Рекурсивная процедура вычисления факториала целого числа

```
CREATE PROCEDURE FACTORIAL (NUM INTEGER)
   RETURNS (N_FACTORIAL DOUBLE PRECISION)
AS
DECLARE VARIABLE NUM_LESS_ONE INT;
BEGIN
   IF (NUM = 1) THEN
   BEGIN /**** Простейший случай: 1! = 1 ****/
      N_FACTORIAL = 1;
      SUSPEND;
   END
   ELSE
   BEGIN /**** Рекурсия: NUM! = (NUM * (NUM-1))! ****/
      NUM_LESS_ONE = NUM – 1;
      EXECUTE PROCEDURE FACTORIAL NUM_LESS_ONE
      RETURNING_VALUES N_FACTORIAL;
      N_FACTORIAL = N_FACTORIAL * NUM;
```

```
      SUSPEND;
    END
END ^
```

Хранимая процедура может обращаться сама к себе. Максимальный уровень рекурсивных обращений не может превышать 1000, чтобы предотвратить зацикливание. В некоторых случаях при отсутствии необходимых ресурсов это количество может быть гораздо меньшим.

Разберитесь с текстом самостоятельно. Обратите внимание, здесь входной параметр `NUM` свободно используется как локальная переменная. Это никак не влияет на его значение в вызвавшей программе.

Проверьте работу процедуры, выполнив оператор:

```
SELECT *
FROM FACTORIAL(5);
```

Результатом будет 120.000. Форма результата отличается от предыдущего по той причине, что здесь выходной параметр описан с типом `DOUBLE PRECISION`, в то время как в предыдущей нашей процедуре выходной параметр имел тип данных `INTEGER`.

Сравните текст этой процедуры с первой версией (см. листинг 9.22). Результат один и тот же, однако насколько проще и понятнее (может быть только с моей точки зрения) наша прежняя версия процедуры. Вспоминается фраза: "Будь проще, и к тебе потянутся люди". Более серьезным является принцип Оккама (называемый также "бритвой Оккама"), который формулируется так: "Не следует посредством большего делать то, что можно сделать посредством меньшего". Кстати, это замечательный принцип, который можно повсеместно использовать в программировании.

Последний вариант хранимой процедуры следует рассматривать только как пример рекурсивной процедуры. Не надо думать, что это лучшее решение задачи.

На этом мы заканчиваем рассматривать примеры выполняемых хранимых процедур. При необходимости вы напишите любую необходимую вам процедуру такого типа.

9.5.3. Хранимые процедуры выбора

Основное назначение хранимых процедур выбора — осуществление сложной выборки данных из таблицы (таблиц). Процедуры выбора, являясь императивными объектами, дают возможность осуществить такую выборку данных, которая не по силам декларативным средствам языка SQL.

Основу процедуры выбора составляет оператор или операторы SELECT. Оператор возвращает произвольное количество строк, однако в процедуре должен присутствовать оператор SUSPEND, дойдя до которого процедура временно приостанавливает свою работу и возвращает управление вызвавшей программе вместе с очередной найденной строкой данных.

Обращение из программ (из клиентских приложений, триггеров или хранимых процедур) к процедурам выбора осуществляется оператором SELECT, где в предложении FROM вместо имени таблицы указывается имя процедуры. Если процедура принимает входные параметры, то нужное количество параметров с соответствующими типами данных должно перечисляться после имени процедуры в скобках.

Чтобы разобраться с этими процедурами и приобрести опыт написания полезных процедур выбора, рассмотрим ряд примеров.

Простой вариант процедуры выбора регионов всех стран

Начнем с простейшей процедуры выбора. Эта процедура будет выбирать все регионы всех стран из таблицы REFREG. Назовем ее PROC_SELECT_REGION1 (в дальнейшем будут и 2, и 3). Текст процедуры представлен в листинге 9.24.

Листинг 9.24. Простой вариант процедуры выбора регионов всех стран

```
CREATE PROCEDURE PROC_SELECT_REGION1
  RETURNS (CODREG CHAR(2), NAMEREG CHAR(40), CENTER CHAR(25))
AS
BEGIN
  FOR SELECT CODREG,
             NAMEREG,
             CENTER
      FROM REFREG
      INTO :CODREG, :NAMEREG, :CENTER
  DO
    SUSPEND;
END ^
```

Здесь используется цикл FOR-SELECT-DO для выбора данных из таблицы регионов. Указанные в операторе столбцы очередной строки таблицы помещаются в выходные параметры. Это задается в предложении INTO. Дойдя до оператора SUSPEND, процедура приостанавливает свою работу и возвращает выходные параметры (очередную строку) вызвавшей программе. Программа (с помощью используемых компонентов обращения к базе данных) помещает строку во внутренний набор данных и выдает через функцию API запрос на

получение следующей строки из таблицы. После этого работа процедуры продолжается с оператора SELECT, который выбирает очередную строку таблицы, пока не будут найдены все строки, соответствующие условию поиска, определенному в предложении WHERE — в этом операторе отсутствует.

Обратите внимание, что в операторе SELECT в предложении INTO именам выходных параметров предшествует символ двоеточия. Напомню, что в операторах SELECT, INSERT, UPDATE и DELETE именам параметров и локальных переменных обязательно должно предшествовать двоеточие. Это делается для того, чтобы не спутать их с именами столбцов таблицы.

Возвращаемые параметры NAMEREG и CENTER имеют тип данных, несколько отличающийся от типов данных, указанных для столбцов таблицы регионов — у нас CHAR, в таблицах VARCHAR. Это вполне допустимо.

Выполните обращение к этой процедуре из окна SQL Editor программы IBExpert:

```
SELECT *
  FROM PROC_SELECT_REGION1;
```

Вы получите 130 записей всех регионов всех стран, присутствующих в нашей базе данных.

В операторе можно указать и имена выходных параметров, например:

```
SELECT NAMEREG, CENTER
  FROM PROC_SELECT_REGION1;
```

Фактически это тот же оператор, что и

```
SELECT NAMEREG, CENTER FROM REFREG;
```

При обращении к процедуре выбора можно задать и упорядочение полученного набора данных как обычно, в предложении ORDER BY:

```
SELECT NAMEREG, CENTER
  FROM PROC_SELECT_REGION1
  ORDER BY CENTER;
```

В результате список будет упорядочен по названиям центра региона.

Еще лучше. Мы можем отобрать нужные нам строки, полученные из процедуры выбора, используя предложение WHERE. Например:

```
SELECT NAMEREG, CENTER
  FROM PROC_SELECT_REGION1
  WHERE NAMEREG CONTAINING 's'
  ORDER BY CENTER;
```

Результатом будет упорядоченный список из 23 регионов. Насколько внимательно я посмотрел на список, не знаю, но, по-моему, там три графства Великобритании и 20 штатов США.

Можете поэкспериментировать дальше. Когда закончите, начнем улучшать нашу процедуру.

Более полный вариант процедуры выбора регионов страны

Похоже, нам понравилась наша процедура, а главное — возможности SQL по использованию процедур выбора. Однако вызывает некоторое сомнение целесообразность применения такой процедуры. Она выбирает регионы всех стран, которые соответствуют условиям поиска в вызывающем процедуру операторе `SELECT`. Вряд ли такой вариант процедуры найдет широкое практическое применение.

Изменим процедуру, добавив входной параметр, в котором при вызове процедуры будет задаваться код страны, регионы которой нас интересуют. Новой версии процедуры присвоим имя `PROC_SELECT_REGION2` (листинг 9.25).

Листинг 9.25. Более полный вариант процедуры выбора регионов одной страны

```
CREATE PROCEDURE PROC_SELECT_REGION2 (CODCTR CHAR(3))
  RETURNS (CODREG CHAR(2), NAMEREG CHAR(40), CENTER CHAR(25))
AS
BEGIN
  FOR SELECT CODREG,
             NAMEREG,
             CENTER
      FROM REFREG
      WHERE CODCTR = :CODCTR
      INTO :CODREG, :NAMEREG, :CENTER
  DO
    SUSPEND;
END ^
```

В заголовок процедуры мы поместили входной параметр `CORCTR`, который имеет тип данных `CHAR(3)`, как и соответствующий столбец в таблице регионов.

В оператор `FOR-SELECT-DO` мы добавили предложение `WHERE`:

```
WHERE CODCTR = :CODCTR
```

Обратите опять внимание — имя входного параметра начинается с двоеточия, чтобы отличить его от имени столбца таблицы.

Проверим работу процедуры. Введем и выполним оператор:

```
SELECT *
  FROM PROC_SELECT_REGION2('558');
```

Мы получим список регионов России. Здесь мы также можем выполнять нужное упорядочение результата, задавать дополнительные условия поиска записей.

Мне еще раз хочется показать вам всю мощь оператора `SELECT`. Введите и выполните оператор:

```
SELECT *
  FROM PROC_SELECT_REGION2((SELECT CODCTR
                            FROM REFCTR
                            WHERE UPPER(NAME) = UPPER('Россия')));
```

Здесь вместо кода страны мы используем вложенный оператор `SELECT`, который возвращает нам код страны Россия. Обратите внимание, сколько уровней скобок у нас присутствует после имени процедуры. Самый внешний уровень скобок заключает в себе входной параметр. Просто здесь написать оператор `SELECT` мы не можем, мы обязаны также заключить его в скобки.

> ***Замечание***
>
> Выполнение этого оператора в InterBase 2007 дало мне нулевое количество записей. Дело в том, что в этом сервере функция `UPPER` в применении к литералу, содержащему буквы кириллицы, не выполняет никаких преобразований. Латинские же буквы преобразуются правильно. При этом функция `UPPER`, примененная к столбцу таблицы, содержащему буквы кириллицы, выполняется верно.

Еще один нюанс в предложении `WHERE`. Здесь и к имени столбца таблицы, и к литералу я применил функцию `UPPER`, возвращающую текст в верхнем регистре. Для столбца это точно имеет смысл, если мы не знаем, в каком регистре представлены там тексты. Обращаю ваше внимание. Это работает. Особенно хорошо работает, когда тексты в столбцах записываются и строчными, и прописными буквами.

Если хотите, выполните различные варианты обращения к этой процедуре. Попробуйте параметром процедуры задать `NULL`. Результатом будет пустой список.

Возникает идея изменить процедуру так, чтобы при задании для входного параметра пустого значения процедура возвращала бы полный список всех регионов, хранящихся в нашей базе данных.

Безупречный вариант процедуры выбора регионов страны

Создадим процедуру с именем `PROC_SELECT_REGION3`, которая объединит две предыдущие процедуры. Текст процедуры представлен в листинге 9.26.

Листинг 9.26. Безупречный вариант процедуры выбора регионов страны

```
CREATE PROCEDURE PROC_SELECT_REGION3 (CODCTR CHAR(3))
  RETURNS (CODREG CHAR(2), NAMEREG CHAR(40), CENTER CHAR(25))
AS
BEGIN
  IF (CODCTR IS NULL) THEN
  BEGIN
  FOR SELECT CODREG,
             NAMEREG,
             CENTER
      FROM REFREG
      INTO :CODREG, :NAMEREG, :CENTER
  DO
    SUSPEND;
  END
  ELSE
  BEGIN
  FOR SELECT CODREG,
             NAMEREG,
             CENTER
      FROM REFREG
      WHERE CODCTR = :CODCTR
      INTO :CODREG, :NAMEREG, :CENTER
  DO
    SUSPEND;
  END
END ^
```

В этой процедуре нам с вами все понятно. Если входной параметр имеет пустое значение, то выбираются все регионы, иначе — только регионы соответствующей, заданной входным параметром, страны.

Сложное получение списка организаций

Когда мы проектировали наши таблицы, в таблице организаций мы ввели признак размещения организации. Тогда я пообещал, что позже мы напишем хранимую процедуру выбора, позволяющую надлежащим образом формировать запись набора данных, где будут присутствовать либо название регионального центра, либо название региона, а также название района или название районного центра (либо пустое значение). Настала пора выполнить обещание.

Напомню структуру таблицы предприятий `ORGANIZATION` — листинг 9.27.

Листинг 9.27. Структура таблицы организаций

```
    /*** Список организаций ***/
CREATE TABLE ORGANIZATION
( COD         D_INTEGER NOT NULL,    /* Код организации */
  CODCTR      D_CHAR3,               /* Код страны */
  CODREG      D_CHAR2,               /* Код региона */
  CODAREA     D_CHAR2,               /* Код района */
  LOCATION    D_CHAR1 DEFAULT '0',   /* Признак адреса: */
                                     /* 0 – областной (региональный) центр, */
                                     /* 1 – районный центр, */
                                     /* 2 – район. */
  NAME        D_CHAR60,              /* Название организации */
  CODFORMORG D_CHAR2 DEFAULT '00', /* Организационно-правовая форма */
  CONSTRAINT PK_ORGANIZATION PRIMARY KEY (COD),
  CONSTRAINT CH_ORGANIZATION CHECK (LOCATION IN ('0', '1', '2')),
  CONSTRAINT FK1_ORGANIZATION
    FOREIGN KEY (CODCTR, CODREG) REFERENCES REFREG (CODCTR, CODREG)
      ON DELETE SET NULL
      ON UPDATE CASCADE,
  CONSTRAINT FK2_ORGANIZATION
    FOREIGN KEY (CODFORMORG) REFERENCES REFFORMORG (COD)
      ON DELETE SET NULL
      ON UPDATE CASCADE
);
CREATE GENERATOR GEN_ORGANIZATION;
COMMIT;
```

В этой таблице столбец `LOCATION` задает признак размещения организации (признак адреса):

- 0 — областной центр;
- 1 — районный центр;
- 2 — район.

Наша задача сейчас — написать хранимую процедуру выбора, которая в зависимости от значения этого признака будет формировать в выходном наборе в одном поле либо название регионального центра, либо название региона, а для другого поля — название районного центра, название района или пустое значение.

Давайте четко сформулируем постановку задачи. От правильной постановки зависит и правильность решения.

Постановка задачи

Создать хранимую процедуру выбора, которая на основании таблицы организаций будет формировать набор данных, содержащий перечисленные ниже столбцы. Для каждого столбца указывается человеческое название, имя выходного параметра процедуры и способ получения значения.

- Название организации — `NAME`. Выбирается обычным образом из таблицы организаций `ORGANIZATION`.
- Страна размещения — `COUNTRYNAME`. Отыскивается на основании значения внешнего ключа `CODCTR` в справочнике стран `REFCTR` при использовании левого внешнего соединения.
- Регион — `REGION`. Отыскивается в справочнике регионов `REFREG`:
  - если признак адреса имеет значение 0, в поле помещается название центра региона (поле `CENTER` таблицы `REFREG`);
  - если же признак адреса имеет значение 1 или 2, в поле помещается название региона (поле `NAMEREG` таблицы `REFREG`).
- Район — `AREA`. Отыскивается в справочнике районов `REFAREA`:
  - если признак адреса имеет значение 0, в поле помещается `NULL`;
  - если признак адреса 1, в поле помещается название районного центра (поле `CENTER` таблицы `REFAREA`).;
  - если признак адреса имеет значение 2, в поле помещается название района (поле `NAMEAREA` таблицы `REFAREA`).
- Код региона — `CODREG`. Выбирается из таблицы организаций. Его присутствие в списке выходных параметров нужно лишь для того, чтобы при обращении к нашей процедуре выбора пользователь мог бы задать в условиях поиска предложения `WHERE` выбор организаций, находящихся в конкретном регионе (регионах). По-хорошему, для аналогичных целей следовало бы также поместить в список выходных параметров и код страны, и код района. Также можно было бы включить и код формы собственности. Часто пользователи, выбирая себе организации для возможного сотрудничества, учитывают и их форму собственности.

  Замечание

  Многие не хотят работать, например, с государственными предприятиями. В нашей базе данных не указывается форма собственности для организаций, хотя такой справочник присутствует — `REFFORMPROP`. В него помещены некоторые данные. В своей реальной базе данных вы можете в таблице организаций создать внешний ключ, ссылающийся на этот справочник.

- Организационно-правовая форма — `FORMORG`. Отыскивается на основании значения внешнего ключа `CODFORMORG` в справочнике организационно-

правовых форм REFFORMORG с использованием внешнего левого соединения.

Здесь мы четко расписали, что хотим получить от нашей хранимой процедуры.

Приступим к реализации.

Решение задачи

Далее все достаточно просто. Реализуем наши знания в области языка SQL, языка хранимых процедур и триггеров. Используем наши обычные знания, умения и навыки написания программ.

Как писал в свое время Карел Чапек, главное — придумать название. Назовем нашу процедуру PROC_ORGANIZATION. Входные параметры ей не нужны, пять выходных параметров мы описали чуть раньше. Все они имеют символьный тип данных, нужно только уточнить размеры. Для параметров, которые могут хранить данные из разных столбцов таблиц, разумеется, нужно указать наибольший размер.

В результате заголовок будет выглядеть следующим образом:

```
CREATE PROCEDURE PROC_ORGANIZATION
  RETURNS (NAME VARCHAR(60),
           COUNTRYNAME VARCHAR(30),
           REGION VARCHAR(40),
           AREA VARCHAR(35),
           CODREG CHAR(2),
           FORMORG VARCHAR(120))
AS
```

В основе выполняемой части процедуры будет цикл FOR-SELECT-DO. В нем оператор SELECT выбирает строки из таблицы организаций и помещает данные во внутренние переменные. Поскольку для поиска страны, в которой располагается организация, нам не потребуется императивных средств, мы сразу получим краткое название страны, используя левое внешнее объединение таблицы организаций с таблицей стран.

Поскольку получение организационно-правовой формы мы можем выполнить в одном операторе SELECT, мы добавляем к операции выборки еще одно левое внешнее объединение главной таблицы со справочной таблицей организационно-правовых форм REFFORMORG.

Так как в дальнейшем нам понадобится выполнять поисковые действия с использованием прочитанных данных, для их хранения мы опишем в процедуре локальные переменные:

```
DECLARE VARIABLE CODCTR CHAR(3);   /* Код страны */
```

```
DECLARE VARIABLE CODAREA CHAR(2);  /* Код района */
DECLARE VARIABLE LOCATION CHAR(1); /* Признак адреса */
```

Итак, основной цикл:

```
  FOR SELECT O.NAME,
             O.CODCTR,
             O.CODREG,
             O.CODAREA,
             O.LOCATION,
             C.NAME,
             F.NAME
        FROM ORGANIZATION O
          LEFT OUTER JOIN REFCTR C
            ON O.CODCTR = C.CODCTR
          LEFT OUTER JOIN REFFORMORG F
            ON O.CODFORMORG = F.COD
      INTO :NAME, :CODCTR, :CODREG, :CODAREA, :LOCATION,
           :COUNTRYNAME, :FORMORG
  DO
  ...
```

Заметьте, во избежание лишних недоразумений в операторе `SELECT` мы для каждой таблицы задаем псевдоним и используем его для уточнения всех имен столбцов таблиц, участвующих в операторе `SELECT`.

Окончательный текст нашей процедуры представлен в листинге 9.28.

Листинг 9.28. Хранимая процедура "интеллектуального" выбора организаций

```
CREATE PROCEDURE PROC_ORGANIZATION
  RETURNS (NAME VARCHAR(60),
           COUNTRYNAME VARCHAR(30),
           REGION VARCHAR(40),
           AREA VARCHAR(35),
           CODREG CHAR(2),
           FORMORG VARCHAR(120))
AS
DECLARE VARIABLE CODCTR CHAR(3);   /* Код страны */
DECLARE VARIABLE CODAREA CHAR(2);  /* Код района */
DECLARE VARIABLE LOCATION CHAR(1); /* Признак адреса */
BEGIN
  FOR SELECT O.NAME,
             O.CODCTR,
             O.CODREG,
```

```
          O.CODAREA,
          O.LOCATION,
          C.NAME,
          F.NAME
      FROM ORGANIZATION O
        LEFT OUTER JOIN REFCTR C
          ON O.CODCTR = C.CODCTR
        LEFT OUTER JOIN REFFORMORG F
          ON O.CODFORMORG = F.COD
    INTO :NAME, :CODCTR, :CODREG, :CODAREA, :LOCATION,
         :COUNTRYNAME, :FORMORG
DO
BEGIN
  IF (LOCATION = '0') THEN  /* Региональный центр */
  BEGIN
    SELECT CENTER
      FROM REFREG
      WHERE CODCTR = :CODCTR AND CODREG = :CODREG
    INTO :REGION;
  END
  IF (LOCATION > '0') THEN  /* Районный центр или район */
  BEGIN
    SELECT NAMEREG
      FROM REFREG
      WHERE CODCTR = :CODCTR AND CODREG = :CODREG
    INTO :REGION;
  END
  IF (LOCATION = '0') THEN  /* Региональный центр */
    AREA = NULL;
  IF (LOCATION = '1') THEN  /* Районный центр */
  BEGIN
    SELECT CENTER
      FROM REFAREA
      WHERE CODCTR = :CODCTR AND
            CODREG = :CODREG AND
            CODAREA = :CODAREA
    INTO :AREA;
  END
  IF (LOCATION = '2') THEN  /* Район */
  BEGIN
    SELECT NAMEAREA
      FROM REFAREA
      WHERE CODCTR = :CODCTR AND
            CODREG = :CODREG AND
```

```
                CODAREA = :CODAREA
        INTO :AREA;
    END
    SUSPEND;
  END
END ^
```

> ***Замечание***
>
> В некоторых случаях сервер базы данных выполняет неявное преобразование данных. Например, при проверке значения поля `LOCATION` в операторах `IF` можно было бы использовать не строковые константы, как мы это сделали в нашей процедуре, а числовые. Это также сработает, я проверял. Однако рекомендую особенно не рассчитывать на неявные преобразования и использовать соответствующие литералы и явное преобразование типов функцией `CAST`.

Основная "интеллектуальная" работа выполняется после главного оператора `SELECT`. Эту программную часть я написал навскидку. Может быть, есть более элегантный способ получения нужных значений. Например, можно было бы проверять признак адреса, и в зависимости от его значения получать сразу все необходимые данные по названию регионов и районов.

Проверим работу нашей процедуры.

Вначале выполните оператор:

```
SELECT  *
FROM   PROC_ORGANIZATION;
```

Вы получите список из 33 организаций, хранящихся в нашей базе данных. В структуре записи будут присутствовать и коды, которые нам не очень-то и нужны.

Давайте скорректируем оператор выборки, задав только нужные реквизиты, установив правильные заголовки и ограничив список только организациями, располагающимися в Саратове и Саратовской области. Так как все организации находятся в России, нет необходимости выводить и название страны.

Оператор вызова процедуры будет таким:

```
SELECT NAME AS "Организация",
       REGION AS "Регион",
       AREA AS "Район"
FROM   PROC_ORGANIZATION
WHERE CODREG = '64'
ORDER BY NAME;
```

В результате мы получим 21 запись организаций (листинг 9.29), располагающихся в Саратове или Саратовской области (код региона 64), которые будут упорядочены по названиям.

Листинг 9.29. Результат выполнения процедуры

```
Организация                                          Регион              Район
==================================================== =================== =======
ЗАО "Саратовский авиационный завод"                  САРАТОВ             NULL
ВОЛГОИНВЕСТБАНК                                      САРАТОВ             NULL
ГАЗПРОМБАНК САРАТОВ                                  САРАТОВ             NULL
ГП "Приборомеханический завод"                       САРАТОВ             NULL
ГП "САРАТОВСКИЙ АГРЕГАТНЫЙ ЗАВОД"                    САРАТОВ             NULL
Детский сад № 123                                    САРАТОВ             NULL
ЗАО "ИнфоТел"                                        САРАТОВ             NULL
МУП "Городской центр размещения рекламы              САРАТОВ             NULL
НАРАТБАНК ЭНГЕЛЬСКИЙ ФИЛИАЛ                          Саратовская область ЭНГЕЛЬС
ОАО "САРАТОВСКИЙ ЗАВОД ТЯЖЕЛЫХ ЗУБОРЕЗНЫХ СТАНКОВ"   САРАТОВ             NULL
ОАО "Саратовдизельаппарат"                           САРАТОВ             NULL
ОАО "Энергомаш"                                      САРАТОВ             NULL
ОАО "САРАТОВСКИЙ ЗАВОД ЗУБОСТРОГАЛЬНЫХ СТАНКОВ"      САРАТОВ             NULL
ОАО "САРАТОВСКИЙ ПОДШИПНИКОВЫЙ ЗАВОД"                САРАТОВ             NULL
ООО "Алан"                                           САРАТОВ             NULL
ООО "Какаду"                                         САРАТОВ             NULL
ООО "ПОДИУМ"                                         САРАТОВ             NULL
ООО" Инфомаркет"                                     САРАТОВ             NULL
Саратовское электроагрегатное производственное..     САРАТОВ             NULL
ЭКОНОМБАНК ЭНГЕЛЬС                                   Саратовская область ЭНГЕЛЬС
ЭНГЕЛЬС-БАНК                                         Саратовская область ЭНГЕЛЬС
```

В *приложении 4*, где описаны некоторые системные таблицы и программы просмотра этих таблиц, вы можете посмотреть, какие триггеры и хранимые процедуры существуют в нашей учебной базе данных. Вы можете увидеть в системных таблицах и тексты написанных нами триггеров и хранимых процедур.

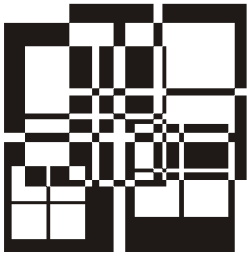
Что там за перевалом?

Триггеры и хранимые процедуры — мощные императивные средства обработки данных в реляционных базах данных. Мы довольно подробно рассмотрели эти средства. Еще раз вспомнили о возможностях оператора `SELECT`. Привели множество примеров.

Чуть позже мы еще вернемся к языку хранимых процедур и триггеров, когда станем рассматривать пример программы, использующей транзакции, работающие с двумя и более базами данных. Полученные здесь знания нам в скором времени потребуются опять в рамках этой книги, не говоря уж о дальнейшей реальной жизни.

На сегодняшний день мы получили знания во всех основных разделах реляционных баз данных. Теперь, когда мы можем решить практически любую задачу обработки данных, мы имеем право потренироваться в разработке весьма интересных и довольно сложных программ.

ГЛАВА 10



Для особо одаренных. Полезные программы работы с базой данных

В этой главе мы с вами напишем еще некоторое количество программ. Они будут несколько сложнее предыдущих.

Тексты программ этой главы находятся в каталоге Chapter10.

10.1. Программа работы со списком людей

Сведения о людях хранятся в базе данных в таблице PEOPLE. Ее структура представлена в листинге 10.1.

Листинг 10.1. Таблица данных по людям PEOPLE

```
    /*** Список людей ***/
CREATE TABLE PEOPLE
( COD       D_INTEGER NOT NULL,  /* Код человека */
  NAME1     D_CHAR15,            /* Имя */
  NAME2     D_CHAR15,            /* Отчество */
  NAME3     D_CHAR20,            /* Фамилия */
  BIRTHDAY  D_DATE,              /* Дата рождения */
  SEX       D_CHAR1 DEFAULT '0', /* Пол: */
                                 /* 0 — мужской, */
                                 /* 1 — женский. */
  FULLNAME  COMPUTED BY          /* Вычисляемый столбец */
    (NAME3 || ' ' || NAME1 || ' ' || NAME2),
  CODMOTHER    D_INTEGER,        /* Ссылка на мать */
  CODFATHER    D_INTEGER,        /* Ссылка на отца */
  CODOTHERHALF D_INTEGER,        /* Ссылка на супруга */
  CONSTRAINT PK_PEOPLE PRIMARY KEY (COD),
```

```
  CONSTRAINT CH_PEOPLE CHECK (SEX IN ('0', '1')),
  CONSTRAINT FK1_PEOPLE
    FOREIGN KEY (CODMOTHER) REFERENCES PEOPLE (COD)
      ON DELETE SET NULL,
  CONSTRAINT FK2_PEOPLE
    FOREIGN KEY (CODFATHER) REFERENCES PEOPLE (COD)
      ON DELETE SET NULL,
  CONSTRAINT FK3_PEOPLE
    FOREIGN KEY (CODOTHERHALF) REFERENCES PEOPLE (COD)
      ON DELETE SET NULL
);
CREATE GENERATOR GEN_PEOPLE;
COMMIT;
```

Если бы нам нужно было получить и отобразить в сетке DBGrid только данные, содержащиеся в таблице `PEOPLE`, мы использовали бы хорошо знакомые нам методы создания программ работы с одной таблицей. Например, см. в *главе 6* создание программы просмотра справочника стран.

Однако нашему пользователю, желающему просматривать список людей, не очень нужны никому непонятные коды супругов, отца и матери. Ему хотелось бы получить нормальный список, где присутствовали бы осмысленные данные.

В *главе 7* мы создали и проверили результаты выполнения оператора `SELECT`, который делает выборку надлежащего вида. Это может быть, например, такой оператор:

```
SELECT PG.FULLNAME AS "Фамилия, имя, отчество",
       PH.NAME3 AS "Супруг / супруга",
       PM.NAME3 AS "Мать",
       PF.NAME3 AS "Отец"
  FROM PEOPLE PG                      /* Главная таблица */
    LEFT OUTER JOIN PEOPLE PH         /* Супруг */
       ON PG.CODOTHERHALF = PH.COD
    LEFT OUTER JOIN PEOPLE PM         /* Мать */
       ON PG.CODMOTHER = PM.COD
    LEFT OUTER JOIN PEOPLE PF         /* Отец */
       ON PG.CODFATHER = PF.COD
  ORDER BY PG.FULLNAME;
```

Я убрал только из созданного ранее оператора проверку на непустое значение кодов супруга и родителей. В операторе применяется три левых внешних соединения для получения фамилий супруга и родителей. Здесь мы не используем внутреннее соединение, поскольку нашему заказчику важно получить данные по всем людям, независимо от того, есть ли для них данные о супругах и родителях, или нет.

Сейчас мы создадим полнофункциональную программу для работы со списком людей, полученным с использованием только что рассмотренного достаточно сложного оператора выборки. Подобный оператор потребует особого внимания к операторам, связанным с изменением, добавлением и обновлением записей.

Кроме того, в этих программах мы сделаем еще несколько полезных вещей, что приблизит наши программы к промышленному стандарту.

Как обычно будет создано две программы — с использованием компонентов FIBPlus и IBX.

10.1.1. Использование компонентов FIBPlus

В Delphi создайте новый проект. Присвойте форме имя `FormMain`, в свойстве `Caption` введите `Список людей`. Установите свойство `WindowState` в `wsMaximized`, чтобы при запуске приложения форма занимала всю поверхность экрана.

Создание элементов управления программой

Положите на форму панель, очистите поле `Caption`, выровняйте по верхнему краю. Это будет, как обычно, панель инструментов. Разместите на панели пять кнопок `SpeedButton` с вкладки **Additional**. Это будут кнопки быстрого доступа для добавления, изменения, удаления строк, обновления списка и выхода из программы.

Поместите на кнопки соответствующие иконки, задайте осмысленные имена и установите тексты подсказок (свойство `Hint`), как показано в табл. 10.1.

Таблица 10.1. *Кнопки быстрого доступа*

Имя	Подсказка
`BAdd`	Добавить (Shift+Ins)|Добавление новой записи
`BEdit`	Изменить (Enter)|Изменение текущей записи
`BDelete`	Удалить (Shift+Del)|Удаление текущей записи
`BRefresh`	Обновить (F5)|Обновление списка
`BExit`	Выйти|Выход из программы

Положите на форму компонент `ImageList` и заполните его иконками, соответствующими элементам меню.

Положите на форму компонент MainMenu и создайте в нем следующие элементы, задав подходящие клавиши быстрого доступа (свойство ShortCut у элементов меню) — см. рис. 10.1.

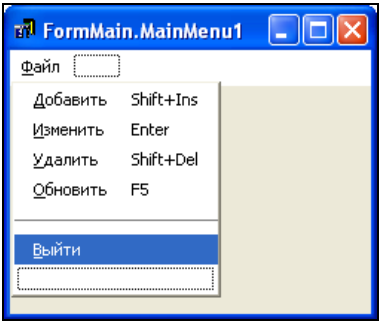


Рис. 10.1. Главное меню программы

Установите именам и подсказкам для элементов меню значения, которые показаны в табл. 10.2.

Таблица 10.2. *Элементы главного меню*

Имя	Подсказка
MFile	—
MAdd	Добавление новой записи
MEdit	Изменение текущей записи
MDelete	Удаление текущей записи
MRefresh	Обновление списка
MExit	Выход из программы

Свяжите главное меню с компонентом ImageList (свойство Images, имя компонента выбираем из выпадающего списка). Для каждого элемента меню задайте соответствующее значение свойству ImageIndex, чтобы у элемента появилась нужная иконка.

Положите на форму контекстное меню, создав два элемента — **Изменить** (PEdit) и **Удалить** (PDelete). Создайте необходимые подсказки. Свяжите с компонентом ImageList, хранящий иконки, и установите для обоих элементов нужное значение свойства ImageIndex.

С вкладки **Win32** положите на форму компонент строки состояния StatusBar. Установите ему нужные характеристики, как обычно мы делали в

предыдущих программах, чтобы в нем отображалось количество записей и подсказки.

Компоненты работы с базой данных

С вкладки **FIBPlus** положите на форму компонент базы данных, присвоив ему имя `Database1`, два компонента транзакции (`ReadTransaction` и `WriteTransaction`) и один компонент набора данных (`DSPeople`). С вкладки **DataAccess** положите компонент `DataSource` и, наконец, с вкладки **DataControls** компонент `DBGrid`, выровняв его по всей клиентской области.

Для компонента `Database1` установите обычным образом, как и в других программах, свойства, чтобы он ссылался на нашу учебную базу данных. Установите соответствующие связи с двумя транзакциями.

Значения свойств транзакций можно не изменять, убедитесь только, что их свойство `DefaultDatabase` ссылается на компонент базы данных `Database1`.

Компонент набора данных свяжите с компонентом базы данных (свойство `Database`, выбираем из выпадающего списка).

Компонент источника данных `DataSource1` свяжите с набором данных `DSPeople`, а компонент `DBGrid1` — с `DataSource1`.

Самая сложная и интересная работа с компонентом набора данных. В нем нам многое придется вводить вручную, поскольку он основывается не на простой таблице, а на многократном соединении таблиц.

Щелкните правой кнопкой мыши по компоненту `DSPeople` и выберите в контекстном меню элемент **SQL Editor** (редактор операторов SQL). Появится окно **SQL edit**, вкладка **Select SQL**. Здесь вместо нормального оператора SQL для одной таблицы нам придется ввести вручную операторы для множественных внешних левых соединений таблицы с собой же.

Введите в окне текст, показанный в листинге 10.2.

Листинг 10.2. Текст оператора `SELECT`

```
SELECT
    PO.COD,
    PO.NAME1,
    PO.NAME2,
    PO.NAME3,
    PO.BIRTHDAY,
    PO.SEX,
    PO.CODMOTHER,
    PO.CODFATHER,
```

```
    PO.CODOTHERHALF,
    PN1.NAME3,
    PN2.NAME3,
    PN3.NAME3
FROM
    PEOPLE PO
  LEFT OUTER JOIN PEOPLE PN1
     ON PO.CODOTHERHALF = PN1.COD
  LEFT OUTER JOIN PEOPLE PN2
     ON PO.CODFATHER = PN2.COD
  LEFT OUTER JOIN PEOPLE PN3
     ON PO.CODMOTHER = PN3.COD
ORDER BY PO.NAME3
```

Если после этого вы попытаетесь обычным образом сгенерировать операторы SQL для изменения данных в нужной таблице, то получите не совсем работоспособные операторы. Например, оператор для добавления новой записи будет выглядеть так, как показано в листинге 10.3.

Листинг 10.3. Текст неверно сгенерированного оператора `INSERT`

```
INSERT INTO PEOPLE(
    COD,
    NAME1,
    NAME2,
    NAME3,
    BIRTHDAY,
    SEX,
    CODMOTHER,
    CODFATHER,
    CODOTHERHALF,
    NAME3,
    NAME3,
    NAME3
)
VALUES(
    :COD,
    :NAME1,
    :NAME2,
    :NAME3,
    :BIRTHDAY,
    :SEX,
    :CODMOTHER,
```

```
    :CODFATHER,
    :CODOTHERHALF,
    :NAME31,
    :NAME32,
    :NAME33
)
```

Мы видим, что столбец с именем NAME3 в операторе встречается четыре раза. При этом ему во второй, третий и четвертый раз присваиваются значения странных параметров NAME31, NAME32 и NAME33. Программа пытается изменять и поля, полученные операцией соединения, чего делать, конечно же, не следует.

Такая же ситуация и с оператором изменения данных строки.

Так как мы используем внешнее левое соединение в нашем операторе SELECT, то система для соединенных данных, которые имеют те же имена, что и столбцы в "нормальных" операторах, присваивает свои имена. Это мы должны учесть, когда станем изменять операторы INSERT и UPDATE.

В операторе INSERT мы должны использовать только имена нашей главной таблицы. Введите для оператора INSERT вручную текст, показанный в листинге 10.4.

Листинг 10.4. Текст созданного вручную оператора INSERT

```
INSERT INTO PEOPLE(
    COD,
    NAME1,
    NAME2,
    NAME3,
    BIRTHDAY,
    SEX,
    CODMOTHER,
    CODFATHER,
    CODOTHERHALF
)
VALUES(
    :COD,
    :NAME1,
    :NAME2,
    :NAME3,
    :BIRTHDAY,
    :SEX,
```

```
    :CODMOTHER,
    :CODFATHER,
    :CODOTHERHALF
)
```

Оператор изменения текущей записи должен быть следующим — листинг 10.5.

Листинг 10.5. Текст оператора `UPDATE`

```
UPDATE PEOPLE
SET
    NAME1 = :NAME1,
    NAME2 = :NAME2,
    NAME3 = :NAME3,
    BIRTHDAY = :BIRTHDAY,
    SEX = :SEX,
    CODMOTHER = :CODMOTHER,
    CODFATHER = :CODFATHER,
    CODOTHERHALF = :CODOTHERHALF
WHERE
    COD = :OLD_COD
```

После добавления новой записи или изменения данных существующей строки таблицы `PEOPLE` выполняется так называемый `Refresh` — повторное чтение из базы данных только одной добавленной (или измененной) строки. В FIBPlus это оператор с тем же именем — `Refresh`. Посмотрим на его вид в листинге 10.6.

Листинг 10.6. Текст оператора типа `Refresh`

```
SELECT
    PO.COD,
    PO.NAME1,
    PO.NAME2,
    PO.NAME3,
    PO.BIRTHDAY,
    PO.SEX,
    PO.CODMOTHER,
    PO.CODFATHER,
    PO.CODOTHERHALF,
    PN1.NAME3,
    PN2.NAME3,
    PN3.NAME3
```

```
FROM
    PEOPLE PO
  LEFT OUTER JOIN PEOPLE PN1
     ON PO.CODOTHERHALF = PN1.COD
  LEFT OUTER JOIN PEOPLE PN2
     ON PO.CODFATHER = PN2.COD
  LEFT OUTER JOIN PEOPLE PN3
     ON PO.CODMOTHER = PN3.COD
 WHERE
        PO.COD = :OLD_COD
```

Он соответствует нашему оператору `SELECT` для "основной" выборки данных. Разумеется, поскольку выбирается ровно одна строка, сюда добавлено предложение `WHERE`, которое содержит условие выборки этой строки.

Оператор удаления также сгенерирован правильно (трудно представить, какие сложности могли бы возникнуть при его создании).

Листинг 10.7. Текст оператора `DELETE`

```
DELETE FROM
    PEOPLE
WHERE
        COD = :OLD_COD
```

Мы потратили довольно много времени и сил, чтобы автоматически сгенерировать неправильные операторы SQL, а потом вручную их изменять, приводя к нормальному виду. На самом деле всего этого можно было бы избежать, если с самого начала при генерации операторов внимательно посмотреть на вкладку **Generate Modify SQLs**, где в правом списке представлены поля, которые будут включены в операторы `INSERT` и `UPDATE`. Нам нужно всего лишь снять отметку с последних трех полей `NAME31`, `NAME32` и `NAME33`. Для этого, держа нажатой клавишу <Ctrl>, щелкаем мышью по соответствующим строкам (рис. 10.2).

После щелчка по кнопке **Generate SQLs** мы сразу же получим верные операторы, выполняющие добавление новой записи и изменение существующей.

Замечания относительно различий версий серверов базы данных

Все только что описанное относится к Firebird версии 1.5. Все программы я в первую очередь проверял в работе именно с этой версией. Когда же приступил к проверке работы в версии 2.0, то получил несколько иные результаты. Существует два отличительных момента.

В версии Firebird 2.0 оператор `SELECT` этой программы не хотел правильно выполняться. Выдавалось сообщение о дублировании имен столбцов или что-то в

этом роде. Пришлось внести изменение в оператор SELECT программы, добавив для имен NAME3 задание псевдонимов. Список выбора выглядит следующим образом:

```
SELECT
    PO.COD,
    PO.NAME1,
    PO.NAME2,
    PO.NAME3,
    PO.BIRTHDAY,
    PO.SEX,
    PO.CODMOTHER,
    PO.CODFATHER,
    PO.CODOTHERHALF,
    PN1.NAME3 AS NAME31,
    PN2.NAME3 AS NAME32,
    PN3.NAME3 AS NAME33
FROM ...
```

Надо сказать, что при использовании компонентов IBX такого не наблюдалось.

Посмотрите на рис. 10.2. Такую картинку мы получаем при использовании Firebird 1.5. Для версии 2.0 имена NAME31, NAME32 и NAME33 в обоих списках отсутствуют. В компонентах IBX они все равно присутствуют.

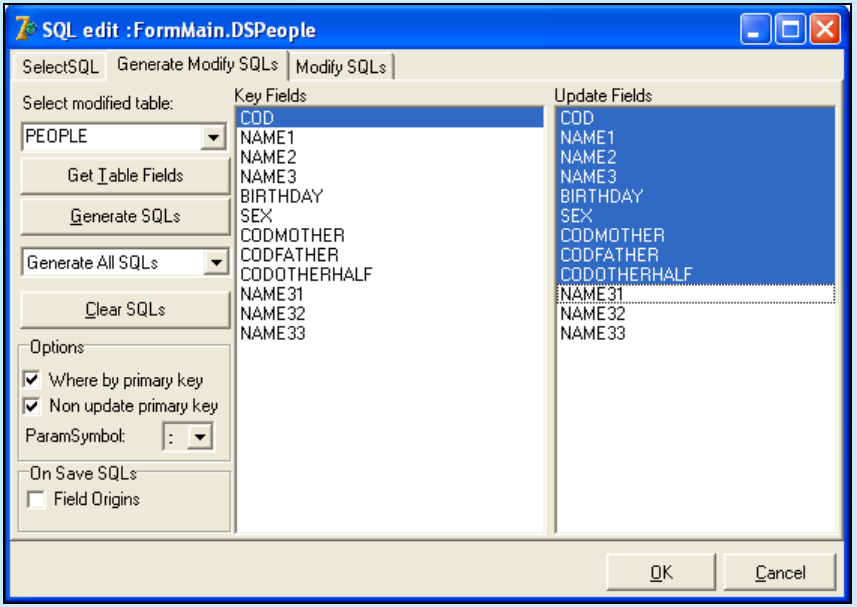


Рис. 10.2. Подготовка к генерации правильных операторов

В компонентах FIBPlus есть простые средства получения значения искусственного первичного ключа из генератора для вновь создаваемой записи. В *главе 9* я говорил об этом.

Выделите на форме компонент `DSPeople` и раскройте свойство `AutoUpdateOptions`, щелкнув мышью слева от имени свойства по символу +. Установите следующие значения подствойств, выполняя действия в указанном порядке:

- для `GeneratorName` (имя генератора базы данных) выберите из выпадающего списка имя соответствующего генератора — `GEN_PEOPLE`;
- значение подсвойства `GeneratorStep` (шаг приращения значения генератора при каждом к нему обращении) оставьте `1`;
- в подсвойстве `UpdateTableNames` выберите из списка *первую* таблицу `PEOPLE`. Именно первую, потому что она является "системообразующей" таблицей нашего оператора `SELECT`;
- для подсвойства `KeyFields` из выпадающего списка выберите имя столбца, первичного ключа, которому будет присваиваться значение, полученное из генератора `GEN_PEOPLE`. Это будет, конечно же, столбец с именем `COD`, первичный ключ нашей записи;
- для подсвойства `WhenGetGenID` (когда получать значение из генератора) выберите из выпадающего списка `wgBeforePost` (перед передачей записи на сервер);
- не забудьте свойство `AutoCommit` (автоматическое подтверждение транзакции после добавления или изменения записи) установить в `True`.

Создадим структуру компонента отображения нашего набора данных — `DBGrid`. Свяжите компонент с контекстным меню: из выпадающего списка у свойства `PopupMenu` выберите наше контекстное меню.

Щелкните правой кнопкой мыши по компоненту `DBGrid` и в контекстном меню выберите элемент **Columns Editor**.

Создайте обычным образом столбцы "Фамилия", "Имя", "Отчество", "Дата рождения".

Создайте столбец с заголовком "Пол". Сюда мы будем помещать не признак пола, а краткое название —"Муж." или "Жен.". Для этого столбца нужно задать такое имя поля (свойство `FieldName`), которого на самом деле не существует в нашей таблице. Наберем вручную в этом свойстве `SEX0`. Мы напишем специальный обработчик для отображения нужного текста.

Для следующего добавляемого в список столбца "Супруг/супруга" из выпадающего списка свойства `FieldName` (имя поля) выберите строку `NAME31`. Это имя, которое сервер базы данных присвоил полю, заданному в операторе `SELECT` в строке `PN1.NAME3`.

Для столбца "Отец" из выпадающего списка того же свойства `FieldName` выберите строку `NAME32`. И, наконец, для столбца "Мать" выберите `NAME33`.

Все приготовления, связанные с описанием базы данных, транзакций и набора данных, на этом закончены. Главная форма программы должна иметь вид, показанный на рис. 10.3.

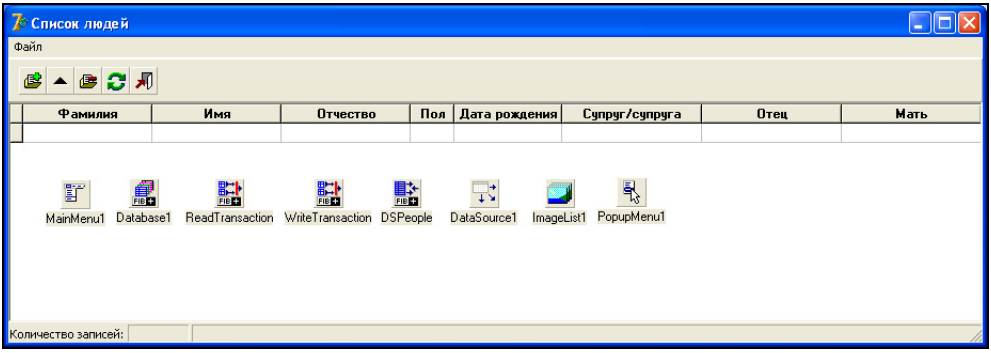


Рис. 10.3. Главная форма программы

Приступим к написанию процедур нашей программы.

Вначале создадим процедуру отображения подсказок. Текст готовим как обычно. Напомню порядок. Вначале в области **`private`** описания формы добавляем строку, описывающую процедуру помещения в строку состояния текста подсказки:

```
private
  procedure ShowHint(Sender: TObject);
```

Теперь пишем саму процедуру вывода подсказки (листинг 10.8).

Листинг 10.8. Вывод текста подсказки

```
procedure TFormMain.ShowHint(Sender: TObject);
begin
  StatusBar1.Panels[2].Text := Application.Hint;
end;
```

Создадим обработчик события активации формы (`OnActivate`), которое, как вы помните, происходит перед отображением формы (листинг 10.9).

Листинг 10.9. Начало работы программы

```
procedure TFormMain.FormActivate(Sender: TObject);
begin
  Database1.Connected := True;
  DSPeople.Open;
  StatusBar1.Panels.Items[1].Text := IntToStr(DSPeople.RecordCount);
  Application.OnHint := ShowHint;
end;
```

Здесь мы соединяемся с базой данных, открываем набор данных, выводим количество записей в строку состояния и задаем адрес процедуры вывода подсказок программы.

При завершении работы программы мы должны закрыть набор данных и отсоединиться от базы данных (листинг 10.10).

Листинг 10.10. Завершение работы программы

```
procedure TFormMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  DSPeople.Close;
  Database1.Connected := False;
end;
```

Еще одна простая процедура — обновление списка. Напишем обработчик щелчка по кнопке **Обновить** и свяжем с ним выбор соответствующего элемента меню (листинг 10.11).

Листинг 10.11. Обновление списка записей

```
procedure TFormMain.BRefreshClick(Sender: TObject);
begin
  DSPeople.FullRefresh;
  StatusBar1.Panels[1].Text := IntToStr(DSPeople.RecordCount);
end;
```

Теперь напишем процедуру для вывода текста в столбец "Пол". Выделите на форме компонент `DBGrid`, перейдите в Инспекторе объектов на вкладку **Events** и дважды щелкните мышью справа от события `OnDrawColumnCell`. Это событие возникает при прорисовке каждой ячейки в сетке отображения строк таблицы.

Напишите обработчик этого события, как показано в листинге 10.12.

Листинг 10.12. Вывод текста для отображения пола человека

```
procedure TFormMain.DBGrid1DrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var S: String;
begin
  if (Column.FieldName = 'SEX0') then
  begin
    if DSPeople.FieldByName('SEX').AsString = '0' then
      S := 'Муж.'
    else
      S := 'Жен.';
    DBGrid1.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
  end
end;
```

Вначале объявляется переменная строкового типа `S`, в которую будет помещаться выводимый текст. В тексте процедуры проверяется, является ли прорисовываемый столбец столбцом с именем `SEX0`. Далее на основании значения признака пола формируется и помещается в ячейку нужный текст.

У компонента `DBGrid` есть замечательное свойство `Canvas`, позволяющее выполнять на компоненте рисунки, выводить текст, менять цвета. Если вы работали с этим свойством, то знаете его необыкновенно интересные, но в то же время и сложные возможности. В нашей процедуре мы используем метод `TextOut`, который позволяет размещать по указанным координатам текст. Необходимые координаты передаются процедуре в константном параметре `Rect` типа `TRect`. Мы использовали координаты левого верхнего угла этого прямоугольника `Rect.Left` и `Rect.Top`.

Мы специально в сетке `DBGrid` присвоили столбцу имя, отсутствующее среди имен столбцов нашей таблицы из базы данных. Иначе в ячейку выводились бы и символы 0 или 1 и сформированный в процедуре текст, что выглядит некрасиво.

Теперь начнем создавать средства для изменения данных нашего набора данных.

Напишите обработчик события щелчка по кнопке удаления (листинг 10.13) и свяжите с ним соответствующие элементы главного и контекстного меню. Эта процедура нам с вами хорошо известна по предыдущим программам.

Листинг 10.13. Удаление текущей или группы отмеченных записей

```
procedure TFormMain.BDeleteClick(Sender: TObject);
var i: Integer;
begin
  if DSPeople.RecordCount = 0 then exit;
  if DBGrid1.SelectedRows.Count > 1 then
  begin
    if Application.MessageBox(PAnsiChar('Удалить отмеченные записи: ' +
       IntToStr(DBGrid1.SelectedRows.Count) + '?'),
       'Удаление', MB_OKCANCEL + MB_ICONQUESTION) = mrOk then
    begin
      for i := 0 to DBGrid1.SelectedRows.Count – 1 do
      begin
        DSPeople.GotoBookmark(pointer(DBGrid1.SelectedRows.Items[i]));
        DSPeople.Delete;
      end;
    end;
  end
  else
  begin
    if Application.MessageBox('Удалить текущую запись?',
       'Удаление', MB_YESNO + MB_ICONQUESTION) = mrYes then
      DSPeople.Delete;
  end;
  DSPeople.FullRefresh;
  StatusBar1.Panels[1].Text := IntToStr(DSPeople.RecordCount);
end;
```

Здесь после удаления одной или более записей необходимо выполнить повторное чтение всех записей набора данных, потому что в результате удаления записи у некоторых записей могут быть установлены в `NULL` значения внешних ключей, которые ссылались на удаленные записи.

Создадим форму для добавления в таблицу новой записи.

Выберите в меню **File | New | Form**. Появится новая форма. Присвойте ей имя `FormAddPeople` и сохраните модуль (меню **File | Save As**) с именем AddPeople.

> ***Замечание по именованию модулей***
>
> То имя модуля, которое мы сейчас использовали, больше всего соответствует правилам английского языка. Однако в своей практике я предпочитаю выполнять перестановку, т. е. в подобном случае использовать имя PeopleAdd. Это приводит к тому, что при отображении модулей проекта в алфавитном порядке в программе Мой компьютер или в Total Commander все модули, относящиеся к одной таблице, будут располагаться рядом: PeopleAdd, PeopleEdit, PeopleView и т. д.

В результате в наш проект будет добавлена новая форма. По умолчанию каждая форма проекта при запуске полученной программы на выполнение будет создаваться автоматически. Это хорошо, потому что освобождает нас от необходимости явно создавать каждую форму. Однако если в вашем реальном проекте присутствует около 100 форм или хотя бы даже только 50 (что обычно для многих достаточно серьезных программ), то при выполнении такой программы на компьютере может просто не хватить ресурсов вычислительной системы для такого количества сложных объектов.

По этой причине нормальной практикой является динамическое создание по мере необходимости формы, а после того, как отработает соответствующая функция, форма уничтожается, высвобождая ресурсы вычислительной системы.

Хотя в создаваемой сейчас нашей программе небольшое количество форм, мы с этого момента начнем практиковать использование правильного механизма создания и удаления форм.

Вызовите окно **Project Options** (меню **Project | Options** (Проект | Режимы)) (рис. 10.4).

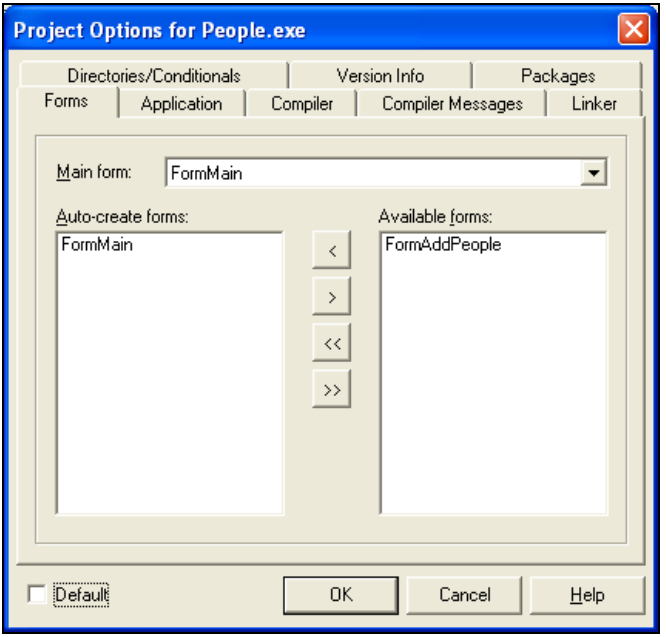


Рис. 10.4. Окно **Project Options**

Выберите вкладку **Forms** (впрочем, она и так будет текущей). Основную часть этой вкладки составляют два списка. Слева находится список автоматически создаваемых форм (**Auto-create forms**). Справа расположен список

остальных доступных форм (**Available forms**). Перед использованием формы из этого правого списка вы вначале должны ее создать, а после использования — удалить.

Наша главная форма (`FormMain`), разумеется, должна создаваться автоматически. Остальные же формы нашей программы имеет смысл создавать вручную. Выделите мышью в левом списке форму `FormAddPeople` и щелкните по кнопке >. Эта форма будет перемещена в правый список. Щелкните по кнопке **OK**.

Давайте сразу же напишем программный код обращения к этой форме из главного модуля. Неважно, что сама форма добавления еще нефункциональна. Напишите обработчик события щелчка мышью по кнопке добавления и свяжите его с обработчиком выбора в главном меню элемента **Добавить** (листинг 10.14).

Но сначала свяжите главную форму с формой добавления. Для этого в главной форме нужно в меню **File** выбрать элемент **Use Unit** (Используемые модули) и в появившемся окне выбрать пока единственный дополнительный модуль AddPeople.

Листинг 10.14. Вызов формы добавления новой записи

```
procedure TFormMain.BAddClick(Sender: TObject);
begin
 FormAddPeople := TFormAddPeople.Create(Application);
 if FormAddPeople.ShowModal = mrOk then
 begin
   DSPeople.FullRefresh;
   StatusBar1.Panels.Items[1].Text := IntToStr(DSPeople.RecordCount);
 end;
 FormAddPeople.Free;
end;
```

Здесь перед обращением к форме выполняется ее создание:

```
FormAddPeople := TFormAddPeople.Create(Application);
```

Если вы пытались подробнее разобраться в объектно-ориентированном программировании (необыкновенно интересная область программистской культуры), то вы знаете, что метод `Create` всегда является методом класса, а не объекта. Его можно использовать, когда ни одного объекта данного класса еще не существует. Этот метод создает новый объект, в нашем случае создается объект класса `TFormAddPeople`, т. е. нужная нам форма.

Далее происходит обычное модальное обращение к форме с использованием метода `ShowModal`. Если пользователь вышел из формы щелчком по кнопке

OK, то в программе осуществляется обновление списка и новое количество записей помещается в строку состояния.

В конце обязательно нужно удалить созданную форму и освободить ресурсы вычислительной системы:

```
FormAddPeople.Free;
```

Не забывайте уничтожать отработавшую форму! Иначе у вас со временем могут возникнуть крупные неприятности в работе[14].

Вернемся к форме добавления. Выполним установки необходимых характеристик диалоговой формы. Задайте заголовок (`Caption`) `Добавление данных о человеке`. `BorderStyle` установите в `bsDialog`, чтобы сделать окно диалоговым с неизменяемыми размерами. Размещение формы (`Position`) задайте в центре экрана, `poScreenCenter`.

Разместите на форме метки, описывающие названия вводимых пользователем данных, а также три компонента `Edit` для ввода фамилии, имени и отчества, задав для них, соответственно, имена `NAME3`, `NAME1` и `NAME2`. Укажите максимально возможное количество символов для каждого поля — 20, 15, 15. Эти числа вводятся в свойство `MaxLength`. Для выбора пола человека используем радиогруппу. Положите с вкладки **Standard** компонент `RadioGroup`, установив подходящие размеры. Задайте для этого компонента количество столбцов (свойства `Columns`) равное двум, в свойство `Items` (элементы) поместите две строки:

```
Мужской
Женский
```

Свойству `ItemIndex` присвойте значение 0, чтобы отмеченным был первый элемент списка, мужской пол.

Для ввода даты рождения используем, во-первых, `CheckBox`, присвоив ему имя `CBDate`, во-вторых, очень красивый и удобный компонент `DateTimePicker` с вкладки **Win32**. Компонент флажок (`CheckBox`) будет использоваться, чтобы указать, известна ли нам дата рождения. Другой компонент (присвойте ему имя `BIRTHDAY`) позволит выбирать конкретную дату, если пользователем отмечен флажок `CheckBox`. Сделайте компонент недоступным — установите свойство `Enabled` в `False`. Он будет переводиться в доступное состояние, если пользователь отметит флажок `CBDate`.

---

[14] Когда мы все перейдем работать на технологию .NET, проблемы с освобождением ресурсов отпадут, так как они будут освобождаться автоматически. А пока ведем себя прилично, ресурсы освобождаем. Как говорит один мой знакомый программист: "Поел — убери за собой посуду".

Для сведений о родителях и супруге положим еще три метки и три компонента `Edit`, присвоив им, соответственно, имена `MOTHER`, `FATHER` и `OTHERHALF`. Их задачей является отображение фамилий соответствующих родственников. Пользователю мы не позволим вводить туда свои данные (он будет только выбирать нужные строки из списков), поэтому всем этим полям зададим значение `True` в свойстве `ReadOnly` (только для чтения).

Справа от этих элементов редактирования поместим кнопки обзора (`Button`), щелчок по которым позволит просматривать соответствующие списки для выбора нужного значения.

Положим также две кнопки (`Button`), задав надписи `OK` и `Отменить`. Кнопку **OK** установим в недоступное состояние (`Enabled` = `False`). Ее мы будем делать доступной только в том случае, если пользователь в полях имени, отчества и фамилии ввел непробельные символы. Для этой кнопки также установите свойство `Default` в `True`, чтобы она "срабатывала", когда пользователь нажмет клавишу <Enter>, ее свойство `ModalResult` установите в `mrOk`, чтобы сообщить вызвавшей программе о выборе пользователя.

> ***Замечание***
>
> В подобной ситуации установление значения `ModalResult` для кнопки **OK** не является хорошей идеей. При попытке записать новую строку в базу данных могут возникнуть ошибки. В этом случае не следовало бы выходить из формы добавления.

Для кнопки **Отменить** установите свойство `Cancel` в `True` (нажатие клавиши <Esc>), `ModalResult` установите в `mrCancel`.

Форма примет приблизительно такой вид — рис. 10.5.

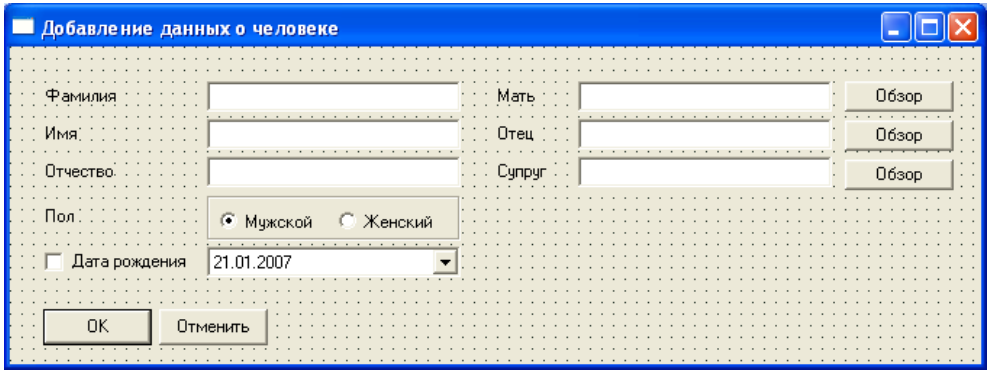


Рис. 10.5. Форма добавления данных о новом человеке

Напишем обработчик события `OnShow` для формы (листинг 10.15).

Листинг 10.15. Обработчик события OnShow формы добавления новой записи

```
procedure TFormAddPeople.FormShow(Sender: TObject);
begin
  NAME3.Text := '';
  NAME1.Text := '';
  NAME2.Text := '';
  SEX.ItemIndex := 0;
  BIRTHDAY.Date := Date();
  MOTHER.Text := '';
  FATHER.Text := '';
  OTHERHALF.Text := '';
end;
```

Всем полям редактирования мы присваиваем пустые значения, а в компонент для ввода даты рождения помещаем значение текущей даты.

Вообще говоря, полям редактирования можно было бы не присваивать пустых значений, потому что при создании формы для них уже будут установлены эти значения. Точно так же мы могли бы не задавать признак пола. Однако это хорошая практика явно формировать нужные значения. Тем более если вдруг вы не захотите создавать эту форму вручную, то такие присваивания будут обязательными.

Займемся установкой доступности кнопки **OK**. Ничего нового опять же здесь для нас с вами нет. Нечто подобное мы делали и раньше. Напишем, как показано в листинге 10.16, обработчик события `OnChange` (при изменении значения) для поля редактирования `NAME3` и свяжем этот обработчик с теми же событиями у элементов редактирования `NAME1` и `NAME2`.

Листинг 10.16. Обработчик события OnChange элементов редактирования

```
procedure TFormAddPeople.NAME3Change(Sender: TObject);
begin
  BOK.Enabled := (Trim(NAME3.Text) <> '') and
    (Trim(NAME2.Text) <> '') and
    (Trim(NAME1.Text) <> '');
end;
```

Эта процедура сделает кнопку доступной, если пользователь во все три поля ввел непробельные значения.

Еще один похожий обработчик события нам нужен, чтобы менять доступность компонента для ввода даты рождения. Это нужно сделать для события `OnClick` компонента `CBDate` — листинг 10.17.

Листинг 10.17. Обработчик события `OnClick` компонента `CBDate`

```
procedure TFormAddPeople.CBDateClick(Sender: TObject);
begin
  if CBDate.Checked then
    BIRTHDAY.Enabled := True
  else
    BIRTHDAY.Enabled := False;
end;
```

В этом обработчике мы выполняем установку доступности компонента, что называется "в лоб". Более элегантным решением, конечно же, будет один оператор:

```
BIRTHDAY.Enabled := CBDate.Checked;
```

Поля редактирования имен матери, отца и супруга мы сделали только для чтения, не дав возможности нашему пользователю вводить туда какие-либо данные. Эти сведения можно будет получать только через кнопки **Обзор**, которые вызовут другую форму, позволяющую из списка выбрать нужного человека. Кроме того, свойству `TabStop` этих полей задайте значение `False`, чтобы на них не попадал фокус ввода, когда пользователь на форме нажимает клавишу <Tab>. Думаю, вы помните, что можно выделить все эти объекты, держа нажатой клавишу <Shift> и щелкая мышью по нужным компонентам, а потом в Инспекторе объектов установить значение свойства `TabStop`, которое будет присвоено сразу всем выделенным компонентам.

В области `private` описания формы задайте три целочисленные переменные:

```
private
  CODMOTHER: Integer;
  CODFATHER: Integer;
  CODOTHERHALF: Integer;
```

Сюда будут помещаться коды выбранных родственников.

Как мы будем помещать в коды и поля редактирования нужные значения, рассмотрим чуть позже. Сейчас напишем обработчик события подтверждения добавления — щелчка по кнопке **OK** (листинг 10.18).

Листинг 10.18. Обработчик события подтверждения добавления

```
procedure TFormAddPeople.BOKClick(Sender: TObject);
begin
  FormMain.DSPeople.Insert;
  FormMain.DSPeople.FieldByName('NAME3').AsString :=
    NAME3.Text;
```

```
  FormMain.DSPeople.FieldByName('NAME1').AsString :=
    NAME1.Text;
  FormMain.DSPeople.FieldByName('NAME2').AsString :=
    NAME2.Text;
  FormMain.DSPeople.FieldByName('SEX').AsString :=
    IntToStr(SEX.ItemIndex);
  if CBDate.Checked then
    FormMain.DSPeople.FieldByName('BIRTHDAY').AsDateTime :=
      BIRTHDAY.Date;
  if MOTHER.Text <> '' then
    FormMain.DSPeople.FieldByName('CODMOTHER').AsInteger :=
      CODMOTHER;
  if FATHER.Text <> '' then
    FormMain.DSPeople.FieldByName('CODFATHER').AsInteger :=
      CODFATHER;
  if OTHERHALF.Text <> '' then
    FormMain.DSPeople.FieldByName('CODOTHERHALF').AsInteger :=
      CODOTHERHALF;
  FormMain.DSPeople.Post;
end;
```

Напомню, что кнопка **OK** будет доступной, если пользователь ввел непробельные значения во все три поля: фамилия, имя, отчество. Иначе никакими способами в этот обработчик попасть нельзя.

Вначале мы переводим набор данных в режим добавления новой строки (оператор `FormMain.DSPeople.Insert`). После этого присваиваем полям соответствующие значения.

Для родственников мы выполняем проверку на непустое значение поля редактирования, после чего присваиваем значение кода. Если же в поле редактирования нет текста, то значение коду не присваивается. В этом случае код, как и полагается, получит значение `NULL`.

Похожим образом мы поступаем и с датой рождения. Если пользователь отметил флажок **Дата рождения**, то полю присваивается выбранная дата.

Создадим, наконец, единую форму для выбора матери, отца или супруга.

Выберите в меню **File | New | Form**. Присвойте форме имя `FormParentView`, свяжите ее с главной формой `FormMain` (меню **File | Use Unit**).

Положите на форму уже привычные для нас компоненты, необходимые для выполнения просмотра списка людей. Получится форма следующего вида — рис. 10.6.

Компонент набора данных DSPeople свяжите с компонентом базы данных, находящимся на главной форме — из выпадающего списка **Database** выберите единственное присутствующее там значение **FormMain.Database1**. Главное для этого набора данных — задать оператор SELECT. Остальные операторы нам не нужны, поскольку никаких изменений в базе данных через этот компонент производиться не будет.

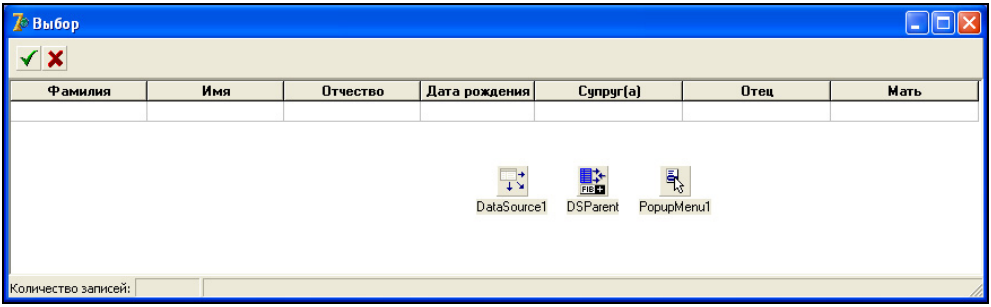


Рис. 10.6. Форма выбора родственников

Проще всего скопировать оператор SELECT из подобного набора данных, находящегося на главной форме, и перенести в новый набор данных (см. листинг 10.2).

В разделе **public** описания формы опишем целочисленную переменную.

```
public
  TypePeople: Integer;  // Тип человека:
                        // 0 — отец или муж,
                        // 1 — мать или жена.
```

В эту переменную (она доступна и внешним программам, потому что располагается в разделе public) вызывающая программа перед обращением к нашей форме будет помещать требуемый признак пола просматриваемых людей. Такую переменную можно рассматривать как входной параметр.

Обработчик события OnShow формы следует написать такой, как показано в листинге 10.19.

Листинг 10.19. Обработчик события OnShow формы просмотра родственников

```
procedure TFormParentView.FormShow(Sender: TObject);
var S: String;
begin
  case TypePeople of
    0: S := '(PO.SEX = ''0'')';
    1: S := '(PO.SEX = ''1'')';
  end;
```

```
  DSParent.MainWhereClause := S;
  DSParent.Open;
  StatusBar1.Panels.Items[1].Text := IntToStr(DSParent.RecordCount);
end;
```

Вначале проверяется значение переменной `TypePeople`, выполняющей роль входного параметра (для разнообразия я использовал здесь конструкцию **`case of`**, хотя можно было спокойно применять **`if`**). В результате формируется текст условия, которое будет использовано в предложении `WHERE` в нашем операторе `SELECT`.

> ***Замечание***
>
> Вместо оператора **`case`** или **`if`** Сергей Востриков предложил более элегантное решение:
>
> ```
> S := '(PO.SEX = ''' + IntToStr(TypePeople) + ''')';
> ```

Затем в оператор `SELECT` добавляется сформированное предложение `WHERE`. Для этого использовано очень удобное и полезное свойство компонента набора данных (существует в FIBPlus, но не в IBX) `MainWhereClause`. После этого открывается набор данных, и количество считанных записей помещается в строку состояния.

Положите на панель инструментов две кнопки (`SpeedButton`) — кнопку выбора и кнопку отмены. Для кнопки выбора напишите обработчик щелчка по этой кнопке — листинг 10.20.

Листинг 10.20. Обработчик щелчка по кнопке выбора записи

```
procedure TFormParentView.BSelectClick(Sender: TObject);
begin
  if DSParent.RecordCount <> 0 then
    ModalResult := mrOk
  else
    ModalResult := mrCancel;
end;
```

Обратите внимание, какими предусмотрительными мы с вами здесь оказались. Если пользователь щелкнул по кнопке выбора текущей записи, мы еще проверяем, есть ли в списке записи. Если список пуст, мы сообщаем вызвавшей программе, что пользователь отменил выбор. Это избавит нас от неприятностей в случае отсутствия записей в списке, хотя такое в нашей практике может встретиться весьма редко.

Для отмены просмотра в соответствующем обработчике (листинг 10.21) мы просто устанавливаем нужное значение переменной `ModalResult` (`mrCancel`).

Листинг 10.21. Обработчик отмены просмотра списка родственников

```
procedure TFormParentView.BCancelClick(Sender: TObject);
begin
  ModalResult := mrCancel;
end;
```

Положите на форму компонент `PopupMenu`. Создайте в нем два элемента — **Выбрать**, задав ему "горячую" клавишу <Enter>, и **Отменить** ("горячая" клавиша <Esc>). Свяжите выбор элементов с соответствующими обработчиками щелчка по кнопкам. Свяжите сетку `DBGrid` с этим всплывающим меню. Вообще говоря, немного противоестественно помещать элемент отмены просмотра в контекстное меню, потому как туда следует помещать только такие функциональные элементы, которые имеют отношение к одной текущей записи. Я это сделал только лишь для того, чтобы отмена просмотра происходила и после того, как пользователь нажмет клавишу <Esc>. В противном случае пришлось бы писать обработчик события `OnKeyPress` для формы, чего я очень не люблю по причине его некорректной работы при некоторых условиях.

Для события двойного щелчка по сетке `DBGrid` (`OnDblClick`) укажите уже созданный обработчик события выбора элемента.

В результате всех этих действий пользователь может выбрать текущую строку несколькими способами:

- щелчок по кнопке выбора;
- выбор в контекстном меню элемента **Выбрать**;
- нажатие клавиши <Enter>;
- двойной щелчок мышью по конкретной строке сетки.

Наш пользователь может выбрать для себя вариант по своему вкусу.

Вернитесь в форму добавления человека и напишите три процедуры, являющиеся реакцией на щелчки по кнопкам **Обзор** для матери, отца и супруга. Они очень похожи с учетом отличия в используемых именах (листинги 10.22—10.24).

Листинг 10.22. Щелчок по кнопке *Обзор* для матери

```
procedure TFormAddPeople.BMOTHERClick(Sender: TObject);
begin
  FormParentView := TFormParentView.Create(Application);
  FormParentView.TypePeople := 1;
  if FormParentView.ShowModal = mrOk then
```

```
  begin
    MOTHER.Text :=
      FormParentView.DSParent.FieldByName('NAME3').AsString;
    CODMOTHER :=
      FormParentView.DSParent.FieldByName('COD').AsInteger;
  end;
  FormParentView.DSParent.Close;
  FormParentView.Free;
end;
```

Листинг 10.23. Щелчок по кнопке *Обзор* для отца

```
procedure TFormAddPeople.BFATHERClick(Sender: TObject);
begin
  FormParentView := TFormParentView.Create(Application);
  FormParentView.TypePeople := 0;
  if FormParentView.ShowModal = mrOk then
  begin
    FATHER.Text :=
      FormParentView.DSParent.FieldByName('NAME3').AsString;
    CODFATHER :=
      FormParentView.DSParent.FieldByName('COD').AsInteger;
  end;
  FormParentView.DSParent.Close;
  FormParentView.Free;
end;
```

Листинг 10.24. Щелчок по кнопке *Обзор* для супруга

```
procedure TFormAddPeople.BOTHERHALFClick(Sender: TObject);
begin
  FormParentView := TFormParentView.Create(Application);
  if SEX.ItemIndex = 0 then
    FormParentView.TypePeople := 1
  else
    FormParentView.TypePeople := 0;
  if FormParentView.ShowModal = mrOk then
  begin
    OTHERHALF.Text :=
      FormParentView.DSParent.FieldByName('NAME3').AsString;
    CODOTHERHALF :=
      FormParentView.DSParent.FieldByName('COD').AsInteger;
  end;
  FormParentView.DSParent.Close;
```

```
  FormParentView.Free;
end;
```

Запустите программу на выполнение. Лучше не из среды Delphi, а как обычную программу, чтобы ошибочные ситуации, которые мы с вами скоро начнем создавать, обрабатывались не в Delphi, а только в нашей программе. Проверьте, как выполняется добавление. Для добавляемой записи выбирайте родителей, супругов из списка.

Обратите внимание на то, что после добавления нового человека и задания для него "второй половины" у этого второго человека также появляется имя только что введенного нами супруга. Мы в нашей программе ничего для этого не делали. Это срабатывает написанный нами ранее триггер, который выполняется после добавления новой записи, `TAI_PEOPLE`.

Удалите введенную строку. У второго супруга в списке тут же пропадает имя супруга. Здесь наши триггеры ни при чем. Это срабатывает автоматически созданный системой триггер на основании нашего описания внешнего ключа, ссылающегося на супруга. Напомню, как мы описали этот внешний ключ:

```
CONSTRAINT FK3_PEOPLE
  FOREIGN KEY (CODOTHERHALF) REFERENCES PEOPLE (COD)
    ON DELETE SET NULL
```

То есть при удалении строки супруга в нашей записи ссылка на супруга устанавливается в пустое значение `NULL`.

А теперь попробуйте при вводе нового человека задать для него в качестве супруга человека, у которого уже существует другой супруг. Вы немедленно получите сообщение в диалоговом окне: `У супруга установлена связь с другим человеком`. Это срабатывает наш триггер, выполняемый перед добавлением новой записи.

В диалоговом окне щелкните по кнопке **OK**. Тут же пропадет и диалоговое окно, и наша форма добавления. Дело в том, что для кнопки **OK** мы задали свойству `ModalResult` значение `mrOk`. Иными словами, модуль добавления независимо от наличия или отсутствия ошибок будет завершать работу, уверяя вызвавшую форму в том, что все в порядке.

Имеет смысл внести изменения в этот модуль. Свойству `ModalResult` следует установить значение `mrNone`. Последним оператором в обработчике щелчка по этой кнопке нужно написать:

```
ModalResult := mrOk;
```

В нашей программе можно сделать множество улучшений. Например, в главном окне в строке состояния поместить поле, где будет указываться количество отмеченных записей. Далее. При выводе списка претендентов "на руку" вновь вводимого человека можно было бы не включать людей, у которых уже есть супруг и т. д. Я всего хорошего не сделал в расчете на вашу помощь. Дерзайте, доведите программу до промышленного уровня.

Более того, я не создал средств для изменения значений записи человека. Форму для редактирования существующих записей я предлагаю вам написать самим. Своими результатами вы можете поделиться с нами. Присылайте ваши варианты на адрес издательства. Во избежание излишнего трафика отправляйте только файлы pas, dfm, dpr и res в сжатом формате, лучше zip.

Как вы понимаете, я, кончено же, схитрил. Относительно простую работу выполнил вместе с вами, а написание модуля корректировки существующих записей людей переложил на вас. Собственно, там, насколько я понимаю, могут возникнуть лишь проблемы с изменением некоторых значений внешних ключей на пустые. Потребуются какие-то средства для удаления из записей ссылок на родителей и супругов. Это коснется и пользовательского интерфейса. Короче, дерзайте.

Ничего удивительного и предосудительного в моем поведении нет. Если знаете, основное правило для преподавателя звучит приблизительно так:

"Никогда не пиши шпаргалки сам. Лучше отбери их у наиболее талантливых студентов. Это может сильно расширить твой кругозор".

Так что отправляйте ваши шпаргалки, т. е. программы. Буду расширять свой кругозор.

На этом мы закончили написание программы с использованием компонентов FIBPlus.

10.1.2. Использование компонентов IBX

Из созданной нами программы с использованием компонентов FIBPlus мы довольно легко получим программу, которая применяет только компоненты IBX. Ее также можно создать и полностью заново, с нуля.

Создадим новое приложение (**File | New | Application**). Разместим на форме нужные компоненты. С вкладки **InterBase** положим на форму компоненты базы данных, транзакции и набора данных.

Для базы данных задайте необходимые характеристики. Щелкните правой кнопкой мыши на компоненте и в контекстном меню выберите **Database**

Editor. В появившемся окне **Database Component Editor** введите данные, как показано на рис. 10.7.

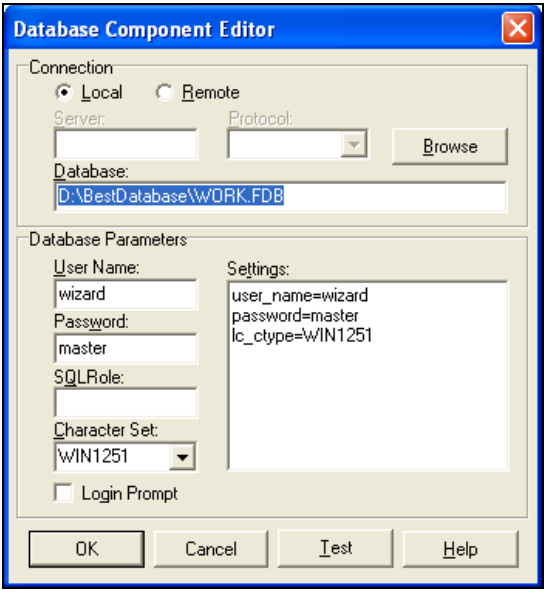


Рис. 10.7. Характеристики компонента базы данных

Это можно сделать и иначе. В диалоговом окне для свойства `Params` нужно ввести следующие три строки:

```
user_name=wizard
password=master
lc_ctype=WIN1251
```

Для компонента транзакции можно ничего не менять, оставив все значения по умолчанию.

Компонент набора данных свяжите с компонентом базы данных (свойство `Database`). Вызовите диалог **SelectSQL** и введите наш сложный оператор `SELECT`, как и в предыдущей версии программы (см. листинг 10.2).

После этого щелкните правой кнопкой мыши на компоненте и в меню выберите **Dataset Editor** (Редактор набора данных). Не станем повторять ошибок, допущенных в предыдущей версии программы, и сразу подготовимся к генерации правильных обновляющих операторов. В появившемся окне выделите мышью (держа нажатой клавишу <Ctrl>) в списках элементы, как показано на рис. 10.8.

Щелкните по кнопке **Generate SQL**. Будут сгенерированы необходимые операторы. Все, кроме `Refresh`, будут правильными. Для оператора повторного чтения текущей записи нужно вручную ввести текст, показанный в листинге 10.25.

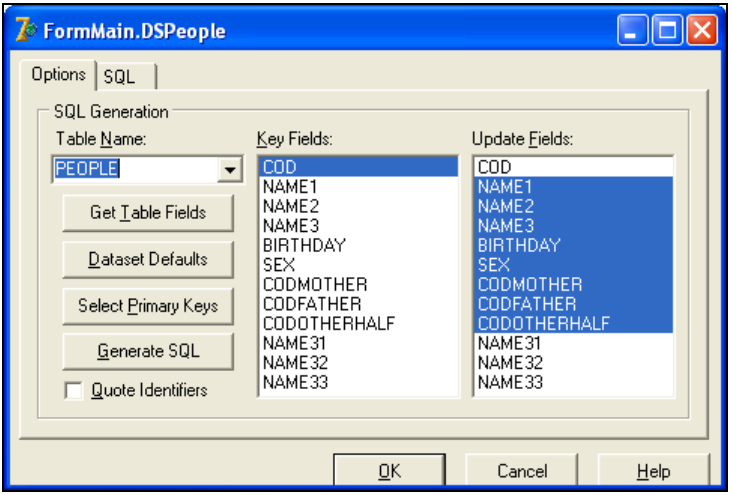


Рис. 10.8. Генерация операторов SQL

Листинг 10.25. Оператор чтения текущей записи

```
SELECT
    PO.COD,
    PO.NAME1,
    PO.NAME2,
    PO.NAME3,
    PO.BIRTHDAY,
    PO.SEX,
    PO.CODMOTHER,
    PO.CODFATHER,
    PO.CODOTHERHALF,
    PN1.NAME3,
    PN2.NAME3,
    PN3.NAME3
FROM
    PEOPLE PO
  LEFT OUTER JOIN PEOPLE PN1
     ON PO.CODOTHERHALF = PN1.COD
  LEFT OUTER JOIN PEOPLE PN2
     ON PO.CODFATHER = PN2.COD
```

```
  LEFT OUTER JOIN PEOPLE PN3
     ON PO.CODMOTHER = PN3.COD
where
  PO.COD = :OLD_COD
```

А вот в компонентах FIBPlus этот оператор генерируется правильно.

Для автоматического получения значения первичного ключа из генератора при добавлении в базу данных новой записи нужно вызвать диалоговое окно (рис. 10.9) в свойстве `GeneratorField`.

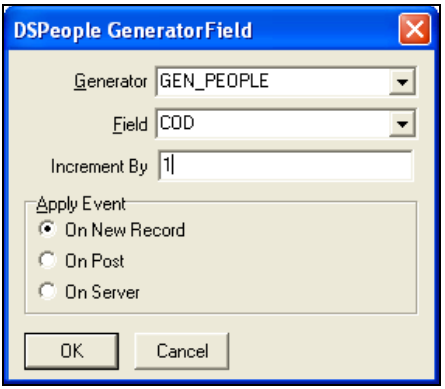


Рис. 10.9. Создание данных о первичном ключе в IBX

Из выпадающего списка генераторов выбираем "наш" генератор, а из выпадающего списка полей, поле `COD`.

Программный код нужно будет несколько подкорректировать с учетом отличия компонентов IBX от FIBPlus. Везде, где мы обращались к методу `FullRefresh`, нужно добавить целочисленную переменную для хранения первичного ключа текущей записи:

```
var Cod: Integer;
```

Само переоткрытие набора данных выполняется следующими операторами:

```
Cod := DSPeople.FieldByName('COD').AsInteger;
DSPeople.Close;
DSPeople.Open;
DSPeople.FetchAll;
DSPeople.Locate('COD', Cod, [loPartialKey]);
```

Здесь в локальной переменной `Cod` сохраняется значение первичного ключа текущей записи. После этого выполняется закрытие и повторное открытие

набора данных с загрузкой всех записей. Под конец при использовании метода `Locate` текущей делается нужная нам запись.

Текст модуля добавления практически не меняется. Нужно только при помещении записи в базу данных после вызова метода `Post` выполнить подтверждение транзакции, т. к. в IBX нет средств автоматического подтверждения транзакций. Добавьте следующую строку:

```
FormMain.Transaction1.CommitRetaining;
```

В модуле выбора человека из списка для компонента набора данных нужно сформировать только оператор `SELECT`. Он будет иметь точно такой же вид, как и в главном модуле программы. Впрочем, его можно и не создавать, потому что мы каждый раз заново создаем этот оператор при отображении формы. При использовании компонентов FIBPlus мы в операторе только меняли предложение `WHERE`. В IBX такого средства нет. Нам нужно самим в программе создать весь оператор `SELECT` с подходящим предложением `WHERE`. Текст обработчика представлен в листинге 10.26.

Листинг 10.26. Обработчик события `OnShow` в форме выбора человека

```
procedure TFormParentView.FormShow(Sender: TObject);
var S: String;
begin
  case TypePeople of
    0: S := 'WHERE PO.SEX = ''0''';
    1: S := 'WHERE PO.SEX = ''1''';
  end;
  DSParent.SelectSQL.Clear;
  DSParent.SelectSQL.Add('SELECT PO.COD,');
  DSParent.SelectSQL.Add('PO.NAME1,');
  DSParent.SelectSQL.Add('PO.NAME2,');
  DSParent.SelectSQL.Add('PO.NAME3,');
  DSParent.SelectSQL.Add('PO.BIRTHDAY,');
  DSParent.SelectSQL.Add('PO.SEX,');
  DSParent.SelectSQL.Add('PO.CODMOTHER,');
  DSParent.SelectSQL.Add('PO.CODFATHER,');
  DSParent.SelectSQL.Add('PO.CODOTHERHALF,');
  DSParent.SelectSQL.Add('PN1.NAME3,');
  DSParent.SelectSQL.Add('PN2.NAME3,');
  DSParent.SelectSQL.Add('PN3.NAME3');
  DSParent.SelectSQL.Add('FROM PEOPLE PO');
  DSParent.SelectSQL.Add('LEFT OUTER JOIN PEOPLE PN1');
  DSParent.SelectSQL.Add('ON PO.CODOTHERHALF = PN1.COD');
  DSParent.SelectSQL.Add('LEFT OUTER JOIN PEOPLE PN2');
```

```
  DSParent.SelectSQL.Add('ON PO.CODFATHER = PN2.COD');
  DSParent.SelectSQL.Add('LEFT OUTER JOIN PEOPLE PN3');
  DSParent.SelectSQL.Add('ON PO.CODMOTHER = PN3.COD');
  DSParent.SelectSQL.Add(S);
  DSParent.SelectSQL.Add('ORDER BY PO.NAME3');
  DSParent.Open;
  DSParent.FetchAll;
  StatusBar1.Panels.Items[1].Text := IntToStr(DSParent.RecordCount);
end;
```

Вначале очищается содержимое оператора `SELECT`, а затем в свойство `SelectSQL` добавляются необходимые строки.

Эта программа так же, как и предыдущая, имеет недостатки, которые вы можете смело устранить.

10.2. Программа работы с двумя базами данных

В *главе 8* мы рассматривали транзакции. Тогда я пообещал, что мы с вами напишем программу, которая будет использовать транзакцию, одновременно работающую с двумя базами данных. Настала пора выполнить обещание. Пусть только любители компонентов IBX на меня не обижаются. Я собираюсь написать программу лишь с использованием компонентов FIBPlus. Перенос такой программы на компоненты IBX потребует слишком много сил, а главное, времени.

Серверы баз данных InterBase и Firebird обеспечивают работу одной транзакции с несколькими базами данных. Особенности такой работы — двухфазное подтверждение транзакций, появление "зависших" (in limbo) транзакций — подробно описаны в литературе, и мне не хочется на этом сейчас останавливаться.

Наша программа продемонстрирует работу одной транзакции одновременно с двумя базами данных. Кроме того, нам понадобятся наши обширные знания в области языка триггеров и хранимых процедур.

Для демонстрационных целей, помимо основной базы данных Work.fdb, нам понадобится вторая база данных, которую так и назовем, Second.fdb. Для создания этой базы данных, всех ее объектов и заполнения таблицы данными вы можете использовать скрипт CreateSecondDatabase.sql.

В этой базе данных будет присутствовать только одна таблица PEOPLE, которая по своей структуре несколько отличается от такой же таблицы в основной базе данных. Структура таблицы приведена в листинге 10.27.

Листинг 10.27. Создание таблицы по людям и генератора во второй базе данных

```
/* Таблица данных по людям */
CREATE TABLE PEOPLE
( COD       D_INTEGER NOT NULL,  /* Код человека */
  NAME1     D_CHAR15,            /* Имя */
  NAME2     D_CHAR15,            /* Отчество */
  NAME3     D_CHAR20,            /* Фамилия */
  CODCTR    D_CHAR3,             /* Код страны */
  CONSTRAINT PK_PEOPLE PRIMARY KEY (COD)
);
CREATE GENERATOR GEN_PEOPLE;
COMMIT;
```

Здесь мы убрали лишние столбцы, не имеющие никакого отношения к целям нашей демонстрации, и добавили код страны, имеющий самое прямое отношение к целям нашей демонстрации. Создали генератор GEN_PEOPLE для генерации значения первичного ключа этой таблицы.

Напишем также обычный триггер автоматического получения значения первичного ключа из генератора — листинг 10.28.

Листинг 10.28. Создание триггера для получения первичного ключа

```
SET TERM ^;
/* Формирование значения первичного ключа для таблицы PEOPLE */
CREATE TRIGGER TBI_PEOPLE FOR PEOPLE
  BEFORE INSERT POSITION 0
AS
BEGIN
  IF (NEW.COD IS NULL) THEN
    NEW.COD = GEN_ID(GEN_PEOPLE, 1);
END ^
SET TERM ;^
COMMIT;
```

В таблицу добавлены практически те же записи, что и в аналогичную таблицу основной базы данных.

В записи присутствует столбец код страны CODCTR, который по сути своей является внешним ключом, ссылающимся на код страны в таблице REFCTR

(столбец CODCTR), находящейся в *другой* базе данных Work.fdb. Поскольку никакие связи на уровне структур данных, описываемых средствами SQL, между таблицами в разных базах данных невозможны, мы не можем указать здесь предложение FOREIGN KEY и молча использовать средства сервера базы данных для поддержания ссылочной целостности данных. Мы даже не можем написать триггер или хранимую процедуру для выполнения соответствующих действий, поскольку и триггер, и процедура могут работать только с одной, "своей", базой данных. Все это нам придется делать самостоятельно.

К счастью, в FIBPlus существуют подходящие средства, которыми мы сейчас воспользуемся.

Создадим новый проект, который назовем MultiBase. Положим на форму два компонента DBGrid — один для справочника стран, другой для списка людей, зададим им соответствующие значения выравнивания (свойство Align), положим разделитель Splitter. Разместим два компонента базы данных TpFIBDatabase, два компонента TpFIBTransaction, один для чтения, другой для обновления, два компонента набора данных и два компонента DataSource.

В обоих компонентах TpFIBDatabase установим свойство DefaultTransaction (транзакция по умолчанию, или, что то же самое, транзакция для чтения) в значение ReadTransaction, а свойство DefaultUpdateTransaction (обновляющая транзакция по умолчанию) в значение WriteTransaction. В результате этого внутренний список баз данных каждой транзакции (свойство Databases, являющееся массивом) будет содержать эти две базы. При запуске транзакций они будут стартовать сразу в двух базах каждая. Если мы поместим на форму десять компонентов баз данных, указав таким же образом две транзакции по умолчанию, то каждая транзакция будет запускаться в десяти базах данных.

Положим компонент главного меню и создадим два элемента для подтверждения (Commit) и для отмены (Rollback) обновляющей транзакции. Обработчики событий выбора этих элементов меню представлены в листингах 10.29 и 19.30.

Листинг 10.29. Подтверждение транзакции

```
procedure TFormMain.MCommitClick(Sender: TObject);
begin
  WriteTransaction.CommitRetaining;
  PersonData.FullRefresh;
end;
```

Листинг 10.30. Отмена транзакции

```
procedure TFormMain.MRollbackClick(Sender: TObject);
begin
  WriteTransaction.RollbackRetaining;
  CountryData.FullRefresh;
end;
```

Я добавил в меню еще один элемент — **Transact Databases**. В обработке вызова этого элемента отображаются имена баз данных обеих транзакций (листинг 10.31).

Листинг 10.31. Отображение списка баз данных обеих транзакций

```
procedure TFormMain.MTransactDatabasesClick(Sender: TObject);
var i: integer;
    S: String;
begin
  S := 'WriteTransaction'+#10#13;
  S := S + 'DatabaseCount: ' +
       IntToStr(WriteTransaction.DatabaseCount)+#10#13;
  for i := 0 to WriteTransaction.DatabaseCount – 1 do
    S := S + WriteTransaction.Databases[i].Name+#10#13;

  S := S+#10#13 + 'ReadTransaction'+#10#13;
  S := S + 'DatabaseCount: ' +
       IntToStr(ReadTransaction.DatabaseCount)+#10#13;
  for i := 0 to ReadTransaction.DatabaseCount – 1 do
    S := S + ReadTransaction.Databases[i].Name+#10#13;
  Application.MessageBox(PChar(S),'ReadTransaction',MB_OK);
end;
```

В компоненте `PersonData` зададим автоматический запуск и автоматическое подтверждение транзакций (в компоненте `CountryData` эти установки делать не будем с учетом целей наших исследований), сформируем необходимые операторы SQL. Свяжем компоненты между собой обычным образом, устанавливая значения соответствующих свойств.

Подтверждение транзакции имеет смысл, когда мы корректируем справочник стран (при корректировке людей подтверждение выполняется автоматически). Чтобы на сетке `DBGrid` отображались изменения в записях персонала, мы обращаемся к методу `FullRefresh` для этого набора данных.

Аналогично, если мы корректируем страны, то на сетке видны новые, измененные, записи. При отмене транзакции нужно заново считать страны, чтобы вернуться к старому виду записей.

В событии формы `OnShow` напишем операторы соединения с базами данных, запуска обеих транзакций и открытия обоих наборов данных (листинг 10.32). Те же действия, с точностью до наоборот, выполним в обработчике события закрытия формы (листинг 10.33).

Листинг 10.32. Обработка события `OnShow` формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin
  DatabaseRef.Connected := True;
  DatabaseData.Connected := True;
  ReadTransaction.StartTransaction;
  WriteTransaction.StartTransaction;
  CountryData.Open;
  PersonData.Open;
  DBGridCtr.SetFocus;
end;
```

Листинг 10.33. Обработка события `OnClose` формы

```
procedure TFormMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  CountryData.Close;
  PersonData.Close;
  DatabaseRef.Connected := False;
  DatabaseData.Connected := False;
end;
```

На рис. 10.10 представлен общий вид формы (там присутствуют еще четыре компонента `UpdateObject`, про которые мы пока не говорили).

В таком виде все замечательно работает. Обе транзакции запускаются для двух баз данных. При этом мы можем независимо просматривать и корректировать обе таблицы. На самом же деле нам нужно еще реализовать соответствующие связи между этими таблицами, расположенными в разных базах данных.

Воспользуемся возможностями компонента `TpFIBUpdateObject`, входящего в состав набора компонентов FIBPlus. Положим на форму два компонента `TpFIBUpdateObject`, присвоив им имена `UpdateObjectEdit` и `UpdateObjectDelete`. Их мы будем использовать для поддержания соответст-

вия таблицы PEOPLE таблице REFCTR. Их задачами, соответственно, являются обеспечение функциональности, указываемой для внешнего ключа, — ON UPDATE CASCADE и ON DELETE CASCADE.

Для UpdateObjectEdit установим значение свойства Database в DatabaseData, это означает, что оператор SQL этого компонента работает с базой данных, заданной в компоненте DatabaseData (это база данных Second.fdb), и его оператор SQL будет относиться к таблице, находящейся в этой базе данных.

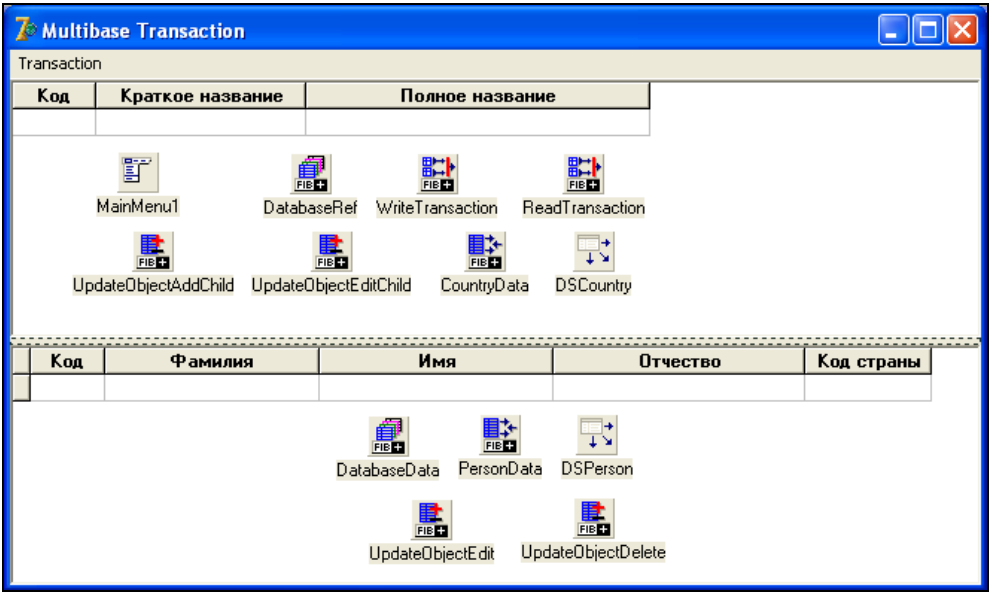


Рис. 10.10. Проект MultiBase

Свойство DataSet установим в CountryData (точнее, выберем из выпадающего списка). Это имя набора данных, на одно из событий которого будет реагировать наш компонент. Этот набор данных относится к таблице, хранящейся в другой базе данных. Конкретное событие мы выберем из выпадающего списка свойства KindUpdate: ukModify. Это означает, что будет выполняться оператор SQL нашего компонента UpdateObjectEdit, когда произойдет изменение данных в наборе данных CountryData. Зададим транзакцию для компонента: WriteTransaction. Это транзакция, в контексте которой будет выполняться оператор SQL компонента UpdateObjectEdit.

В свойстве SQL (это обычное многострочное диалоговое окно) создадим следующий оператор:

```
UPDATE PEOPLE SET CODCTR = :CODCTR
WHERE CODCTR = :OLD_CODCTR
```

Как это работает? Когда пользователь изменяет любую строку в таблице REFCTR и отправляет изменения в базу данных (щелкает мышью по любой другой строке таблицы), запускается на выполнение оператор SQL, заданный в нашем компоненте UpdateObjectEdit. Этот оператор заменяет все значения кода страны (CODCTR) в таблице PEOPLE на новое значение, получаемое из таблицы REFCTR (параметр :CODCTR). Изменению подвергаются только те строки, код страны в которых был равен старому значению кода страны из таблицы REFCTR (параметр :OLD_CODCTR).

Таким способом мы смоделировали предложение ON UPDATE CASCADE в описании внешнего ключа.

Похожим образом зададим свойства компонента UpdateObjectDelete. Только в свойстве KindUpdate мы выберем значение ukDelete, а в свойстве SQL запишем:

```
DELETE FROM PEOPLE
WHERE CODCTR = :OLD_CODCTR
```

При удалении какой-либо страны в таблице стран в первой базе данных будут удалены все люди, имеющие соответствующее значение кода страны в таблице людей во второй базе данных. Иными словами, таким способом мы смоделировали описание внешнего ключа, если бы такое было возможно, для таблицы людей из другой базы данных:

```
CONSTRAINT FK_PEOPLE
  FOREIGN KEY (CODCTR)
    REFERENCES [Work.fbd] REFCTR (CODCTR)
      ON UPDATE CASCADE
      ON DELETE CASCADE
```

Понятно, что предыдущий оператор условный. Такого в природе, точнее в InterBase и Firebird, быть не может. Однако именно его мы прямо сейчас и промоделировали.

Запустите программу на выполнение. Измените в любой строке справочника стран код страны и перейдите к любой другой строке, чтобы отправить изменения на сервер. При этом, поскольку транзакция не подтверждена, во второй таблице мы не можем видеть изменения (транзакция для чтения видит только подтвержденные изменения). Подтвердите транзакцию через элемент меню **Commit**. Тут же вы увидите, что соответствующие коды страны поменялись во всех нужных записях таблицы PEOPLE. Удалите одну из стран (клавиши <Ctrl>+<Del> и подтверждение удаления в диалоговом окне). Подтвердите транзакцию. Будут удалены все записи PERSON, ссылающиеся на данную страну.

Вы можете выполнять произвольное количество изменений и удалений записей стран. Соответствующие изменения в списке персонала появятся только лишь после подтверждения транзакции.

Таким способом мы промоделировали поведение внешнего ключа, правда, пока только в одну сторону.

Для полного счастья необходимо выполнить проверки на правильность добавления новых строк в таблицу `PEOPLE` и изменения значения в этой таблице "внешнего ключа" — кода страны. Нам нужно запретить добавление и изменение этого столбца, если в таблице стран нет записи с соответствующим значением первичного ключа. Поскольку внешний ключ может иметь значение `NULL`, мы не должны выполнять такую проверку для пустого значения этого столбца.

Для реализации этого поведения создадим исключение с именем `NO_COUNTRY` и текстом `В справочнике стран отсутствует страна с указанным кодом` (это исключение присутствует в скрипте создания триггеров) и хранимую процедуру для выполнения соответствующей проверки в базе данных (процедура присутствует в скрипте, создающем хранимые процедуры). Текст хранимой процедуры представлен в листинге 10.34.

Листинг 10.34. Хранимая процедура проверки существования страны

```
/* Проверка присутствия в базе данных страны */
CREATE PROCEDURE TEST_COUNTRY (CODCTR CHAR(3))
AS
DECLARE VARIABLE COUNTRY_NUM INTEGER;
BEGIN
  IF (CODCTR IS NOT NULL) THEN
  BEGIN
    IF (NOT EXISTS(SELECT CODCTR
          FROM REFCTR WHERE CODCTR = :CODCTR)) THEN
      EXCEPTION NO_COUNTRY;
  END
END ^
```

Положим на форму еще два компонента `UpdateObject`: `UpdateObjectAddChild` и `UpdateObjectEditChild`. Свойства `UpdateObjectEditChild` показаны на рис. 10.11.

Его свойство SQL содержит обращение к хранимой процедуре:

```
EXECUTE PROCEDURE TEST_COUNTRY (:CODCTR)
```

Компонент `UpdateObjectAddChild` имеет все те же свойства за исключением значения `KindUpdate`, для которого выбрано `ukInsert`. Это значит, что его

оператор SQL будет выполняться при добавлении новой записи в таблицу людей.

Для набора данных `PersonData` напишем обработчик события `BeforePost` (листинг 10.35).

Листинг 10.35. Обработчик события `BeforePost`

```
procedure TFormMain.PersonDataBeforePost(DataSet: TDataSet);
begin
  UpdateObjectAddChild.ParamByName('CODCTR').AsString :=
    PersonData.FieldByName('COUNTRY').AsString;
  UpdateObjectEditChild.ParamByName('CODCTR').AsString :=
    PersonData.FieldByName('COUNTRY').AsString;
end;
```

Здесь мы формируем значения параметров `CODCTR` компонентов `UpdateObject`.

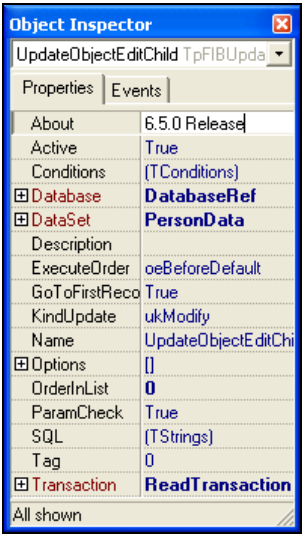


Рис. 10.11. Свойства компонента `UpdateObjectEditChild`

Если при добавлении новой записи в таблицу `PEOPLE` или изменении значения существующей записи значение кода страны не будет найдено в справочнике стран (в другой базе данных), то выдается исключение с текстом `В справочнике стран отсутствует страна с указанным кодом`.

Запустите проект на выполнение, измените в какой-нибудь записи таблицы `PEOPLE` код страны на несуществующее значение. Вы получите наше исклю-

чение. Если же вы просто удалите значение кода страны (код примет значение `NULL`), то ничего не произойдет. Что и требовалось.

С помощью рассмотренных средств мы очень хорошо реализовали возможности систем управления базами данных по обеспечению целостности данных в случае использования внешних ключей для таблиц, присутствующих в разных базах данных.

10.3. Программа просмотра и печати справочника стран

Сейчас мы расширим возможности программы работы со справочником стран. За основу возьмем созданный ранее проект `InsertUpdate` и добавим средства предварительного просмотра и печати отчетов. Для этих целей мы используем программный продукт FastReport. Как установить для вашей среды Delphi эти компоненты, описано в *приложении 1*.

Мы создадим две версии программы — с использованием компонентов FIBPlus и компонентов IBX.

10.3.1. Использование компонентов FIBPlus

Пожалуй, наиболее простой способ получить начальную версию проекта — это переписать в новый каталог проект `InsertUpdate`. Затем нужно открыть его в среде Delphi и сохранить в том же каталоге с именем `Country`.

Оставим существующую функциональность и внесем необходимые добавления для отображения отчетов.

Дважды щелкните мышью по компоненту `ImageList` и в диалоговом окне добавьте (щелчок мышью по кнопке **Add**) две иконки с именами `PreviewM` и `PrintM`. Все иконки, используемые в наших программах для элементов меню и кнопок инструментальной панели, находятся в каталоге Icons.

Диалоговое окно **ImageList** после добавления будет выглядеть следующим образом — рис. 10.12.

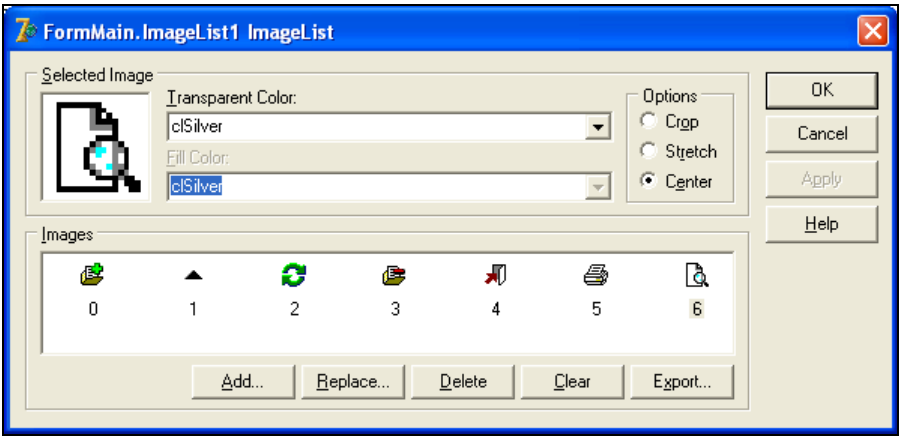


Рис. 10.12. Диалоговое окно **ImageList**

Раскройте редактор главного меню, дважды щелкнув по компоненту `MainMenu`. Добавьте два элемента: **Предварительный просмотр** (имя `MPreview`, клавиши быстрого доступа <Ctrl>+<E>) и **Печать** (имя `MPrint`, <Ctrl>+<P>) (рис. 10.13). Установите для них соответствующие иконки, выбрав из выпадающего списка **ImageIndex**.

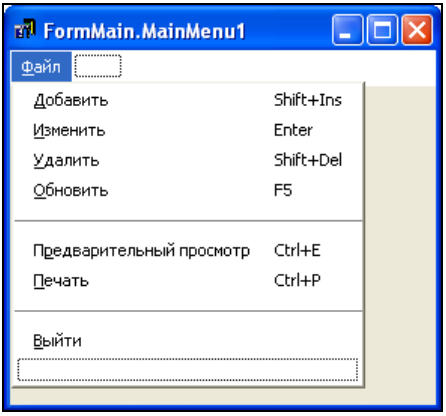


Рис. 10.13. Главное меню программы

Добавьте на инструментальную панель две кнопки `SpeedButton` для предварительного просмотра (имя `BPreview`) и печати (`BPrint`) отчета. Установите соответствующие подсказки (свойство `Hint`) и поместите иконки, используя свойство `Glyph`.

С вкладки **FastReport 4.0** положите на форму компоненты `frxDBDataset1` и `frxReport`.

Теперь форма должна выглядеть так, как показано на рис. 10.14.

У компонента `frxDBDataset1` установите свойству `DataSource` значение `DataSource1` (выберите из выпадающего списка).

> **ВНИМАНИЕ!**
> Все описываемые здесь действия по формированию отчета необходимо выполнять именно в указанном порядке. В случае отступлений от порядка описываемых действий не исключены некоторые неточности в получаемом отчете.

Займемся теперь непосредственно отчетом. Дважды щелкнем мышью по компоненту `frxReport`. Появится окно создания отчета (рис. 10.15).

В верхней части окна располагается главное меню, кнопки инструментальной панели, дублирующие отдельные элементы главного меню, и кнопки редактирования текста. В центре находится рабочее поле, на котором и формируется весь отчет.

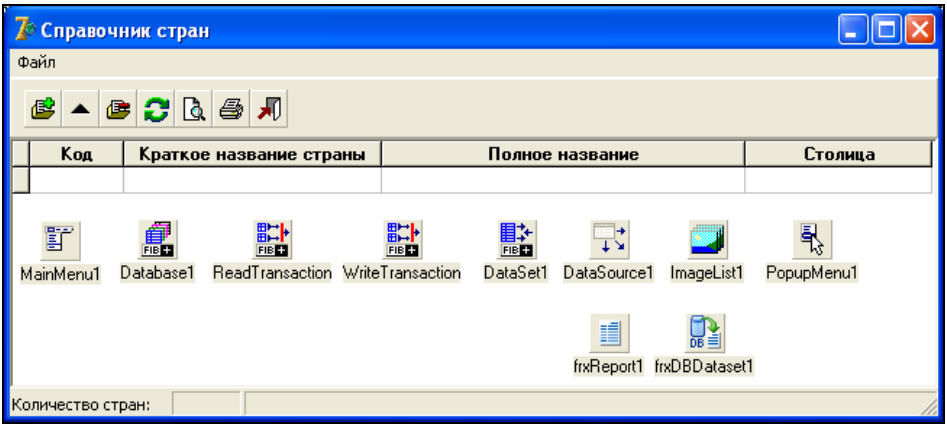


Рис. 10.14. Программа просмотра и печати справочника стран

Слева виден Инспектор объектов с двумя обычными вкладками — **Properties** (Свойства) и **Events** (События).

Вертикально по левому краю располагается инструментальная панель, содержащая объекты, добавляемые к отчету.

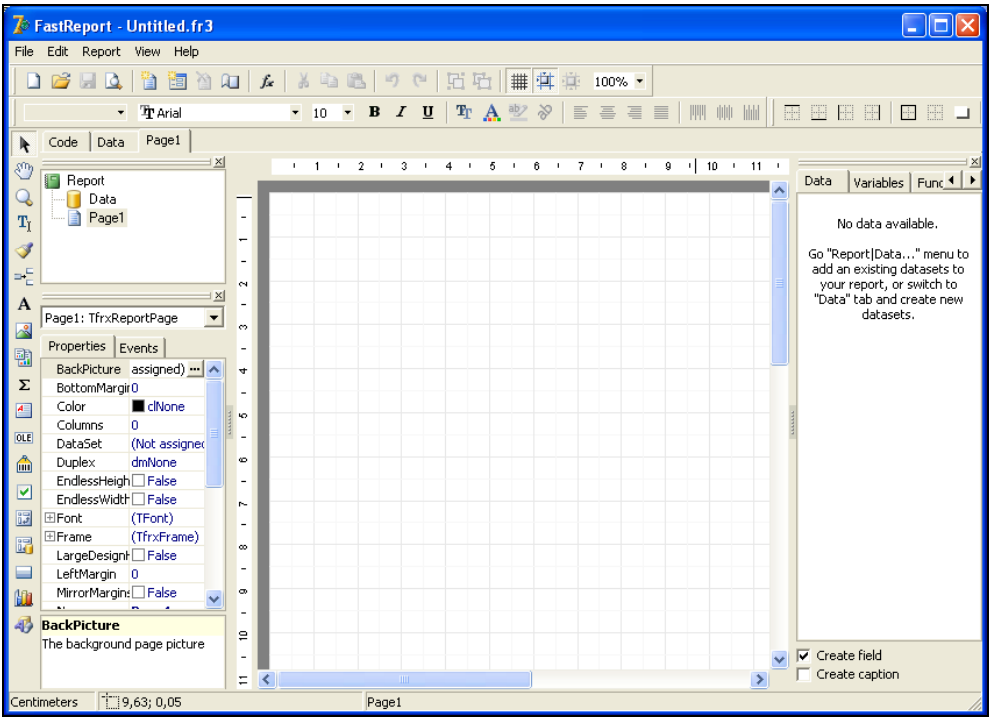


Рис. 10.15. Окно создания отчета

Создадим новый отчет. Выбираем в меню **File | New Report** или щелкаем по кнопке **New Report** на инструментальной панели. В рабочем поле появляется три полосы:

- заголовок отчета (`ReportTitle`). Эта полоса будет появляться только один раз в самом начале отчета;
- данные первого уровня (`MasterData`). Здесь будут помещаться основные данные;
- подвал страницы, или нижний колонтитул (`PageFooter`). Эта полоса будет печататься в конце каждой страницы. Справа этой полосы виден текст **[Page#]**. В это место будут подставляться номера страниц.

Добавим в отчет заголовок страницы, который будет появляться в начале каждой страницы. Слева на вертикальной панели инструментов находим кнопку **Insert Band** (Вставить полосу), щелкаем по ней мышью и в появившемся меню выбираем элемент `PageHeader` (Заголовок страницы, или верхний колонтитул). В рабочем поле появляется новая полоса `PageHeader`.

Все полосы подготовлены. Начнем заполнять их нужными данными.

Щелкните на левой панели инструментов по кнопке **Text object**, затем щелкните мышью по полосе заголовка отчета. Сразу появится окно редактирования текста (рис. 10.16).

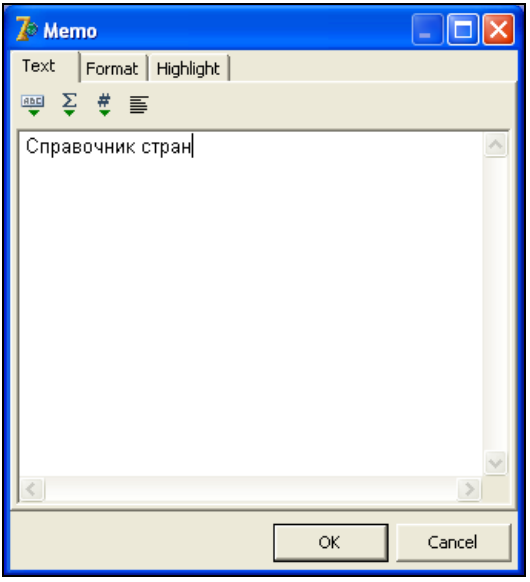


Рис. 10.16. Окно редактирования текста

Введите в нем `Справочник стран` и щелкните по кнопке **OK**.

Сделайте текст полужирным, размер шрифта установите 14. Для этого можно использовать кнопки форматирования текста в верхней части редактора отчета или обычным для Delphi образом установить характеристики шрифта (свойство `Font`) в Инспекторе объектов.

В Инспекторе объектов задайте выравнивание по левому краю: для свойства `Align` из выпадающего списка выберите значение `baLeft`.

На заголовке страницы `PageHeader` разместите четыре надписи: "Код", "Краткое название", "Полное название" и "Столица". На эту же полосу положите две горизонтальные линии на верхнюю часть полосы и на нижнюю. Это позволит несколько украсить заголовок и отделить его от переменной части отчета.

Для того чтобы в отчете можно было использовать набор данных, нужно в главном меню **Report** выбрать элемент **Data**. Появится окно выбора набора данных (рис. 10.17).

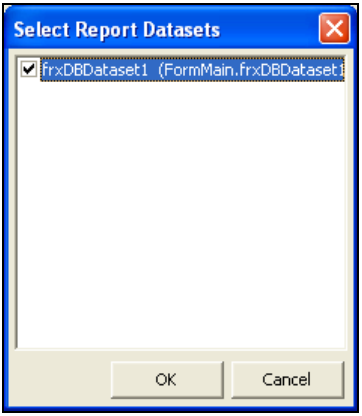


Рис. 10.17. Выбор набора данных отчета

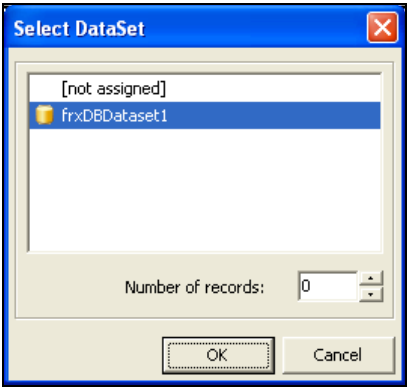


Рис. 10.18. Выбор набора данных для полосы

У нас в программе существует только один доступный набор данных. Отметим флажок слева от имени набора данных и щелкнем по кнопке **OK**.

На полосе данных (`MasterData`) щелкнем правой кнопкой мыши и в контекстном меню выберем **Edit** (Редактировать). Появится окно выбора набора данных, связанного с этой полосой (рис. 10.18).

Выберем набор данных и щелкнем по кнопке **OK**.

Установим для этой полосы в `True` значение свойства `Stretched`. Это приведет к тому, что вертикальный размер полосы будет динамически подгоняться к высоте помещаемых на нее данных. Дело в том, что полное название страны слишком длинное. Мы планируем сделать так, чтобы длинные тексты выводились в несколько строк. В нашей базе данных название только одной страны займет не одну, а две строки.

Поместим на эту полосу четыре компонента **Text object** — код, краткое название, полное название и столица. Чтобы связать эти объекты с соответствующими столбцами нашего набора данных, необходимо, выделив компонент, установить нужное значение свойствам `DataSet` (набор данных — выбираем из выпадающего списка) и `DataField` (поле набора данных — также выбираем из списка).

Остальные значения свойств можно оставить в том виде, как они были заданы по умолчанию, кроме поля для размещения полного названия страны. Горизонтальный размер этого поля нужно сделать таким, чтобы он соответствовал требуемой ширине. Для свойства `StretchMode` из выпадающего списка нужно выбрать `smActualHeight`. Это приведет к тому, что в случае, когда текст в этом поле не будет умещаться в отведенные ему размеры, следующие

слова будут переноситься на вторую строку. При этом и сама полоса будет увеличиваться по высоте.

Следует еще в полосе подвала страницы (`PageFooter`) в верхней ее части разместить горизонтальную линию, для большей красоты.

Проектируемая форма будет иметь такой вид — рис. 10.19.

Пожалуй, все. Закройте окно редактора отчета.

Теперь нужно написать обработчики событий вызова предварительного просмотра и печати отчета.

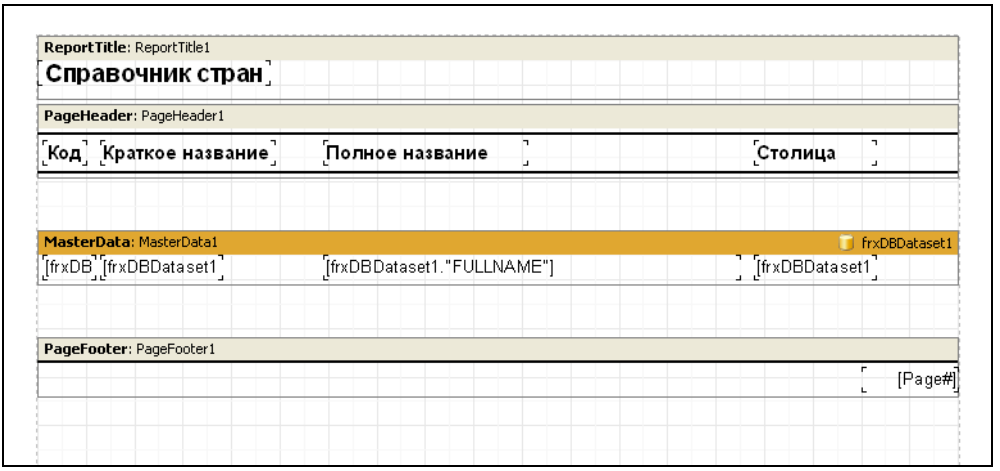


Рис. 10.19. Спроектированный отчет

Раскройте компонент главного меню, дважды щелкнув по нему мышью, и напишите следующий обработчик события выбора элемента меню **Предварительный просмотр** (листинг 10.36).

Листинг 10.36. Выполнение предварительного просмотра отчета

```
procedure TFormMain.MPreviewClick(Sender: TObject);
var Mark: TBookmark;
begin
  Mark := DataSet1.GetBookmark;
  DataSet1.DisableControls;
  frxReport1.ShowReport;
  DataSet1.GotoBookmark(Mark);
  DataSet1.FreeBookmark(Mark);
  DataSet1.EnableControls();
end;
```

Для вызова окна предварительного просмотра отчета используется метод `ShowReport` компонента отчета. В результате выполнения этого метода будет построен и отображен в соответствующем окне наш отчет.

Однако перед началом просмотра мы сохраняем текущую позицию набора данных в локальной переменной типа `TBookmark` (метод набора данных `GetBookmark`). Это называется закладкой по аналогии с закладкой в книге. После выполнения предварительного просмотра мы возвращаемся на ту же запись, которая была перед выполнением просмотра (метод `GotoBookmark`), и освобождаем созданную закладку (метод `FreeBookmark`).

Если бы мы этого не выполняли, то после предварительного просмотра печати у нас всегда текущей записью была бы последняя запись в наборе данных.

Кроме этого, мы перед просмотром отключаем прокрутку на сетке отображения набора данных, используя метод набора данных `DisableControls`. Это позволит нам в процессе создания и отображения отчета избежать перемещения по сетке текущей позиции, что для больших отчетов может вызвать и немалую задержку по времени. После выполнения просмотра мы опять делаем сетку доступной, вызывая метод набора данных `EnableControls`.

Не забудьте связать с этим обработчиком щелчок по кнопке предварительного просмотра на инструментальной панели.

Похожим образом выполняется и печать отчета. Пишем обработчик события выбора в главном меню элемента **Печать** и связываем с ним щелчок по кнопке **Печать** — листинг 10.37.

Листинг 10.37. Выполнение печати отчета

```
procedure TFormMain.MPrintClick(Sender: TObject);
var Mark: TBookmark;
begin
  Mark := DataSet1.GetBookmark;
  DataSet1.DisableControls;
  frxReport1.PrepareReport;
  frxReport1.Print;
  DataSet1.GotoBookmark(Mark);
  DataSet1.FreeBookmark(Mark);
  DataSet1.EnableControls();
end;
```

Прежде чем выполнять печать отчета, необходимо его сформировать. Это выполняет метод компонента отчета `PrepareReport`.

Для печати отчета используется метод `Print` компонента отчета. Появится окно выбора принтера, где можно задать характеристики принтера, объем и порядок печати страниц отчета.

10.3.2. Использование компонентов IBX

Создание программы с использованием компонентов IBX выполняется точно таким же образом. На форме только нужно разместить соответствующие компоненты работы с базой данных с вкладки **InterBase**.

> ***Замечание***
>
> Здесь мы с вами рассмотрели очень и очень простой отчет. Компоненты `FastReport` позволяют создавать весьма сложные и красивые отчеты. Существует документация, в том числе и на русском языке, где подробно рассказывается, как создаются отчеты, рассматриваются многочисленные примеры.

10.4. Программа, включающая средство редактирования полей *BLOB*

Последняя программа, которую мы рассмотрим в этой книге, содержит редактор, позволяющий выполнять редактирование произвольных текстов, помещаемых в поля `BLOB`. Его по моей просьбе написал Сергей Ткаченко, автор компонентов RichView. В своем редакторе он, естественно, использует именно эти компоненты.

За основу мы взяли программу работы со справочником стран. При добавлении новой записи вызывается форма добавления, которая и содержит тот самый редактор.

Вид формы показан на рис. 10.20.

На панели инструментов, в главном и контекстном меню содержатся элементы, позволяющие вызывать разнообразные функции редактирования текста в поле `rve`, имеющем тип `TRichViewEdit`.

Как вы видите, редактор имеет множество функций редактирования.

Напомню, что компоненты работы с полями в формате RTF, поставляемые с Delphi и C++Builder, позволяют как угодно редактировать тексты, но не дают возможности работать с графикой. Компоненты RichView предоставляют нам и такую возможность.

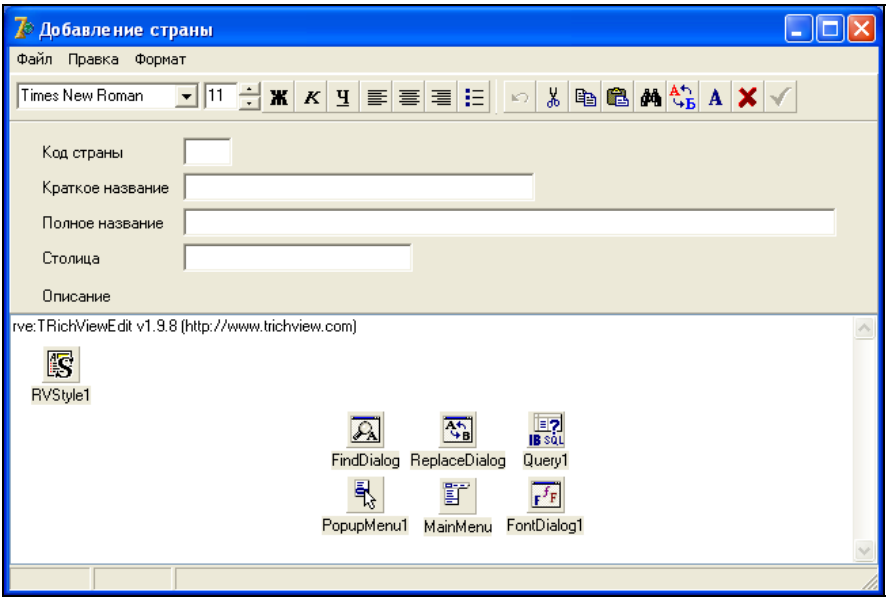


Рис. 10.20. Форма добавления новой записи в справочник стран

Описывать эту программу я не стану. Она достаточно сложная, описание займет много времени и места в книге. Я предлагаю вам самим разобраться с текстом. Сергей в код добавил необходимые комментарии. Вы прекрасно во всем разберетесь.

Что там за перевалом?

По-моему мы с вами молодцы! Сколько пройдено, сколько перевалов, сколько ледников. Сейчас мы имеем полное право считать себя классными специалистами в области реляционных баз данных. Мы с таким блеском написали великое множество полезных программ. Их мы с вами еще будем использовать на благо отечества и вообще мировой цивилизации.

Напоследок мне осталось сказать вам несколько заключительных слов.

ПРИЛОЖЕНИЕ 1

Установка необходимых программ и компонентов

В этом приложении содержится описание установки всех программ и всех компонентов, которые используются при написании и использовании программ в данной книге, за исключением, разумеется, Delphi. Некоторые программные продукты устанавливаются легко и не требуют от нас напряжения душевных сил. Другие же, наоборот, в процессе установки или при попытках дальнейшей эксплуатации могут здорово помотать нервы. И это не означает, что такие программные продукты плохо сделаны. Многое зависит от особенностей и сложностей реализации той среды, куда устанавливаются эти программы — я имею в виду именно Delphi, да и саму операционную систему тоже.

По этой причине в данном приложении я очень (а временами слишком) подробно рассказываю про инсталляцию каждого используемого в книге программного продукта, компонента. Если вас это раздражает, пробегитесь глазами по тексту и выполните привычную для вас работу по инсталляции.

П1.1. Установка Firebird 2.0

Инсталляционный файл для Firebird 2.0 находится на прилагаемом к книге компакт-диске в каталоге Firebird 2.0.

Если у вас уже запущен сервер Firebird или InterBase, то следует его остановить. Для этого нужно вызвать Диспетчер задач (в Windows XP и других вариантах NT щелкнуть правой кнопкой мыши по Панели задач и в контекстном меню выбрать Диспетчер задач. Не поленитесь — в других вариантах операционных систем решите сами, как увидеть Диспетчер задач, обычно для этого нужно нажать три любимые клавиши: <Ctrl>+<Alt>+<Delete>). Далее следует перейти на вкладку **Процессы**, выбрать программу fbguard.exe (программа защиты, или "опекун") и щелкнуть по кнопке **Завершить процесс**.

Потом аналогичную процедуру выполнить для процесса fbserver.exe (это сам сервер базы данных). Завершение процессов нужно выполнять именно в указанном порядке, иначе, если вы остановите сервер базы данных, он тут же будет вновь запущен программой Guardian.

После этого следует деинсталлировать существующую версию Firebird и удалить из каталога установки Firebird все оставшиеся файлы. В некоторых случаях придется из каталога system32 вручную удалить файл gds32.dll.

Запустите программу инсталляции Firebird.

Вообще говоря, лучшим поведением при инсталляции этого сервера базы данных является соглашение с каждым предлагаемым вариантом. Тем не менее рассмотрим весь процесс.

В начале инсталляции Firebird появится окно выбора языка (рис. П1.1).

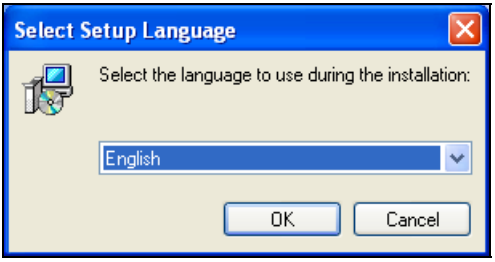


Рис. П1.1. Окно выбора языка

Разумеется, из доступных языков мы выберем наш "родной", английский. Далее появляется окно приветствия (рис. П1.2).

Эта птичка в программах и документах по Firebird всегда производила на меня самое приятное впечатление.

Щелкните по кнопке **Next** (Далее). Появится лицензионное соглашение (рис. П1.3).

Здесь следует ознакомиться с лицензионным соглашением. Надо полагать, никаких возражений оно у вас не вызовет. По этой причине отметьте переключатель **I accept the agreement** (Я принимаю это соглашение) и щелкните по кнопке **Next**. Появится информационное окно (рис. П1.4).

Сейчас можете не читать информацию по инсталляции Firebird. Доверьтесь пока мне. Я вам все расскажу, что надо сделать. Щелкните по кнопке **Next**. Появится окно выбора каталога инсталляции (рис. П1.5).

Здесь можно изменить путь к каталогу инсталляции. Только нужно ли это? Щелкните по кнопке **Next** (рис. П1.6).

Рис. П1.2. Окно приветствия Firebird

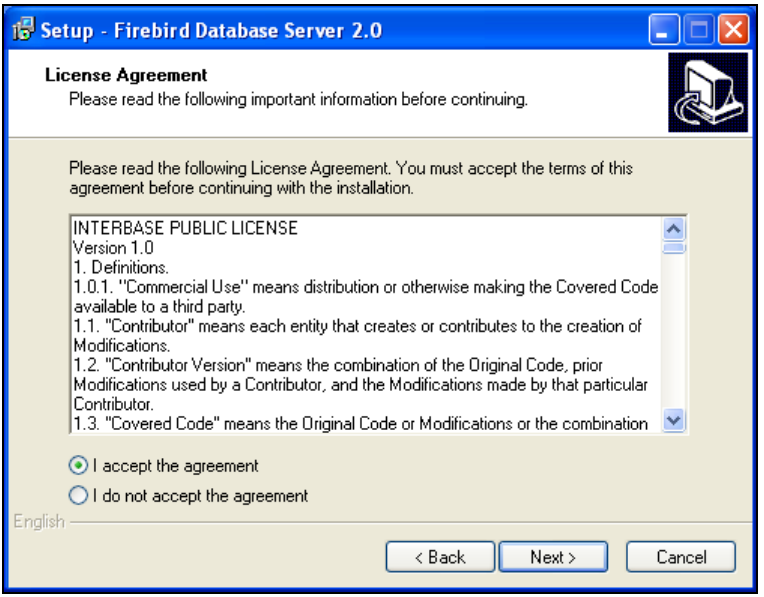


Рис. П1.3. Лицензионное соглашение

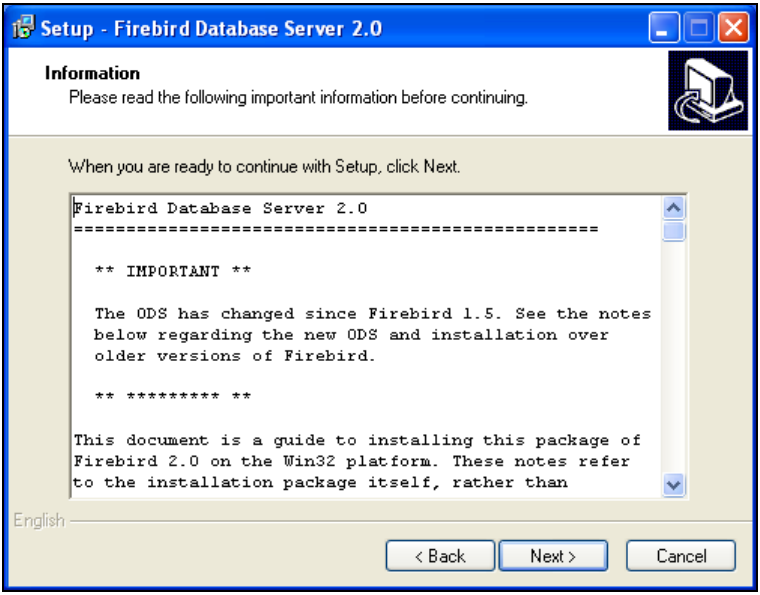


Рис. П1.4. Информация по инсталляции

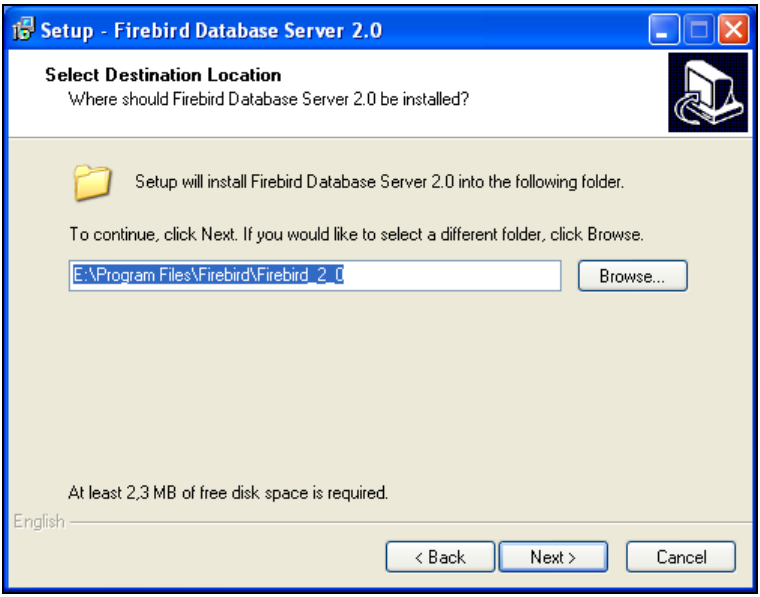


Рис. П1.5. Окно выбора каталога инсталляции

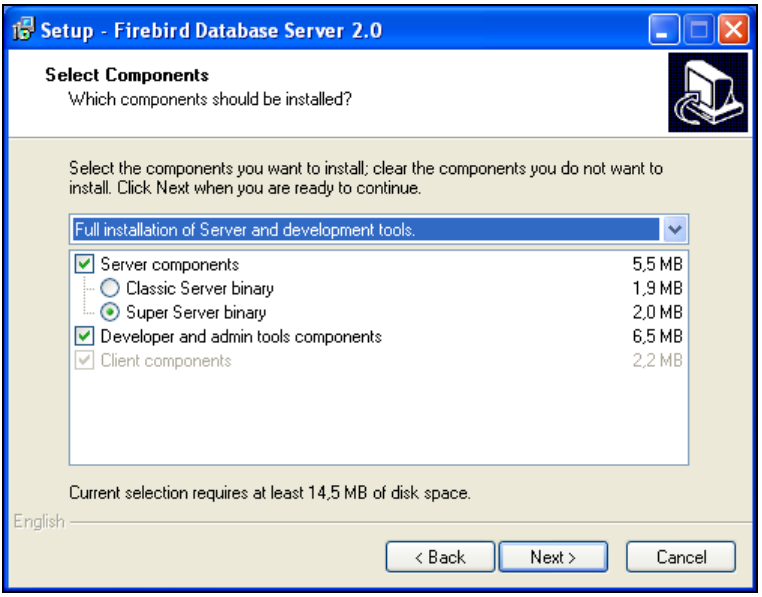


Рис. П1.6. Выбор устанавливаемых компонентов

> ***Замечание***
> На рис. П1.5 вы видите, что у меня Windows XP Home Edition установлен на диске E:. На C: стоит Windows 98. Это мне нужно для проверок работы программных компонентов в различных средах. (Про установленный у меня на компьютере еще и Linux я ничего не скажу.)

Это уже серьезнее. Опять же, предлагаемый вариант инсталляции компонентов является наиболее разумным, однако у вас могут быть свои соображения.

Вам предлагается установить Суперсервер (Super Server). Альтернативный вариант — установка Классического сервера (Classic Server). Соглашайтесь на Суперсервер. Не вдаваясь в подробности, скажу, что Классический сервер менее удобен и вряд ли вам понадобится в будущих разработках, при появлении новых версий сервера базы данных.

Далее указано, что будет устанавливаться сервер (а не только клиентская часть программы) — отмечен флажок **Server components**. Принимаем. Если же вы будете устанавливать систему на "чисто" клиентский компьютер, на котором никогда не будут создаваться свои базы данных, а будет лишь обращение к базам данных, находящимся на сервере, то на него можно поставить только клиентское программное обеспечение. Это позволит сэкономить несколько байтов внешней памяти, точнее 1.7 Мбайт.

Следом идут инструменты разработчика и администратора — **Developer and admin tools components**. Это утилиты командной строки, которые мы с вами

активно использовали. Конечно, мы согласны. Если же вам нужно экономить место во внешней памяти, и ваш компьютер является только клиентским, можете отказаться от установки этих инструментов. При этом сможете сэкономить чуть ли не 5 Мбайт внешней памяти.

Компоненты клиента будут выбраны, независимо от нашего с вами желания.

Щелкните по кнопке **Next**. Появится следующее окно (рис. П1.7).

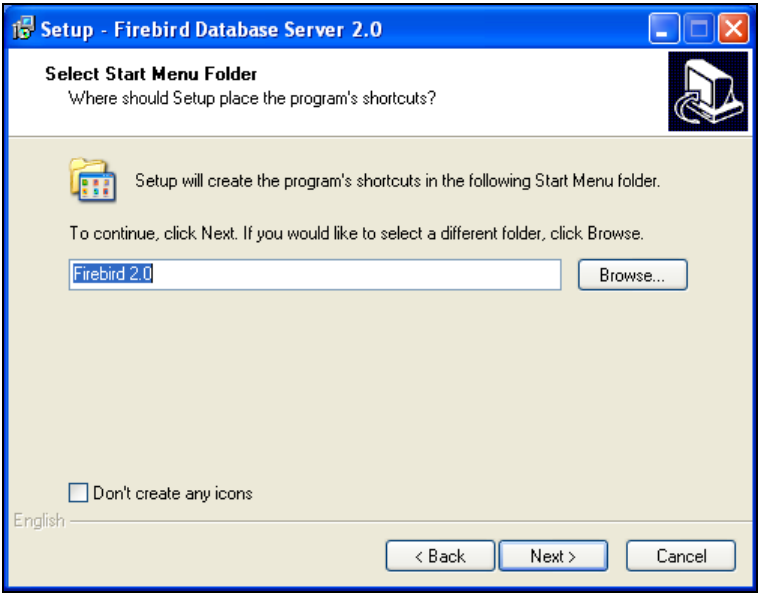


Рис. П1.7. Выбор папки

Предлагается выбрать имя папки. Ежели очень хочется изменить имя, сделаем это. Но, скорее всего, просто щелкаем по кнопке **Next** (рис. П1.8).

Следует сохранить отметку выбора Guardian для управления сервером (**Use the Guardian to control the server**). Если сервер будет по каким-либо неправильным причинам остановлен, например, из-за ошибок в ваших программах, Guardian тут же опять его запустит.

Следующий запрос на вид запуска сервера Firebird: запускать ли его как сервис или как приложение — переключатель **Run as an Application** и **Run as a Service**. В Windows технологии NT, к которой относится и мой Windows XP, желательно запустить его как сервис (почему — честно скажу, не знаю).

Далее спрашивается, запускать ли сервер автоматически при каждой первоначальной загрузке системы (**Start Firebird automatically everytime you boot up**). Если вы будете работать только с Firebird, то лучше согласиться с этим предложением. Firebird будет автоматически запускаться при первоначальной

загрузке Windows на вашем компьютере. У себя я отменил автоматический запуск, потому что на свой компьютер ставлю еще и InterBase 2007 — с исследовательскими целями и по той причине, что пишу эту книгу.

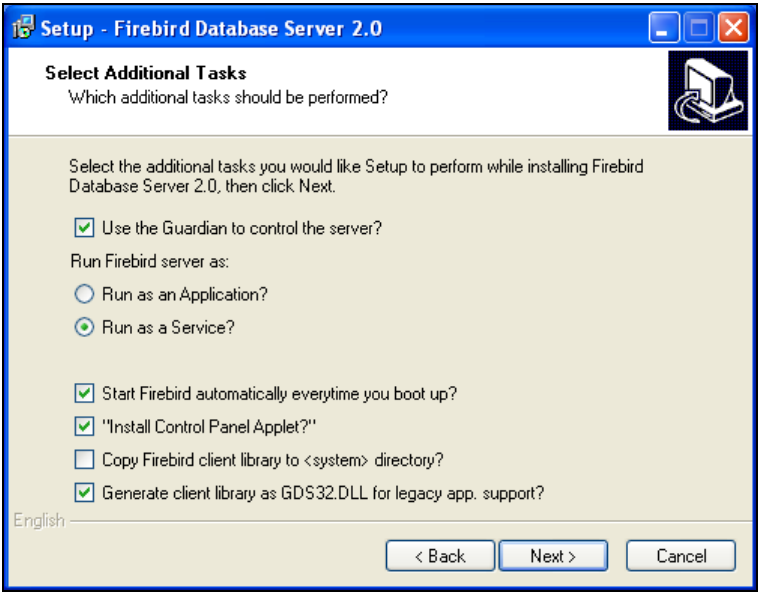


Рис. П1.8. Выбор дополнительных возможностей

Копировать клиентскую библиотеку в системный каталог (**Copy Firebird client library to <system> directory**) вряд ли нужно, поэтому не отмечаем этот флажок.

Остальные флажки оставляем отмеченными, не слишком задумываясь об их предназначении.

В следующем окне (рис. П1.9) показаны результаты вашего выбора. Если что-то не так, щелкните по кнопке **Back** (Назад) и измените предыдущие установки. После щелчка по кнопке **Install** вы увидите окно, иллюстрирующее процесс инсталляции (рис. П1.10).

Далее появится информационное окно (рис. П1.11), прочтите текст, не повредит. Последним на экран будет выведено окно завершения (рис. П1.12).

Оставьте отмеченным флажок запуска сервера Firebird (**Start Firebird Service now**). В результате Firebird будет запущен на выполнение. Щелкните по кнопке **Finish**.

Инсталляция Firebird завершена, сервер базы данных запущен на выполнение. (Кроме этого на компьютер будут установлены три документа в формате PDF.)

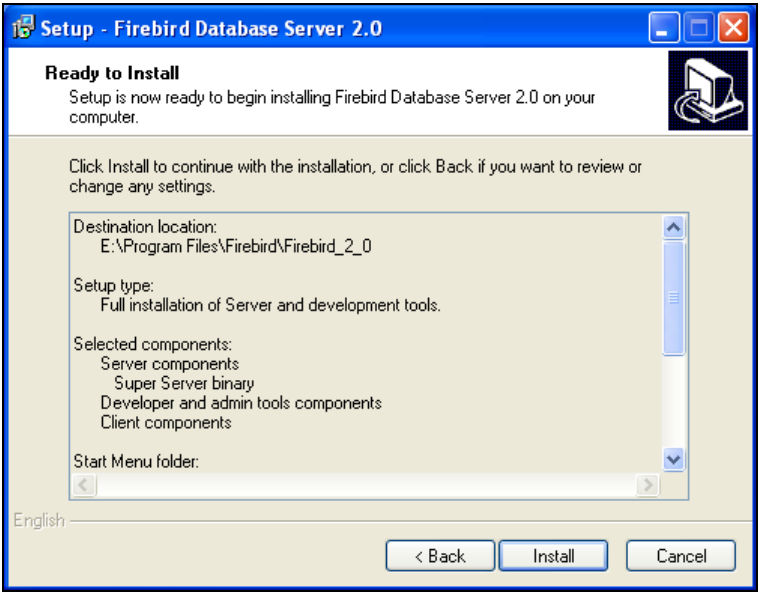


Рис. П1.9. Список выбранных возможностей

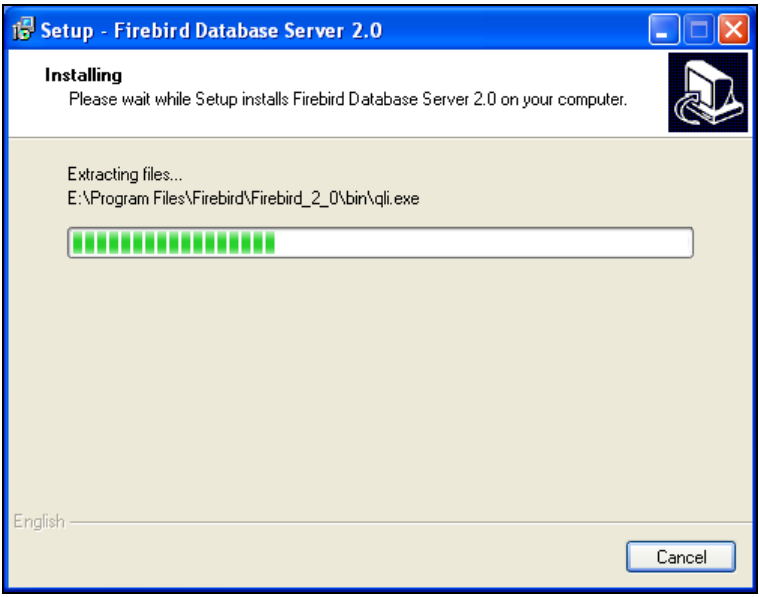


Рис. П1.10. Отображение процесса установки

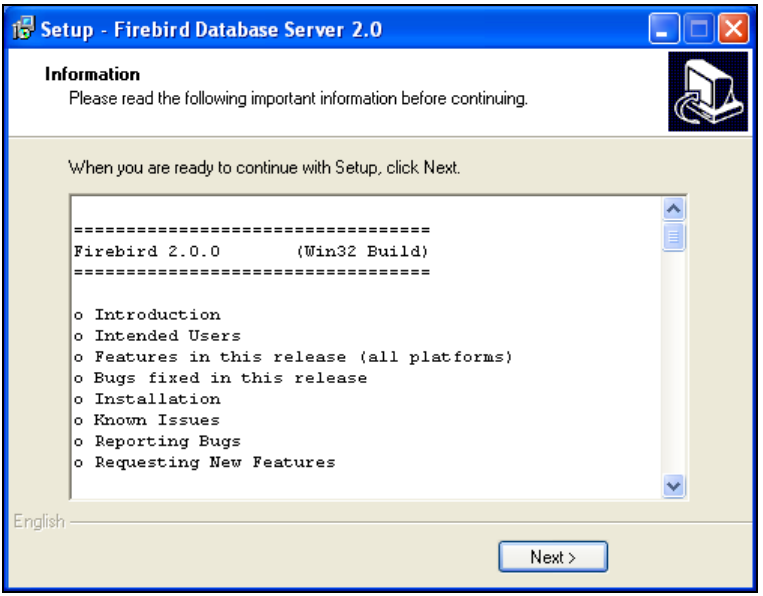


Рис. П1.11. Полезная информация

Рис. П1.12. Окно завершения

П1.2. Установка InterBase 2007 Developer Edition

Установка InterBase, находящегося на прилагаемом к книге компакт-диске, потребует больше времени и усилий, потому что это все-таки коммерческий продукт, и после его установки потребуется выполнить регистрацию через Интернет.

Запустите на выполнение программу install_windows.exe. Появится первое окно установки (рис. П1.13).

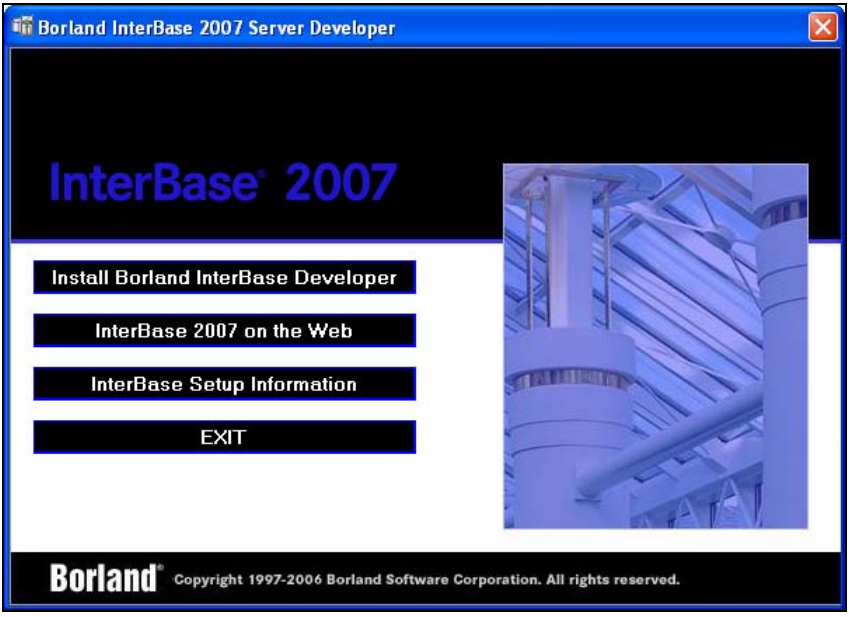


Рис. П1.13. Начальное окно установки InterBase

Нужно щелкнуть мышью по первой строке **Install Borland InterBase Developer**. После этого появится следующее окно (рис. П1.14).

Здесь выбираем только один вариант — **InterBase 2007 Server** и щелкаем по кнопке **Install**. Появится информационное окно (рис. П1.15).

Следующее окно — лицензионное соглашение (рис. П1.16). Прочтите, и если вы по-настоящему согласны выполнять это соглашение, щелкните по кнопке **Yes**.

После этого появится окно, запрашивающее вас об использовании множества экземпляров InterBase в вашей системе (рис. П1.17). В целях изучения

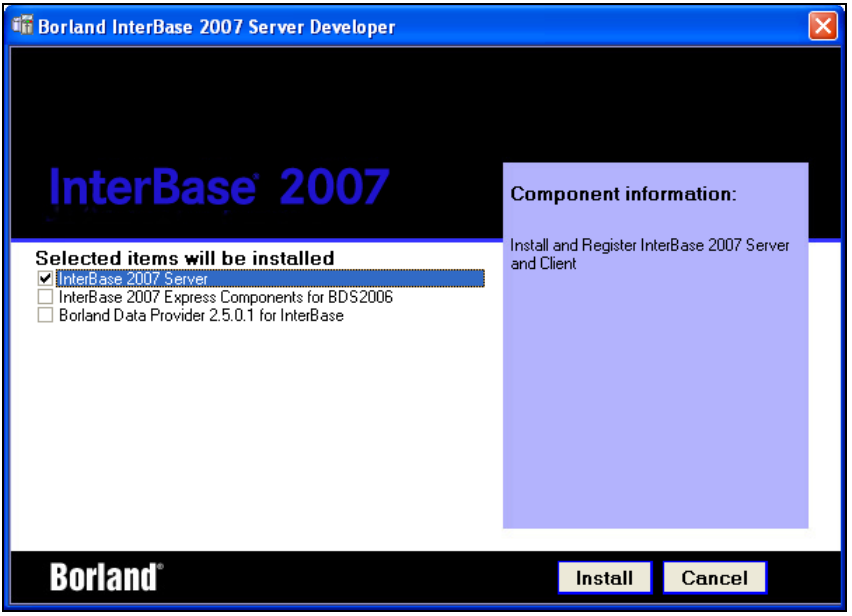


Рис. П1.14. Окно выбора устанавливаемых систем

Рис. П1.15. Информационное окно

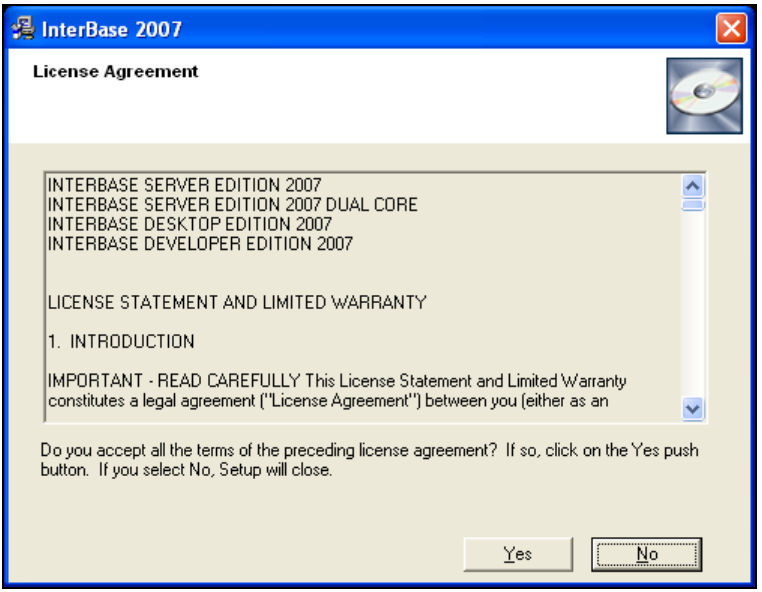


Рис. П1.16. Лицензионное соглашение

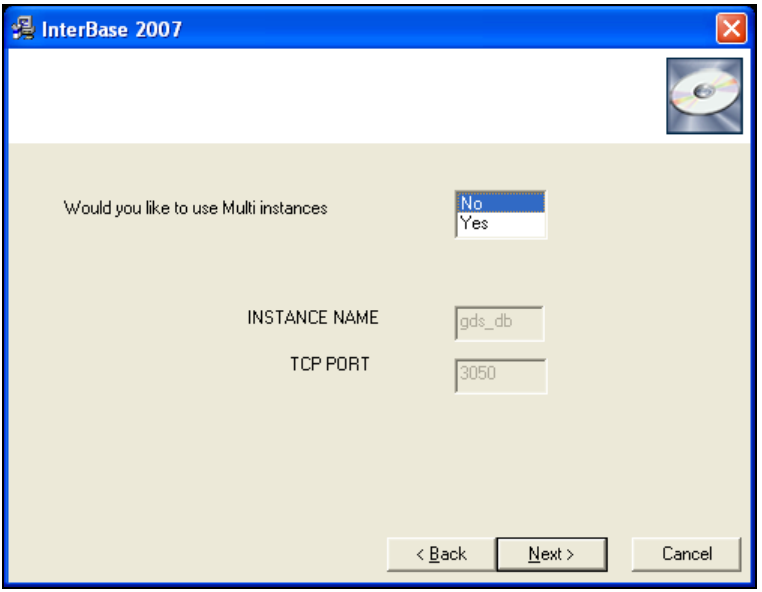


Рис. П1.17. Каталог размещения

системы нам пока не нужно много вариантов СУБД, поэтому выбираем предлагаемый вариант **No** и щелкаем по кнопке **Next**.

Далее нам предлагается выбор компонентов (рис. П1.18). Отмечаем **Server and Client** (серверное программное обеспечение и клиентское) и **Documentation** (помещение на компьютер документации).

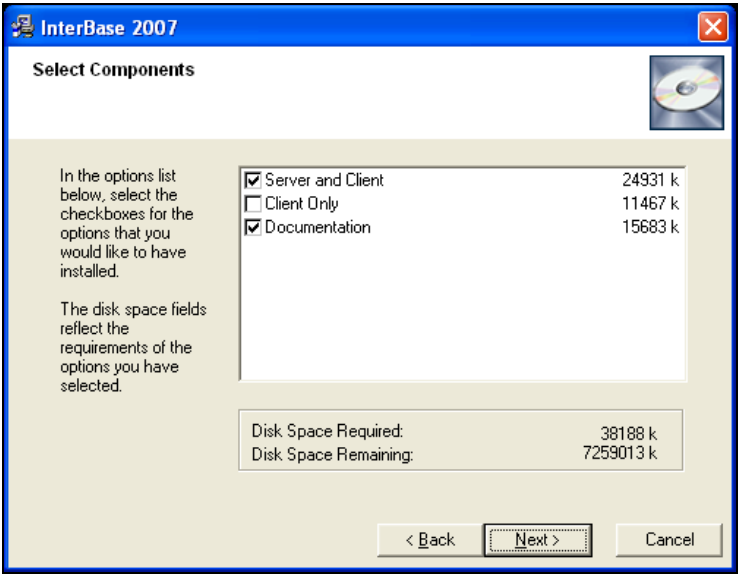


Рис. П1.18. Выбор устанавливаемых компонентов

Следующее окно — выбор каталога размещения (рис. П1.19).

Здесь вы можете изменить каталог, щелкнув по кнопке **Browse**. Следующее окно — последнее перед началом инсталляции (рис. П1.20).

Щелкаем по кнопке **Next**. Появляется окно, отображающее процесс инсталляции (рис. П1.21).

После успешно выполненной инсталляции появляется последнее окно (рис. П1.22).

В любом случае после инсталляции нужно перезагрузить компьютер.

Чтобы установленную программу можно было использовать, необходимо ее зарегистрировать. Для этого в Internet Explorer (или в любой другой подходящей программе) нужно войти на страничку **http://reg.borland.com**.

На страничке найдите **Download** и щелкните по этой гиперссылке мышью.

На следующей странице внизу найдите гиперссылку **InterBase** и щелкните по ней мышью. Появятся две таблицы. В первой таблице в первой строке щелкните по полю **InterBase 2007 Developer Edition**.

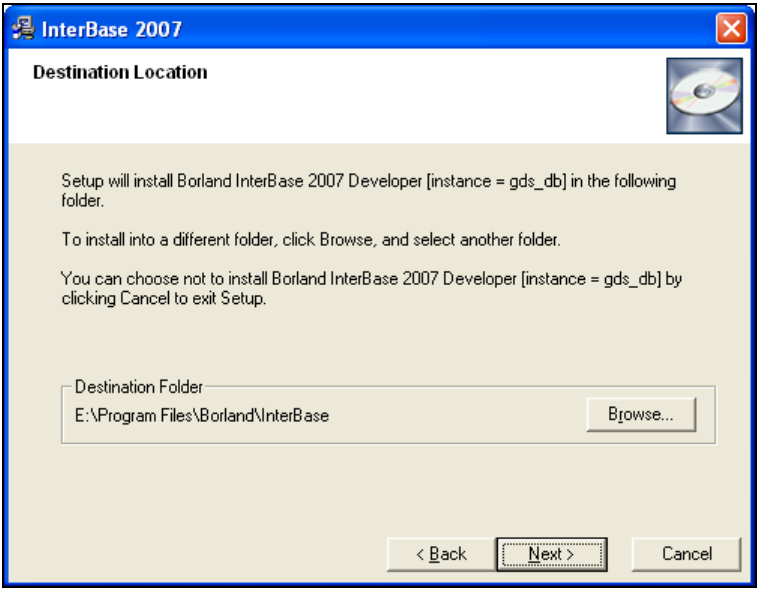


Рис. П1.19. Каталог размещения

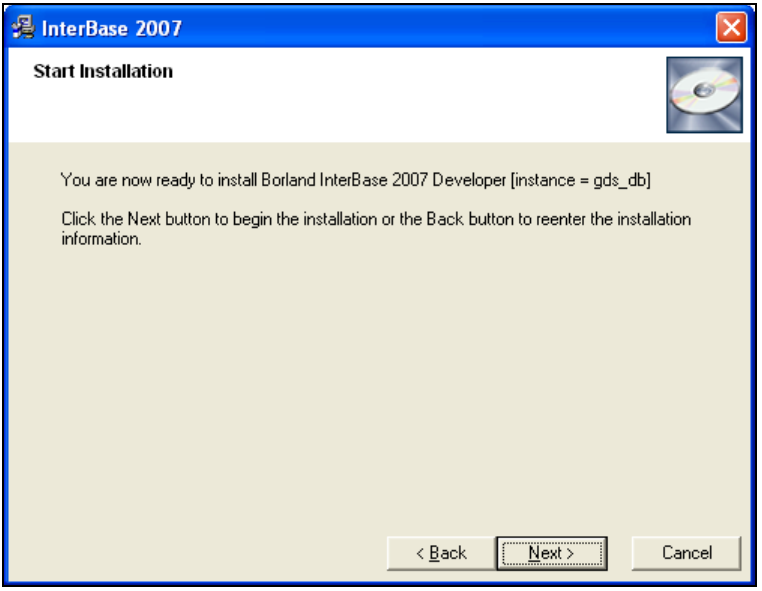


Рис. П1.20. Подготовка к инсталляции

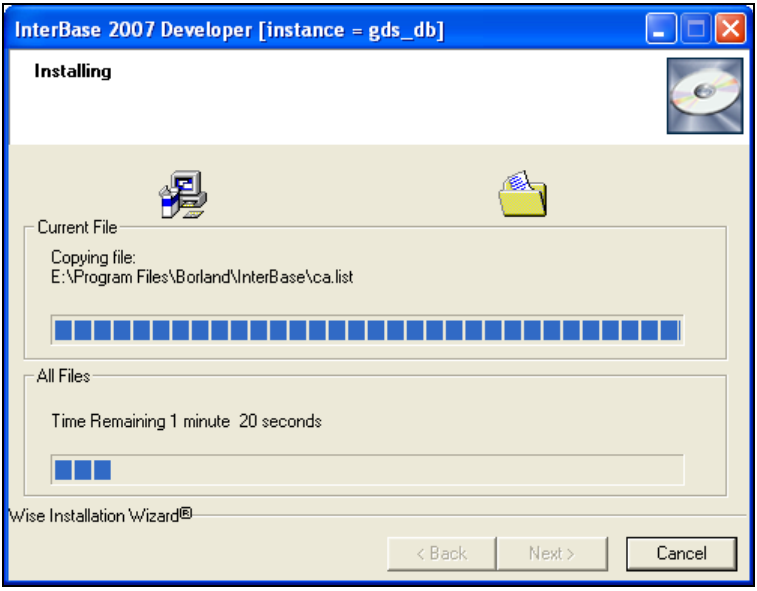


Рис. П1.21. Процесс инсталляции

Рис. П1.22. Завершение инсталляции

Будут появляться диалоговые окна, в которых вам нужно будет дать некоторые сведения о себе. В конце концов, в последнем окне укажите ваш e-mail и щелкните по гиперссылке **Create a new user account** для того, чтобы по указанному адресу получить ключ (рис. П1.23).

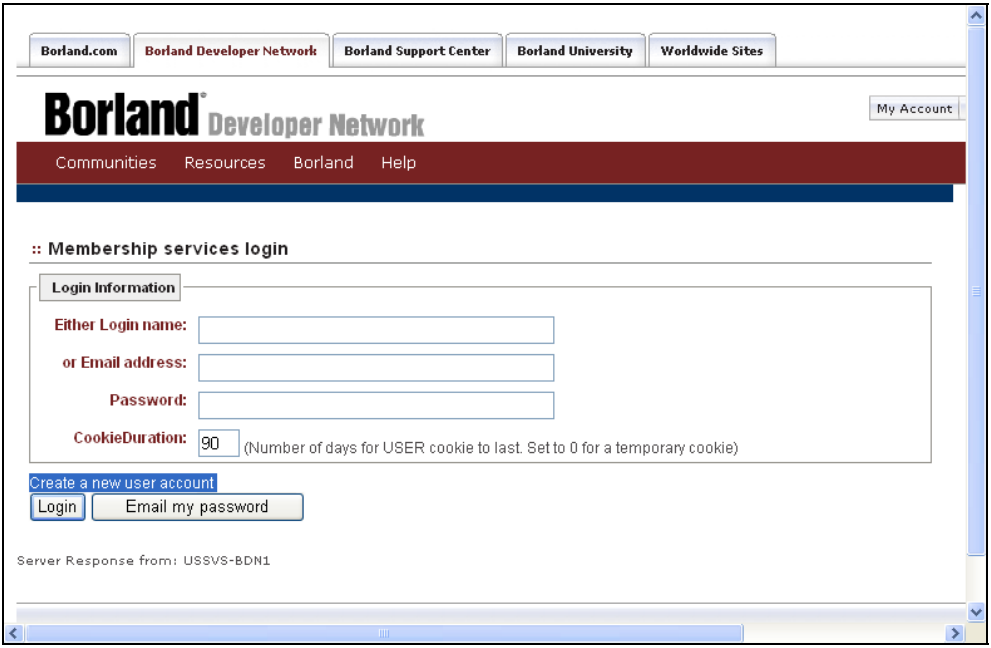


Рис. П1.23. Получение ключа

Полученный ключ нужно поместить в каталог инсталляции InterBase, в папку license.

Регистрация завершена. Вы можете использовать InterBase 2007 Developer Edition. Для этой версии у вас есть лицензия на 20 подключенных к базе данных пользователей.

П1.3. Установка компонентов FIBPlus

Установка компонентов FIBPlus на ваш компьютер потребует внимательности. Все действия по инсталляции следует выполнять только в указанной последовательности. Я описываю инсталляцию системы на тот момент, когда пишется эта книга. К тому времени, когда вы созреете для использования нормальной версии, все может измениться. Следите за публикациями.

Если у вас была установлена более ранняя версия, ее нужно удалить следующим образом.

2. Выберите в меню **Component | Install Packages** (Компонент | Инсталляция пакетов). В появившемся списке найдите группы компонентов, связанных с FIBPlus, и выбрав каждую группу, щелкните по кнопке **Remove** (Удалить).
3. Выберите в меню **Tools | Environment Options** (Инструменты | Режимы среды), перейдите на вкладку **Library** и удалите все пути, связанные с компонентами FIBPlus.
4. Удалите с диска сами компоненты.

> ***ВНИМАНИЕ!***
> После того как вы установите на своем компьютере среду разработки Delphi или C++Builder, не забудьте перезагрузить компьютер и обязательно вызовите на выполнение установленную программу. В противном случае вы можете получить крупные неприятности при попытках инсталлировать новые компоненты, если инсталляция выполняется не из самой среды.

П1.3.1. Установка полной версии

Запустите на выполнение инсталляционную программу. Принимайте все предложенные значения. Она лишь создаст в системном каталоге Program Files несколько вложенных каталогов (Devrace, FIBPlus, Delphi и несколько других) и поместит туда множество программных компонентов, необходимых для инсталляции.

После этого запустите Delphi. Обязательно закройте созданный по умолчанию проект — меню **File | Close All** (Файл | Закрыть все). Задайте для среды дополнительные пути к библиотекам. Для этого выберите в меню **Tools | Environment Options**. В появившемся окне выберите вкладку **Library** (рис. П1.24).

Вам нужно добавить несколько путей в список **Library path**. Щелкните по кнопке [...] справа от этого списка. Появится окно **Directories** (Каталоги) (рис. П1.25).

Здесь нужно щелкнуть по кнопке [...] в нижней половине окна и на появившейся форме выбрать путь к исходным текстам компонентов FIBPlus. Это может быть такой путь:

```
C:\Program Files\Devrace\FIBPlus\Delphi7\source
```

После этого в окне **Directories** станет доступной кнопка **Add**. Щелкните по этой кнопке, чтобы добавить путь к существующему списку. Аналогичным образом добавьте к списку путь

```
C:\Program Files\Devrace\FIBPlus\Delphi7\Editors
```

Закройте окна, щелкая по кнопке **OK**.

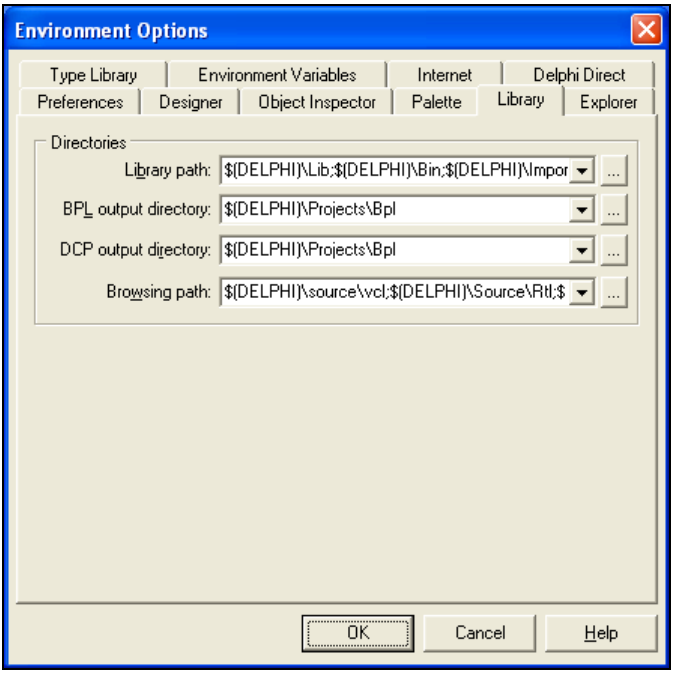


Рис. П1.24. Окно **Environment Options**

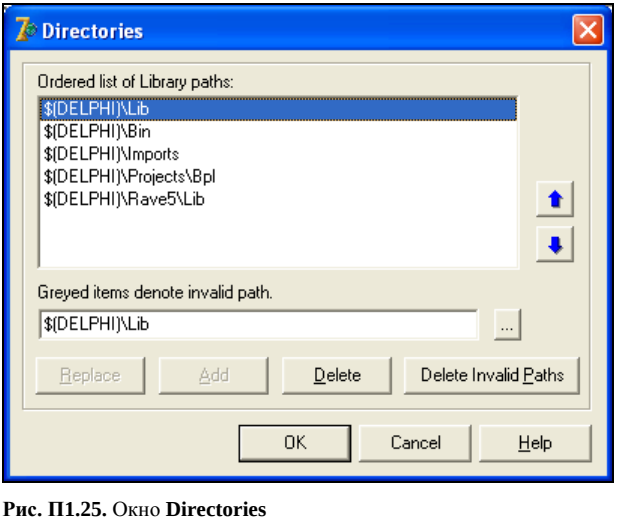


Рис. П1.25. Окно **Directories**

Выберите в меню **File | Open**. В диалоговом окне открытия файла для удобства выбора пакетов установите фильтр файлов:

```
Delphi package (*.dpk, *.dpkw)
```

Войдите в каталог

```
C:\Program Files\Devrace\FIBPlus\Delphi7\source
```

и выберите файл FIBPlus7.dpk. В появившемся окне щелкните по кнопке **Compile** для компиляции пакета, а затем по кнопке **Install** для его инсталляции (рис. П1.26). Появится информационное сообщение об установленных компонентах. Закройте окно. На запрос, сохранять ли изменения в пакетном файле, ответьте **No**.

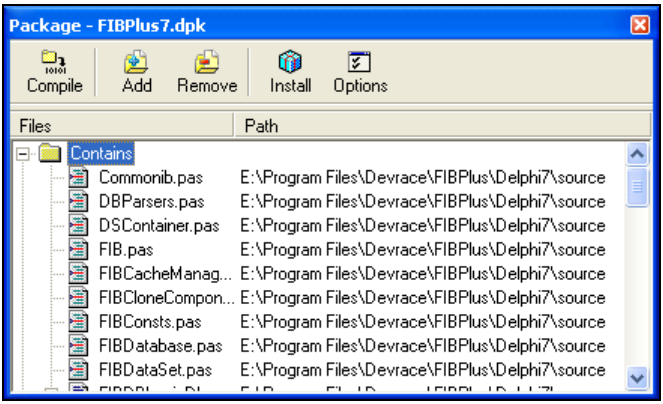


Рис. П1.26. Окно компиляции и инсталляции пакета

Выполните аналогичную инсталляцию, войдя в каталог editors и выбрав пакет FIBPlusEditors7.dpk. Затем вернитесь в предыдущий каталог source и инсталлируйте пакет FIBDBMidas7.dpk.

После этих инсталляций в Палитре компонентов у вас появятся две новые вкладки — **FIBPlus** и **FIBPlusServices**.

Если вы выполняете инсталляцию для других версий Delphi, то нужно выбирать пакетные файлы, в именах которых присутствует соответствующий номер версии.

Если же вы устанавливаете на компьютер не пробную (trial) версию компонентов FIBPlus, то вам нужно будет еще и установить инструменты. Для этого надо войти в каталог Tools и выполнить аналогичные действия по инсталляции инструментов.

П1.3.2. Установка пробной версии

Напоминаю, прежде чем устанавливать пробную версию, обязательно запустите на выполнение программу Delphi, убедитесь, что в Палитре компонентов присутствует достаточное количество вкладок. Закройте программу. Только после этого приступайте к установке компонентов.

В этом случае процесс установки гораздо более простой. Запустите на выполнение программу fibplus_setup.exe. Появится окно выбора языка (рис. П1.27). Вряд ли вам захочется изменить предложенное значение.

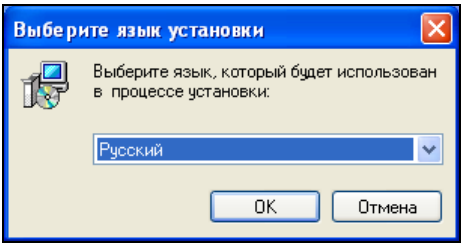


Рис. П1.27. Окно выбора языка

Щелкните по кнопке **OK**.
Следующее окно — информационное (рис. П1.28).

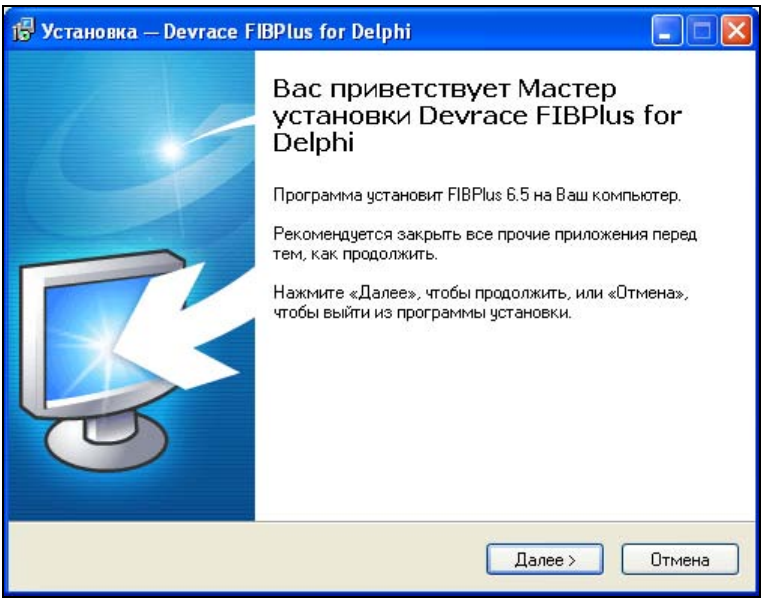


Рис. П1.28. Информационное окно

Щелкните по кнопке **Далее**. Появится окно лицензионного соглашения (рис. П1.29). Прочтите внимательно. Если у вас нет желания нарушать законы Российской Федерации и других стран — членов цивилизованного мирового сообщества, отметьте переключатель **Я принимаю условия соглашения** и щелкните по кнопке **Далее**. В противном случае отметьте **Я не принимаю условия соглашения** и прекращайте работать с компьютерными программами.

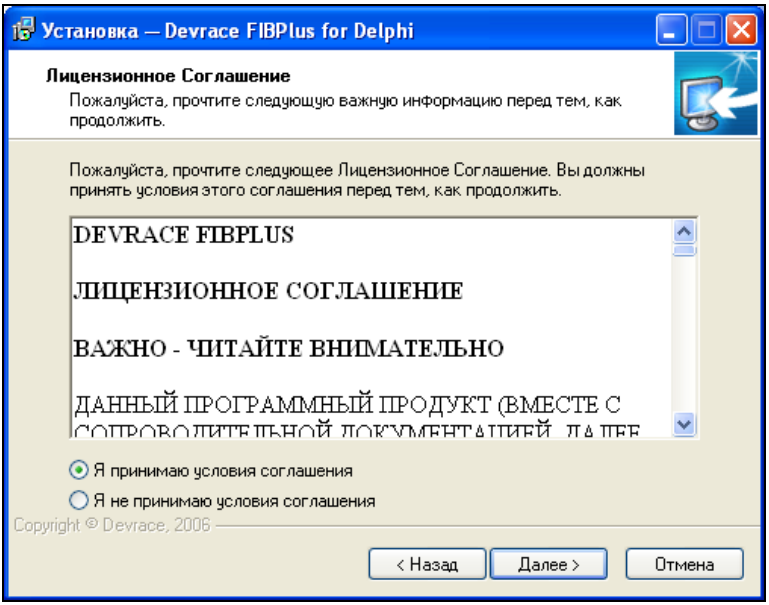


Рис. П1.29. Лицензионное соглашение

После правильного ответа появляется еще одно информационное окно (рис. П1.30), в котором вкратце рассказывается о компонентах и их возможностях, последних изменениях. Указываются расценки.

Щелкните по кнопке **Далее**. Появляется стандартное окно выбора папки для установки компонентов (рис. П1.31).

Похоже, что нормальным вариантом будет согласиться с предложенным размещением. Вы также можете выбрать свою папку. После этого щелкните по кнопке **Далее**.

У вас появится окно, где будут перечислены все обнаруженные компоненты, для которых на сегодняшний день создана инсталлируемая вами версия FIBPlus. Сейчас после переустановки Windows, когда меня замучили противные вирусы, на моем компьютере имеются следующие системы (в данном случае речь идет пока только об установке компонентов для Delphi; для C++Builder установка ничем не отличается от описываемой) — рис. П1.32.

Щелкаем по кнопке **Далее**. Появляется окно выбора режимов установки (рис. П1.33).

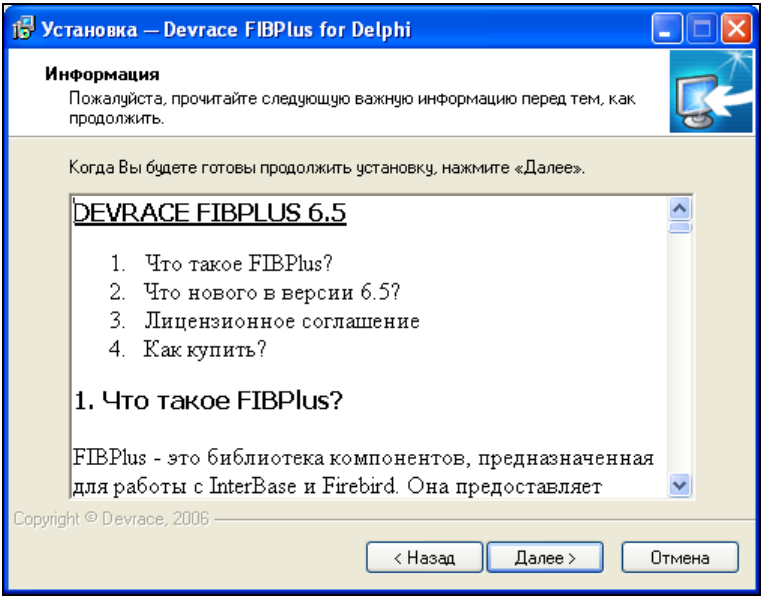


Рис. П1.30. Информационное окно

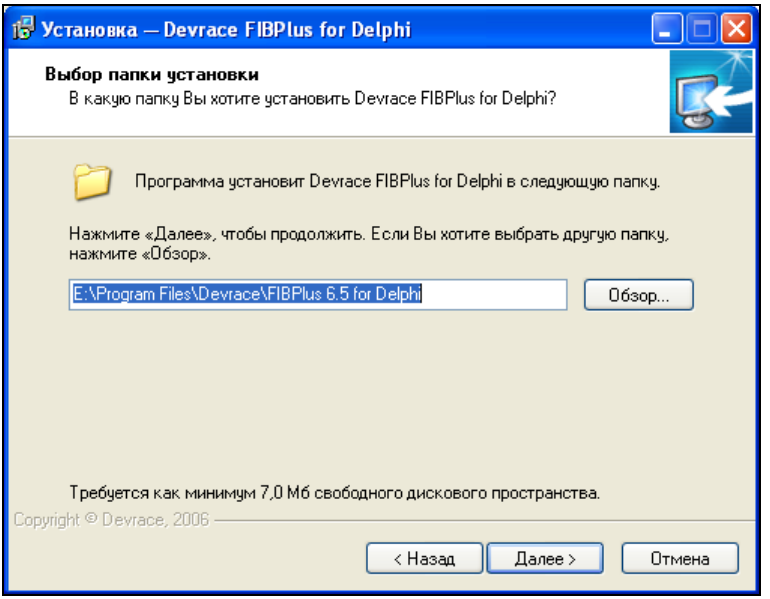


Рис. П1.31. Выбор папки установки компонентов

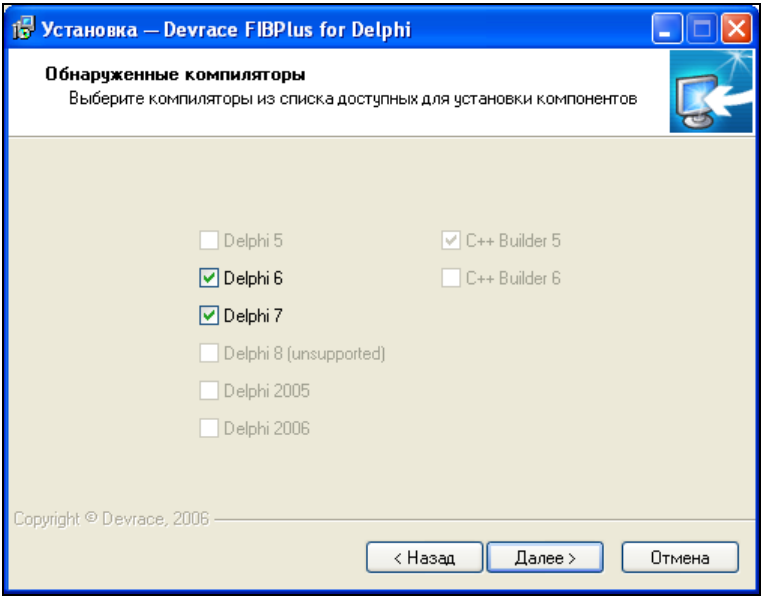


Рис. П1.32. Выбор программ для установки компонентов FIBPlus

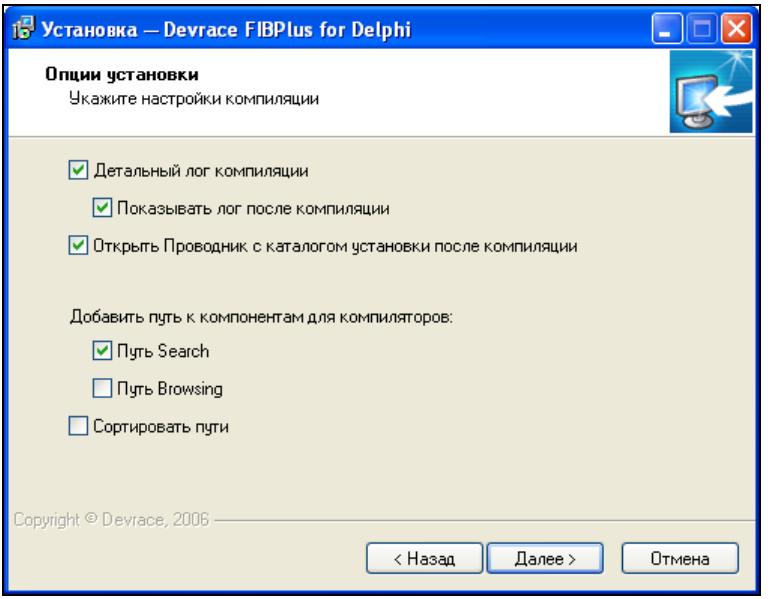


Рис. П1.33. Окно выбора режимов установки

Здесь имеет смысл сразу щелкнуть по кнопке **Далее**, чтобы не мучиться, какие выбрать условия компиляции. Тут же получаем окно выбора устанавливаемых компонентов (рис. П1.34).

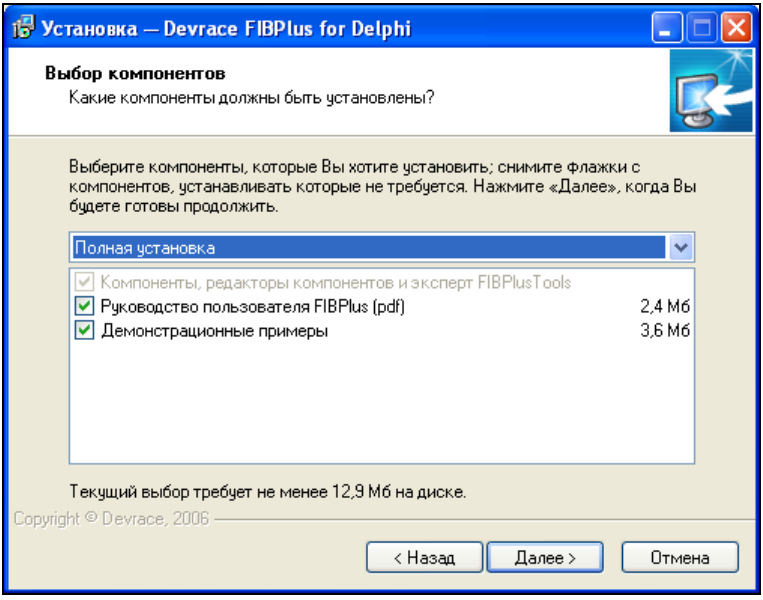


Рис. П1.34. Выбор устанавливаемых компонентов

Выбираем все, чтобы в свободное от программирования время почитать умную документацию (часть из которой, конечно, лучшую, написал я). Щелкаем по кнопке **Далее** (рис. П1.35).

Если хочется, измените папки и пути. Щелкайте по кнопке **Далее**. Установка параметров инсталляции заканчивается (рис. П1.36).

Нажимаем кнопку **Установить**, появляется окно процесса установки (рис. П1.37).

В следующем окне щелкаем по кнопке **Далее** (рис. П1.38).

В следующем окне показывается индикатор выполнения компиляции всех компонентов (рис. П1.39).

> ***Замечание***
> Я просто не успел получить скриншот в процессе компиляции. Поэтому оба индикатора заполнены полностью.

Последним окном после щелчка по кнопке **Далее** будет следующее — рис. П1.40.

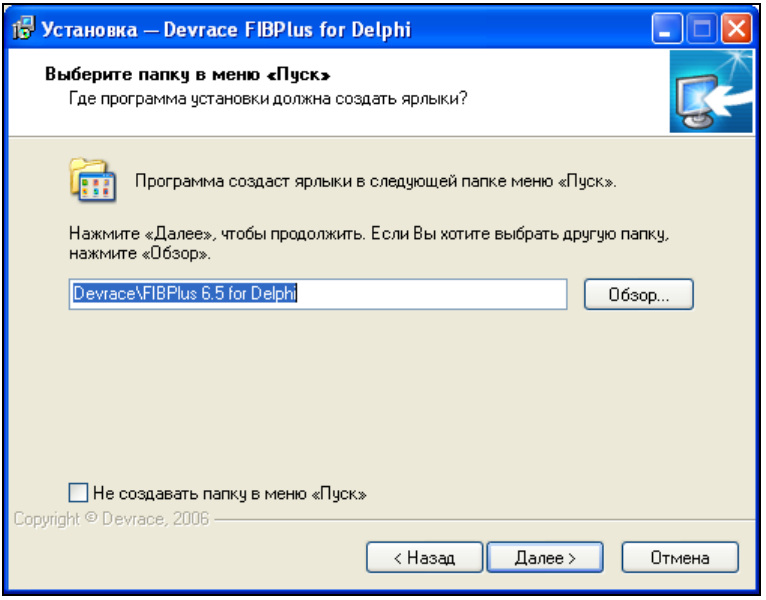


Рис. П1.35. Выбор папки для установки компонентов

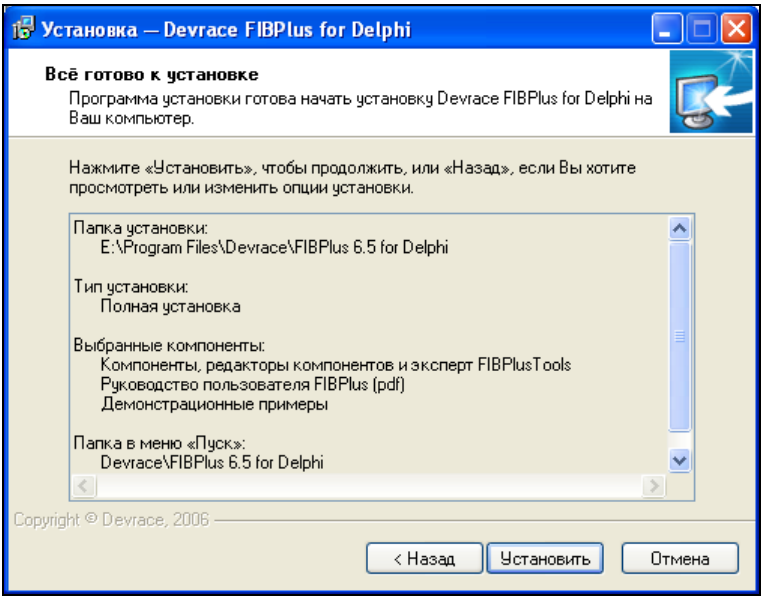


Рис. П1.36. Завершение установки

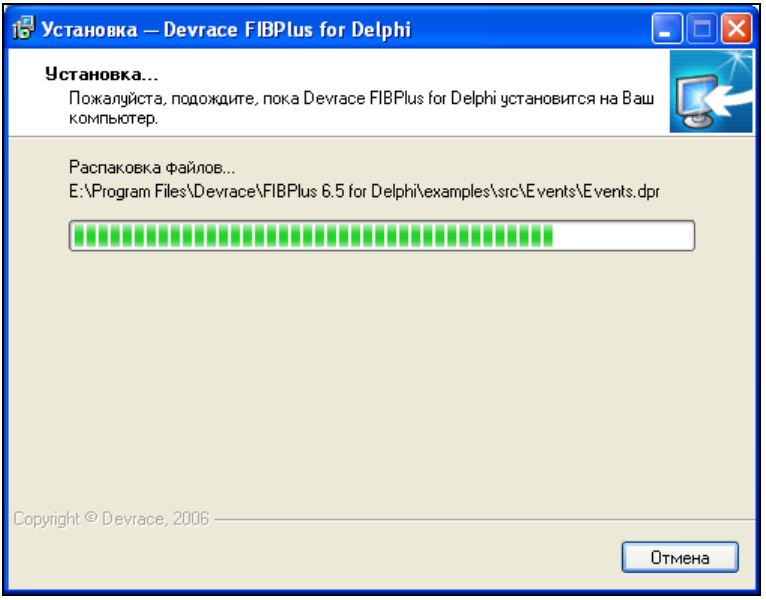


Рис. П1.37. Установка

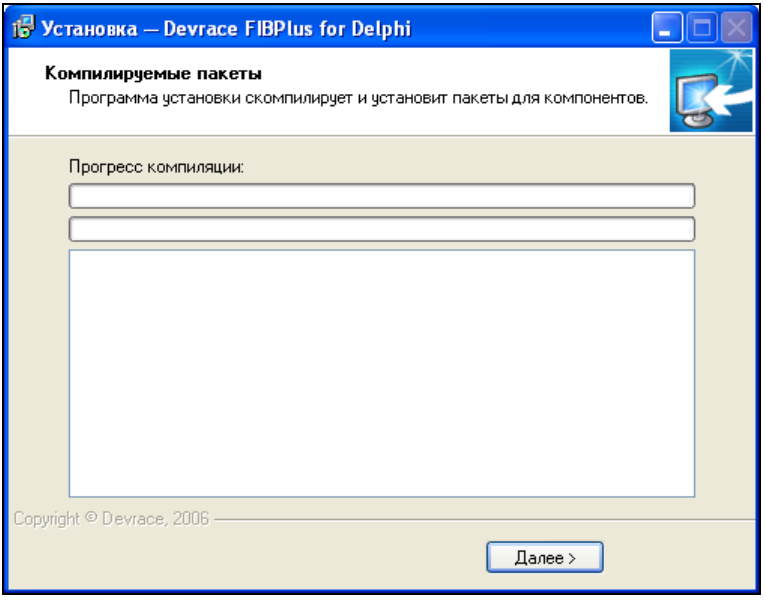


Рис. П1.38. Завершение установки компонентов

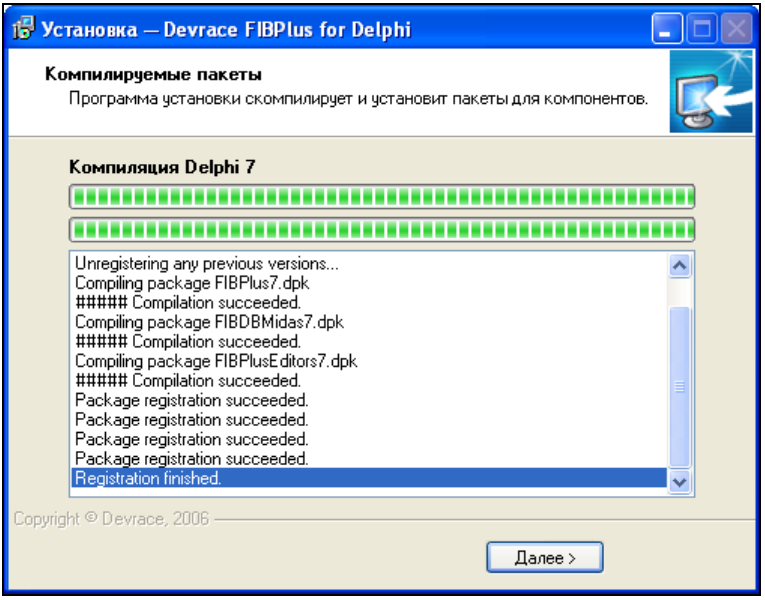


Рис. П1.39. Компиляция компонентов

Рис. П1.40. Полное завершение компиляции компонентов

После щелчка по кнопке **Завершить** во всех отмеченных на рис. П1.32 программных элементах будут установлены компоненты FIBPlus.

П1.4. Установка компонентов RichView

Если у вас установлена старая версия компонентов RichView, вам нужно деинсталлировать их, удалить в Delphi путь к старым компонентам и удалить сами компоненты с диска.

Установка компонентов RichView выполняется таким же образом, как и компонентов FIBPlus.

Как и в случае установки компонентов FIBPlus, добавьте путь к компонентам в среде Delphi. Поочередно инсталлируйте из каталога RichView пакеты RVPkgD7.dpk и RVDBPkgD7.dpk.

После этого в Палитре компонентов появится новая вкладка **RichView**.

П1.5. Установка компонентов FastReport

Установка этих компонентов является наиболее автоматизированной и требует минимальных усилий со стороны программиста.

Если у вас была установлена другая версия компонентов, их нужно деинсталлировать, щелкнув на рабочей панели по кнопке **Пуск**, выбрав **Все программы**, нужную версию FastReport и элемент меню **Uninstall**.

Для установки компонентов FastReport закройте Delphi. Запустите на выполнение программу инсталляции компонентов. Появится окно выбора языка установки (рис. П1.41).

В подобных случаях я всегда по привычке выбираю английский. И причина здесь не в пижонстве. Слишком уж часто русификация различных программ приводит к частичной или полной потере их функциональности. Щелкаем по кнопке **Next**.

После окна приветствия появляется окно лицензионного соглашения (рис. П1.42). Прочтите и согласитесь — отметьте флажок **Yes, I agree with all the terms of this license agreement** (Да, я согласен со всеми пунктами этого лицензионного соглашения).

После щелчка по кнопке **Next** вы увидите информационное окно. Можете прочесть полезную информацию прямо сейчас или отложить чтение на более позднее время, щелкнув по кнопке **Пуск**, выбрав в списке программ **FastReport** и затем элемент **Readme**. Следующее окно позволяет выбрать тип установки — **Full** (полная установка компонентов) или **Custom** (пользовательская) (рис. П1.43).

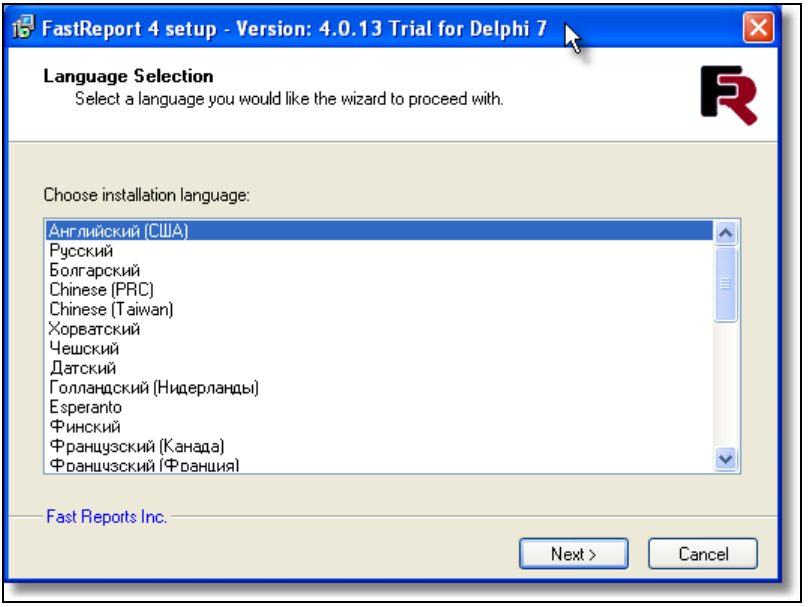


Рис. П1.41. Выбор языка установки

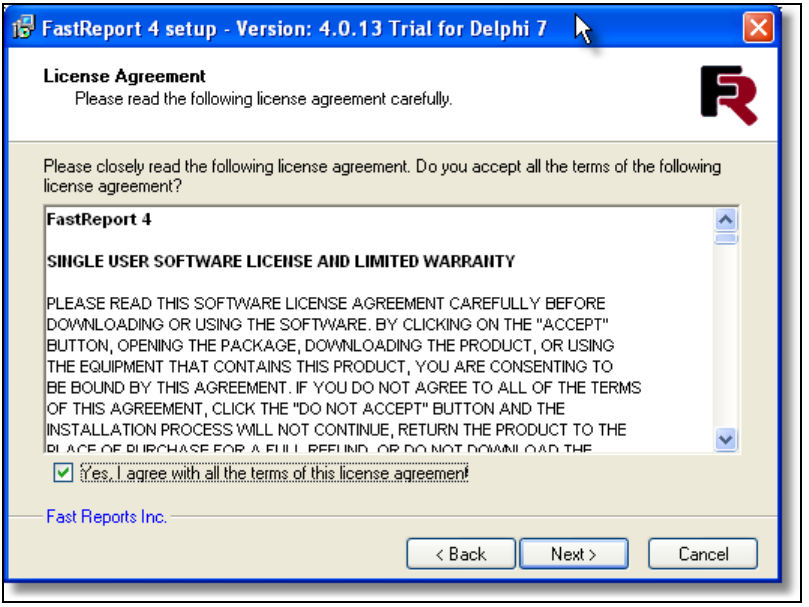


Рис. П1.42. Лицензионное соглашение

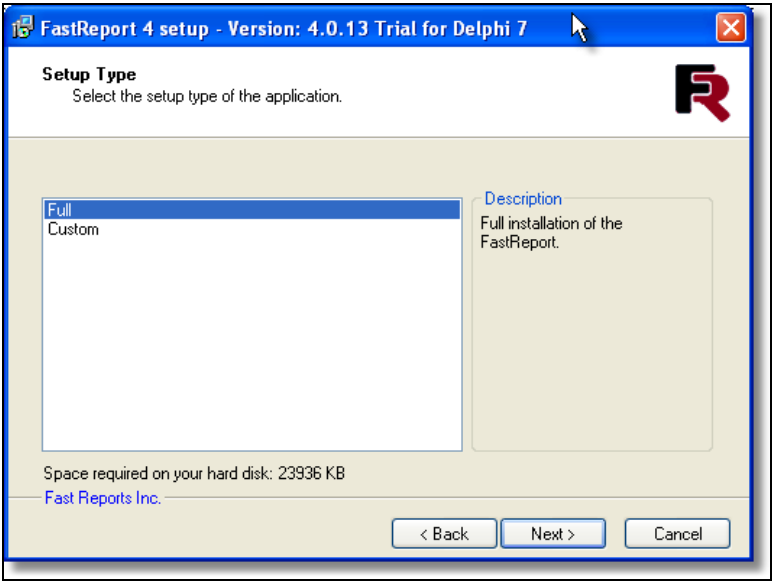


Рис. П1.43. Выбор вида установки

Выбираем полную установку и щелкаем по **Next**. Далее выбираем папку для установки компонентов (рис. П1.44).

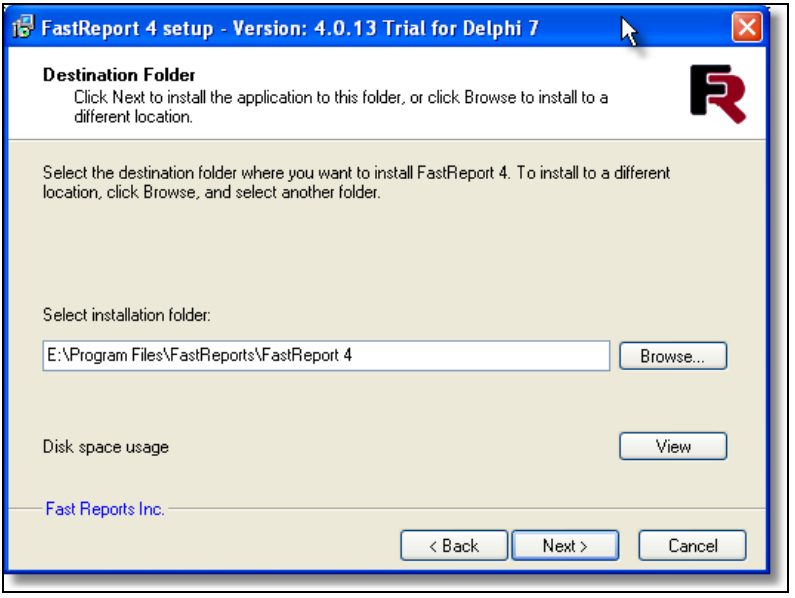


Рис. П1.44. Выбор папки для установки компонентов

Пропускаем несколько окон, щелкая по кнопке **Next**. Появляется окно, отображающее процесс установки (рис. П1.45).

Наконец появляется завершающее окно (рис. П1.46).

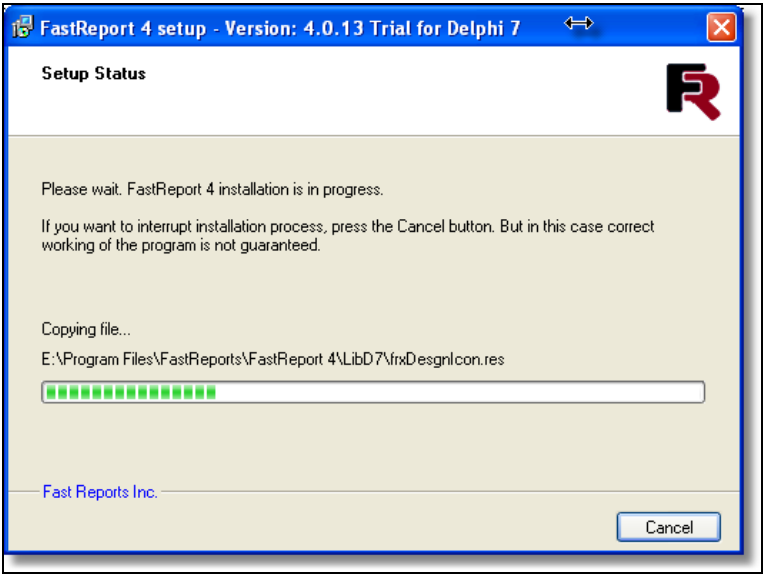


Рис. П1.45. Процесс установки компонентов

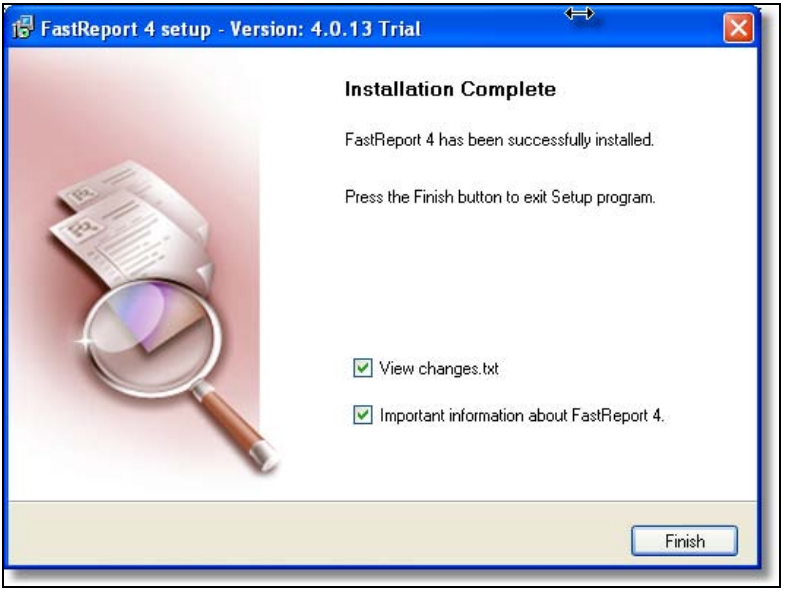


Рис. П1.46. Окно завершения установки

Запустите Delphi. Убедитесь, что в Палитре компонентов добавилось пять новых вкладок, относящихся к FastReport.

П1.6. Установка IBExpert

Вся установка IBExpert заключается только лишь в выполнении инсталляционной программы, которая извлекает два файла .exe и .dll в указанный каталог. Вызов программы IBExpert осуществляется не обычным для Windows способом через кнопку **Пуск**, а двойным щелчком мышью по программе.

Для удобства вызова программы вы можете на Рабочем столе создать ярлык, что я у себя сразу же и сделал после установки IBExpert.

ПРИЛОЖЕНИЕ 2

Наборы символов и порядок сортировки

Здесь я привожу список наборов символов и для каждого из них допустимые варианты порядка сортировки. Мы с вами рассмотрим структуру двух связанных между собою системных таблиц, хранящих эти данные. Затем напишем две программы (с использованием компонентов FIBPlus и IBX) для просмотра наборов символов и порядка сортировки в нашей учебной базе данных.

П2.1. Полный список наборов символов и порядков сортировки

В табл. П2.1 представлены наборы символов (`CHARACTER SET`) и порядок сортировки (`COLLATION ORDER`) для Firebird версии 1.5.3. Эту таблицу я скопировал у Хелен Борри, а затем проверил, написав две программы, которые мы с вами напишем заново.

Для любого столбца таблицы тем или иным образом (тремя описанными далее способами) должен быть явно установлен конкретный набор символов, иначе набором символов для такого столбца будет `NONE`, неприятных приключений в дальнейшей работе с которым вам не избежать. Знаю по своим разработкам. Можете мне поверить.

Напомню, что для большинства ваших русскоязычных приложений подойдет набор символов `WIN1251`, который позволяет хранить как все обычные символы, вводимые с клавиатуры, так и буквы кириллицы в любом регистре.

Набор символов может задаваться в виде набора символов по умолчанию для всей базы данных в предложении `DEFAULT CHARACTER SET` оператора `CREATE DATABASE (SCHEMA)`. Это первый и наиболее надежный способ установить подходящий набор символов для всех строковых столбцов и столбцов `BLOB` вашей базы данных. Тогда для всех столбцов и доменов, для которых не бу-

дет задано иного, набор символов будет установлен в набор символов по умолчанию. Централизованное хранение общих данных всегда лучше любого локального.

Второй способ. Набор символов также может задаваться для отдельного столбца таблицы в предложении `CHARACTER SET` оператора `CREATE TABLE` или `ALTER TABLE`. Это может быть использовано в том случае, когда вам нужно для отдельного столбца (или некоторых столбцов) использовать экзотические наборы символов.

Для группы столбцов, основанных на домене (третий способ), в описании домена также в предложении `CHARACTER SET` оператора `CREATE DOMAIN` или `ALTER DOMAIN` можно указать необходимый набор символов. Тогда все столбцы, основанные на этом домене, получат один и тот же набор символов. Опять же применяем, когда есть экзотика в некоторых столбцах отдельных таблиц[15].

Различные порядки сортировки (`COLLATION ORDER`) определяют в небольших границах способы упорядочения строковых столбцов в предложении `ORDER BY` предложения `SELECT`.

Для набора символов `WIN1251` существующие порядки сортировки определяют взаимную упорядоченность различных символов, в том числе строчных ("маленьких") букв и прописных (заглавных). Не следует ни при каких обстоятельствах использовать для этого набора символов порядок сортировки по умолчанию (`WIN1251`). Он слишком загадочный и не поддается никакой логике. При этом порядке сортировки *все* прописные буквы предшествуют *всем* строчным, т. е., например, строка "Аб" предшествует строке "аб", но при этом строка "Я" будет предшествовать строке "а". Буквы "е" и "Ё" вообще выпали из ряда букв кириллицы и располагаются где-то между латинскими и русскими буквами.

Я в своих программах всегда использую порядок сортировки `PXW_CYRL`. Такой же порядок сортировки используется и в нашей учебной базе данных. При этом порядке сортировки выполняется наиболее естественная упорядоченность строковых данных в так называемом лексикографическом порядке, когда положение строк в списке в точности соответствует положению букв в алфавите соответствующего языка.

- Всем буквам предшествуют цифры.
- Все латинские буквы предшествуют русским.

---

[15] Один мой знакомый профессионально начал заниматься мобильными телефонами. Выяснилось, что ему в работе с соответствующими программами позарез нужен корейский язык. Для этих целей в своей базе данных он стал использовать в описании домена, на котором основывается группа столбцов, специфический набор символов, по-моему, `KSC_5601`.

- Как в русском, так и в латинском алфавитах каждая строчная буква непосредственно предшествует соответствующей прописной букве. Буквы "е" и "Ё" находятся на своем месте.
- Знаки препинания разбросаны в разных частях списка упорядочения: некоторые до цифр, другие между цифрами и латинскими буквами, какие-то располагаются после букв кириллицы.

Вот в каком порядке будут сортироваться символы при использовании PXW_CYRL:

- специальные символы: пробел, !, ", #, $, %, &, ', (,), *, +, запятая, -, /;
- цифры от 0 до 9;
- специальные символы: :, ;, <, =, >, ?;
- буквы латинского алфавита: a, A, b, B, ... z, Z;
- буквы кириллицы: а, А, б, Б, ... я, Я (буквы е и Ё располагаются в правильном порядке);
- специальные символы: [, \,], ^, _, {, |, }, ~, №.

Такой же или почти такой же порядок получается и при использовании порядка сортировки WIN1251_UA. Различия только в расположении последней группы специальных символов. Для них порядок следующий: _, {, |, }, ~, [, \,], №, ^.

> ***Замечание***
> В InterBase 2007 отсутствует порядок сортировки WIN1251_UA.

Если вам в программе действительно будет нужно упорядочивать строковые данные, содержащие буквы, то лучшим решением помимо использования порядка сортировки PXW_CYRL будет представление таких строк в верхнем регистре — в виде прописных букв. Кроме того, это упростит вам жизнь при использовании в операторе SELECT условий выборки данных, где используются средства, чувствительные к регистру (LIKE, STARTING WITH).

Для других наборов символов, которые вы захотите использовать в своих разработках, вам придется провести собственное расследование поведения порядков сортировки — не очень сложное, но достаточно интересное.

Наборы символов хранятся в системной таблице RDB$CHARACTER_SETS. Порядок сортировки для каждого набора символов содержится в связанной системной таблице RDB$COLLATIONS.

> ***Замечание***
> Приведенные наборы символов и порядки сортировки присутствуют в Firebird 1.5. В других серверах и версиях баз данных некоторые из них могут отсутствовать. Проверьте их наличие в вашей конкретной системе управления базами данных.

Таблица П2.1. *Наборы символов и порядок сортировки для Firebird 1.5*

ID	Название	Байтов на символ	Порядок сортировки	Язык
2	ASCII	1	ASCII	Английский
56	BIG_5	2	BIG_5	Китайский, вьетнамский, корейский
50	CYRL	1	CYRL	Русский
			DB_RUS	Русский dBase
			PDOX_CYRL	Русский Paradox
10	DOS437	1	DOS437	Английский — США
			DB_DEU437	Немецкий dBase
			DB_ESP437	Испанский dBase
			DB_FIN437	Финский dBase
			DB_FRA437	Французский dBase
			DB_ITA437	Итальянский dBase
			DB_NLD437	Голландский dBase
			DB_SVE437	Шведский dBase
			DB_UK437	Английский (Великобритания) dBase
			DB_US437	Английский (США) dBase
			PDOX_ASCII	Кодовая страница Paradox—ASCII
			PDOX_SWEDFIN	Paradox Шведская / Финская кодовые страницы
			PDOX_INTL	Paradox международный английский кодовая страница
9	DOS737	1	DOS737	Греческий
15	DOS775	1	DOS775	Балтийский
11	DOS850	1	DOS850	Латинский I (нет символа Евро)
			DB_DEU850	Немецкий
			DB_ESP850	Испанский
			DB_FRA850	Французский
			DB_FRC850	Французский — Канада
			DB_ITA850	Итальянский
			DB_NLD850	Голландский
			DB_PTB850	Португальский — Бразилия
			DB_SVE850	Шведский
			DB_UK850	Английский — Великобритания
			DB_US850	Английский — США

Таблица П2.1 *(продолжение)*

ID	Название	Байтов на символ	Порядок сортировки	Язык
45	DOS852	1	DOS852	Латинский II
			DB_CSY	Чешский dBase
			DB_PLK	Польский dBase
			DB_SLO	Словацкий dBase
			PDOX_CSY	Чешский Paradox
			PDOX_HUN	Венгерский Paradox
			PDOX_PLK	Польский Paradox
			PDOX_SLO	Словацкий Paradox
46	DOS857	1	DOS857	Турецкий
			DB_TRK	Турецкий dBase
16	DOS858	1	DOS858	Латинский I с символом Евро
13	DOS860	1	DOS860	Португальский
			DB_PTG860	Португальский dBase
47	DOS861	1	DOS861	Исландский
			PDOX_ISL	Исландский Paradox
17	DOS862	1	DOS862	Иврит
14	DOS863	1	DOS863	Французский — Канада
			DB_FRC863	Французский dBase — Канада
18	DOS864	1	DOS864	Арабский
12	DOS865	1	DOS865	Скандинавские
			DB_DAN865	Датский dBase
			DB_NOR865	Норвежский dBase
			PDOX_NORDAN4	Paradox Норвегия и Дания
48	DOS866	1	DOS866	Русский
49	DOS869	1	DOS869	Современный греческий
6	EUCJ_0208	2	EUCJ_0208	Японские EUC
57	GB_2312	2	GB_2312	Упрощенный китайский (Гонконг, Корея)

Таблица П2.1 *(продолжение)*

ID	Название	Байтов на символ	Порядок сортировки	Язык
21	ISO8859_1	1	ISO8859_1	Латинский I
			DA_DA	Датский
			DE_DE	Немецкий
			DU_NL	Голландский
			EN_UK	Английский, Великобритания
			EN_US	Английский — США
			ES_ES	Испанский
			FI_FI	Финский
			FR_CA	Французский — Канада
			FR_FR	Французский
			IS_IS	Исландский
			IT_IT	Итальянский
			NO_NO	Норвежский
			PT_PT	Португальский
			SV_SV	Шведский
22	ISO8859_2	1	ISO8859_2	Латинский 2 — Центральная Европа (хорватский, чешский, венгерский, польский, румынский, сербский, словацкий, словенский)
			CS_CZ	Чешский
			ISO_HUN	Венгерский
23	ISO8859_3	1	ISO8859_3	Латинский 3 — Южная Европа (мальтийский, эсперанто)
34	ISO8859_4	1	ISO8859_4	Латинский 4 — Северная Европа (эстонский, латвийский, литовский, гренландский, саамский)
35	ISO8859_5	1	ISO8859_5	Кириллица (русский).\
36	ISO8859_6	1	ISO8859_6	Арабский
37	ISO8859_7	1	ISO8859_7	Греческий
38	ISO8859_8	1	ISO8859_8	Иврит
39	ISO8859_9	1	ISO8859_9	Латинский 5
40	ISO8859_13	1	ISO8859_1	Латинский 7 — Балтика

Таблица П2.1 *(продолжение)*

ID	Название	Байтов на символ	Порядок сортировки	Язык
44	KSC_5601	2	KSC_5601	Корейский
			KSC_DICTIONARY	Корейский — словарный порядок сортировки
19	NEXT	1	NEXT	Кодирование NeXTSTEP
			NXT_DEU	Немецкий
			NXT_FRA	Французский
			NXT_ITA	Итальянский
			NXT_US	Английский — США
0	NONE	1	NONE	Нейтральная кодовая страница. Перевод в верхний регистр ограничен кодами ASCII 97–122. Постарайтесь сделать так, чтобы этот набор символов никогда не появлялся в столбцах ваших баз данных
1	OCTETS	1	OCTETS	Двоичный символ
5	SJIS_0208	2	SJIS_0208	Японский
3	UNICODE_FSS	3	UNICODE_FSS	UNICODE
51	WIN1250	1	WIN1250	ANSI — Центральная Европа
			PXW_CSY	Чешский
			PXW_HUN	Венгерский
			PXW_HUNDC	Венгерский — словарная сортировка
			PXW_PLK	Польский
			PXW_SLO	Словацкий
52	WIN1251	1	WIN1251	ANSI кириллица
			WIN1251_UA	Украинский
			PXW_CYRL	Paradox кириллица (русский)
53	WIN1252	1	WIN1252	ANSI — Латинский I
			PXW_INTL	Английский интернациональный
			PXW_INTL850	Paradox многоязыковый Латинский I
			PXW_NORDAN4	Норвежский и датский
			PXW_SPAN	Paradox испанский
			PXW_SWEDFIN	Шведский и финский

Таблица П2.1 *(окончание)*

ID	Название	Байтов на символ	Порядок сортировки	Язык
54	WIN1253	1	WIN1253 PXW_GREEK	ANSI греческий Paradox греческий
55	WIN1254	1	WIN1254 PXW_TURK	ANSI турецкий Paradox турецкий
58	WIN1255	1	WIN1255	ANSI иврит
59	WIN1256	1	WIN1256	ANSI арабский
60	WIN1257	1	WIN1257	ANSI балтийский

П2.2. Структура системных таблиц

Системные таблицы RDB$CHARACTER_SETS и RDB$COLLATIONS, как и все остальные, автоматически создаются системой в каждой базе данных при создании "пустой" базы данных.

Несколько сокращенная структура системной таблицы RDB$CHARACTER_SETS представлена в табл. П2.2. Размеры строковых столбцов указаны для Firebird 1.5.

Таблица П2.2. *Структура системной таблицы* *RDB$CHARACTER_SETS*

Идентификатор столбца	Тип	Описание
RDB$CHARACTER_SET_NAME	CHAR(31)	Название набора символов
RDB$DEFAULT_COLLATE_NAME	CHAR(31)	Название последовательности сортировки по умолчанию для этого набора символов
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатор набора символов
RDB$SYSTEM_FLAG	SMALLINT	1, если набор символов был определен в системе при создании базы данных; 0 для набора символов, определенного пользователем
RDB$BYTES_PER_CHARACTER	SMALLINT	Размер символов в наборе, указанный в байтах

Сокращенная структура системной таблицы RDB$COLLATIONS представлена в табл. П2.3.

Таблица П2.3. *Структура системной таблицы* ***RDB$COLLATIONS***

Идентификатор столбца	Тип	Описание
RDB$COLLATION_NAME	VARCHAR(31)	Имя последовательности сортировки
RDB$COLLATION_ID	SMALLINT	Идентификатор последовательности сортировки
RDB$CHARACTER_SET_ID	SMALLINT	Идентификатор набора символов
RDB$SYSTEM_FLAG	SMALLINT	Определенное пользователем (0); определенное в системе (1 или выше)

П2.3. Для особо одаренных. Программа просмотра набора символов и порядка сортировки

П2.3.1. Использование компонентов FIBPlus

Напишем небольшую программу для отображения наборов символов и порядка сортировки, хранящихся в нашей учебной базе данных. Между этими двумя системными таблицами существует уже хорошо известная нам с вами связь главная — подчиненная, или master — detail. Мы можем использовать технику разработки программ для таблиц такого типа (см. в *разд. 6.3* программу работы со справочниками стран и регионов).

Запустите Delphi, создайте новый проект (меню **File | New | Application**). Положите на форму две панели. Одну выровняйте по левому краю и тут же положите на форму разделитель (Splitter), выровняв его также по левому краю. Другую панель выровняйте по всей клиентской области.

На обе панели положите еще по панели и выровняйте их по верхнему краю. Здесь будут располагаться данные о количестве наборов символов и о количестве порядков сортировки для текущего набора символов соответственно. На каждую "большую" панель также положите по компоненту DBGrid и выровняйте их по всей родительской поверхности. В левой сетке будут отображаться наборы символов, в правой — соответствующие порядки сортировки.

Положите на форму с вкладки **FIBPlus** компоненты Database, Transaction, два компонента DataSet, присвоив им имена Database1, Transaction1,

DataSetCharacterSet и DataSetCollation. С вкладки **DataAccess** положите два компонента DataSource, присвоив им имена DataSourceCharacterSet и DataSourceCollation.

Форма должна выглядеть следующим образом — рис. П2.1.

Установите обычным образом характеристики компонента Database и свяжите его с нашей учебной базой данных.

Вызовите диалоговое окно **SQL Generator** для компонента DataSetCharacterSet и задайте следующий оператор SELECT:

```
SELECT RDB$CHARACTER_SET_NAME,
       RDB$CHARACTER_SET_ID,
       RDB$BYTES_PER_CHARACTER,
       RDB$DEFAULT_COLLATE_NAME,
       RDB$SYSTEM_FLAG
FROM RDB$CHARACTER_SETS
  ORDER BY RDB$CHARACTER_SET_NAME
```

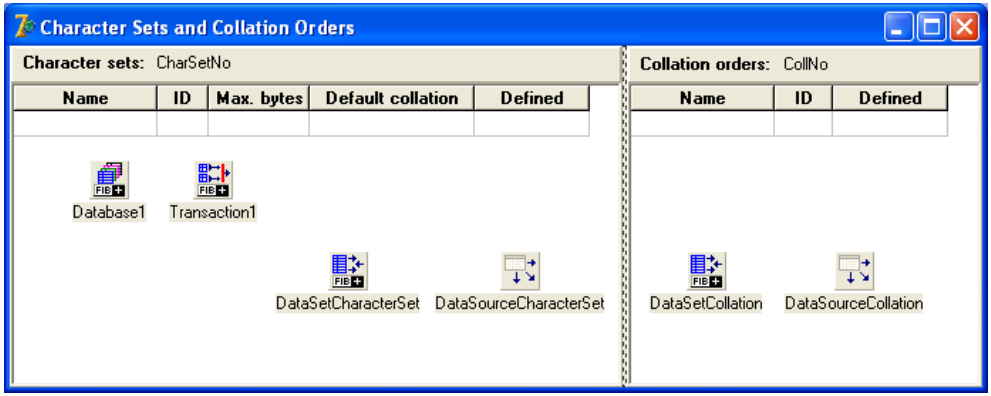


Рис. П2.1. Программа отображения наборов символов и порядка сортировки

Другие операторы SQL нет необходимости генерировать — мы не собираемся вносить изменения в системные таблицы.

Для компонента DataSetCollation создайте следующий оператор SELECT:

```
SELECT RDB$COLLATION_NAME,
       RDB$COLLATION_ID,
       RDB$SYSTEM_FLAG
  FROM RDB$COLLATIONS
WHERE
  RDB$CHARACTER_SET_ID = ?RDB$CHARACTER_SET_ID
ORDER BY RDB$COLLATION_NAME
```

Здесь мы также не будем создавать никаких операторов, выполняющих модификацию этой системной таблицы.

Свяжите этот компонент с первым компонентом набора данных, установив его свойство `DataSource` в `DataSourceCharacterSet`. В свойстве `DetailConditios` установите в `True` подсвойства `dcForceOpen` и `dcWaitEndMasterScroll`. Это позволит автоматически выполнять открытие дочернего набора данных и это открытие осуществлять с задержкой, задаваемой в свойстве набора данных `WaitEndMasterInterval` во время выполнения программы.

В результате подчиненный набор данных, содержащий порядок сортировки для набора символов, будет автоматически открываться при изменении текущего набора символов. Он будет содержать только те строки, которые относятся к данному набору символов.

Отредактируйте содержимое сеток `DBGrid` отображения наборов данных, как показано на рис. П2.1.

Чтобы помещать на форму в информационную панель количество порядков сортировки, нужно написать следующий обработчик, который будет вызываться сразу после автоматического открытия набора данных `DataSetCollation` — листинг П2.1.

Листинг П2.1. Обработчик события открытия набора данных `DataSetCollation`

```
procedure TFormMain.DataSetCollationAfterOpen(DataSet: TDataSet);
begin
  CollNo.Caption := IntToStr(DataSetCollation.RecordCount);
end;
```

Напишите обработчик события отображения формы (листинг П2.2).

Листинг П2.2. Обработчик события отображения формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin
  Database1.Connected := True;
  DataSetCharacterSet.Open;
  DataSetCollation.WaitEndMasterInterval := 500;
  CharSetNo.Caption := IntToStr(DataSetCharacterSet.RecordCount);
  DBGridCharacterSet.SetFocus;
end;
```

Здесь мы соединяемся с нашей базой данных, открываем родительский набор данных (набор символов), устанавливаем задержку повторного открытия до-

чернего набора данных (порядок сортировки), выводим в информационную панель количество наборов символов и устанавливаем фокус ввода на сетку, отображающую наборы символов.

Напомню, что свойство `WaitEndMasterInterval` набора данных задает интервал времени задержки переоткрытия подчиненного набора данных (порядка сортировки) в миллисекундах. Однако есть случаи, когда в некоторых системах эта задержка несколько больше. Вам следует опытным путем выбрать наиболее удобное время задержки.

Соответствующие действия по закрытию наборов данных и разъединению с базой данных выполняются и при закрытии формы (листинг П2.3).

Листинг П2.3. Обработчик события закрытия формы

```
procedure TFormMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  DataSetCollation.Close;
  DataSetCharacterSet.Close;
  Database1.Connected := False;
end;
```

Интересные моменты в этой программе связаны с прорисовкой ячеек, содержащих некоторые признаки.

В системной таблице `RDB$CHARACTER_SETS` есть столбец `RDB$SYSTEM_FLAG`, который содержит данные о том, является ли набор символов определенным системой (тогда значение этого столбца равняется 1), или он определен пользователем (значение 0). Конечно, такие данные можно было бы отображать "как есть", выводя на экран 1 или 0, однако для пользователя это не может быть достаточно удобным.

Для начала нам нужно скорректировать имя поля таблицы (свойство `FieldName`) последнего столбца в сетке, отображающей список наборов символов (`DBGridCharacterSet`). Щелкните правой кнопкой мыши по сетке и в контекстном меню выберите **Columns Editor**. Выделите последний столбец в списке столбцов и измените значение его свойства `FieldName` с имени столбца `RDB$SYSTEM_FLAG` на несуществующее имя `RDB$SYSTEM_FLAG_0`. Такое изменение требуется для того, чтобы программа не выводила в этот столбец 0 или 1. Вывод подходящего текста мы сейчас подготовим.

Нужно написать обработчик события прорисовки ячейки сетки отображения списка набора символов. Выделите на форме компонент `DBGridCharacterSet` и на вкладке **Events** дважды щелкните мышью справа от события `OnDrawColumnCell` (событие прорисовки каждой ячейки столбца таблицы). В появившейся заготовке введите код, показанный в листинге П2.4.

Листинг П2.4. Обработчик события прорисовки ячейки для набора символов

```
procedure TFormMain.DBGridCharacterSetDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var S: String;
begin
  if Column.FieldName <> 'RDB$SYSTEM_FLAG_0' then exit;
  if DataSetCharacterSet.FieldByName('RDB$SYSTEM_FLAG').AsString = '1'
then
    S := 'System defined'
  else
    S := 'User defined';
  DBGridCharacterSet.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
end;
```

Действия в этой процедуре будут выполняться, только если прорисовывается нужный нам столбец `RDB$SYSTEM_FLAG_0`, которого на самом деле не существует в таблице в базе данных:

```
if Column.FieldName <> 'RDB$SYSTEM_FLAG_0' then exit;
```

Если значение столбца `RDB$SYSTEM_FLAG` равняется 1, то внутренней переменной `S` присваивается текст `'System defined'`. Иначе — `'User defined'`. Само помещение полученного текста в ячейку сетки выполняется оператором

```
DBGridCharacterSet.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
```

Здесь используется свойство сетки `Canvas`, дающее возможность выполнять вывод текста и рисовать любые линии, и метод этого свойства `TextOut`, позволяющий поместить по указанным координатам строку символов.

Точно такие же вещи следует сделать и для столбца сетки отображения последовательностей сортировки: присвоить последнему столбцу имя `RDB$SYSTEM_FLAG_0` и написать обработчик события прорисовки ячейки столбца (листинг П2.5).

Листинг П2.5. Обработчик события прорисовки ячейки для последовательностей сортировки

```
procedure TFormMain.DBGridCollationDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var S: String;
begin
  if Column.FieldName <> 'RDB$SYSTEM_FLAG_0' then exit;
  if DataSetCollation.FieldByName('RDB$SYSTEM_FLAG').AsString >= '1' then
```

```
    S := 'System defined'
  else
    S := 'User defined';
  DBGridCollation.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
end;
```

Запустите программу на выполнение. Посмотрите, сколько и какие наборы символов поддерживает ваш сервер базы данных. На сегодняшний день для Firebird 1.5 их 44. В InterBase 2007 — 31. При перемещении по списку наборов символов в правой части формы будет появляться список соответствующих порядков сортировки.

Изменяйте размеры формы и, используя разделитель (splitter), меняя размеры окон отображения списка наборов символов и порядка сортировки, получите самый приличный вид отображения этих данных.

П2.3.2. Использование компонентов IBX

Скопируйте созданный проект в другой каталог, удалите компоненты FIBPlus. С вкладки **InterBase** положите на форму компоненты: базы данных (`TIBDatabase`), транзакции (`TIBTransaction`) и два компонента набора данных (`TIBDataSet`). Задайте им нужные имена, соответственно: `Database1`, `Transaction1`, `DataSetCharacterSet`, `DataSetCollation`.

Для компонента базы данных в свойстве `DatabaseName` задайте полный путь к учебной базе данных, собственно, можно задать любую базу данных, при этом в списке `Params` нужно будет указать правильное имя и пароль. В нашем же случае список `Params` должен содержать следующие строки:

```
lc_ctype=WIN1251
user_name=wizard
password=master
```

Для компонента транзакции оставьте для всех свойств значения по умолчанию.

Для набора данных `DataSetCharacterSet` нужно сформировать свойство `SelectSQL`. Щелкните мышью справа от этого свойства по кнопке  и в окне SQL введите оператор `SELECT`:

```
SELECT RDB$CHARACTER_SET_NAME,
       RDB$CHARACTER_SET_ID,
       RDB$BYTES_PER_CHARACTER,
       RDB$DEFAULT_COLLATE_NAME,
```

```
       RDB$SYSTEM_FLAG
FROM RDB$CHARACTER_SETS
  ORDER BY RDB$CHARACTER_SET_NAME
```

Разумеется, предварительно компонент нужно связать с компонентом базы данных и указать транзакцию. Другие установки для него не требуются.

Для компонента `DataSetCollation` также нужно задать базу данных, транзакцию, в свойстве `DataSource` указать `DataSourceCharacterSet` и сформировать оператор `SELECT` в свойстве `SelectSQL`:

```
SELECT DB$COLLATION_NAME,
       RDB$COLLATION_ID,
       RDB$SYSTEM_FLAG
  FROM RDB$COLLATIONS
WHERE
  RDB$CHARACTER_SET_ID = ?RDB$CHARACTER_SET_ID
ORDER BY RDB$COLLATION_NAME
```

Обработчик события отображения формы в этом варианте будет выглядеть несколько иначе — листинг П2.6.

Листинг П2.6. Обработчик события отображения формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin
  Database1.Connected := True;
  DataSetCharacterSet.Open;
  DataSetCollation.Open;
  CharSetNo.Caption := IntToStr(DataSetCharacterSet.RecordCount);
  DBGridCharacterSet.SetFocus;
end;
```

Здесь мы явно должны выполнить первоначальное открытие набора данных порядка сортировки. В дальнейшем при перемещении по списку наборов символов переоткрытие этого набора данных будет выполняться автоматически.

У компонента набора данных в IBX нет свойства `WaitEndMasterInterval`, поэтому здесь мы не устанавливаем задержку открытия подчиненного набора данных.

Обработка прорисовки ячеек в обоих наборах данных выполняется точно таким же образом, как и при использовании компонентов FIBPlus, — вам тут ничего менять или добавлять не придется.

Запустите программу на выполнение. Убедитесь, что она работает почти так же, как и в случае компонентов FIBPlus. "Почти" потому, что открытие подчиненного набора данных происходит сразу же, как только изменяется текущая запись набора символов. Здесь нельзя задавать задержку переоткрытия. Это не так страшно, если работа идет на локальном компьютере. Более грустная картина получается, если компьютер работает в сети. Тогда постоянно будут выполняться не всегда нужные переоткрытия подчиненного набора данных, а это несколько ухудшит временные характеристики и может привести к большому увеличению сетевого трафика. Речь, конечно, идет не столько об этой программе, сколько о подобных программах вида главная-подчиненная, где в базе данных содержатся большие объемы обрабатываемых данных.

ПРИЛОЖЕНИЕ 3

Коды ошибок

Коды ошибок (табл. П3.1) я опять же привожу по Хелен Борри. Это мой перевод текстов. Если что не так, претензии ко мне, а не к ней. Все коды точно известны в Firebird 1.5.0. В остальных системах могут быть разночтения. Мне такие не известны. Если вы установили на своем компьютере Firebird версии 1.5.0, то соответствие кодов ошибок вашему серверу базы данных обеспечено.

Я не уверен, что вам в вашей практической деятельности потребуются сведения по всем этим ошибкам, однако, поскольку ваши потребности никому заранее не известны (даже вам), лучше привести все существующие варианты, из которых в ваших программах вы выберете то, что нужно.

Таблица П3.1. *Коды ошибок Firebird 1.5.0*

SQLCODE	GDSCODE	Символ	Текст сообщения
101	335544366	segment	-Segment buffer length shorter than expected. Длина сегмента буфера меньше, чем ожидается
100	335544338	from_no_match	-No match for first value expression. Нет соответствия для первого значения выражения
100	335544354	no_record	-Invalid database key. Неверный ключ базы данных
100	335544367	segstr_eof	-Attempted retrieval of more segments than exist. Попытка поиска большего количества сегментов, чем существует

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
100	335544374	stream_eof	-Attempt to fetch past the last record in a record stream. Попытка загрузки в поток записей после последней записи
-84	335544554	nonsql_security_rel	-Table/procedure has non-SQL security class defined. Для таблицы/процедуры определен класс безопасности, не являющийся SQL
-84	335544555	nonsql_security_fld	-Column has non-SQL security class defined. Для столбца определен класс безопасности, не являющийся SQL
-84	335544668	dsql_procedure_use_err	-Procedure <string> does not return any values. Процедура <строка> не возвращает никакого значения
-85	335544747	usrname_too_long	-The username entered is too long. Maximum length is 31 bytes. Введенное имя пользователя слишком длинное. Максимальная длина 31 байт
-85	335544748	password_too_long	-The password specified is too long. Maximum length is 8 bytes. Указанный пароль слишком длинный. Максимальная длина 8 байт
-85	335544749	usrname_required	-A username is required for this operation. Для этой операции требуется имя пользователя

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-85	335544750	password_required	-A password is required for this operation. Для этой операции требуется пароль
-85	335544751	bad_protocol	-The network protocol specified is invalid. Указан неверный сетевой протокол
-85	335544752	dup_usrname_found	-A duplicate user name was found in the security database. В базе данных безопасности обнаружено дублирование имен пользователей
-85	335544753	usrname_not_found	-The user name specified was not found in the security database. Указанное имя пользователя не найдено в базе данных безопасности
-85	335544754	error_adding_sec_record	-An error occurred while attempting to add the user. Появилась ошибка при попытке добавления пользователя
-85	335544755	error_modifying_sec_record	-An error occurred while attempting to modify the user record. Появилась ошибка при попытке изменения записи пользователя
-85	335544756	error_deleting_sec_record	-An error occurred while attempting to delete the user record. Появилась ошибка при попытке удаления записи пользователя

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-85	335544757	error_updating_sec_db	-An error occurred while updating the security database. Появилась ошибка при изменении базы данных безопасности
-103	335544571	dsql_constant_err	-Data type for constant unknown. Неизвестен тип данных для константы
-104	335544343	invalid_blr	-Invalid request BLR at offset <number>. Неверный запрос в BLR со смещением <число>
-104	335544390	syntaxerr	-BLR syntax error: expected <string> at offset <number>, encountered <number>. Ошибка синтаксиса BLR: ожидается <строка> по смещению <число>, встречено <число>
-104	335544425	ctxinuse	-Context already in use (BLR error). Контекст находится в использовании (ошибка BLR)
-104	335544426	ctxnotdef	-Context not defined (BLR error). Контекст не определен (ошибка BLR)
-104	335544429	badparnum	-Bad parameter number. Неверный номер параметра
-104	335544440	bad_msg_vec	—
-104	335544456	invalid_sdl	-Invalid slice description language at offset <number>. Неверный фрагмент языка описания по смещению <число>

Таблица П3.1 *(продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	335544570	dsql_command_err	-Invalid command. Неверная команда
-104	335544579	dsql_internal_err	-Internal error. Внутренняя ошибка
-104	335544590	dsql_dup_option	-Option specified more than once. Режим указан более одного раза
-104	335544591	dsql_tran_err	-Unknown transaction option. Неизвестный режим транзакции
-104	335544592	dsql_invalid_array	-Invalid array reference. Неверная ссылка на массив
-104	335544608	command_end_err	-Unexpected end of command. Неверное завершение команды
-104	335544612	token_err	-Token unknown. Неизвестный синтаксический элемент
-104	335544634	dsql_token_unk_err	-Token unknown—line <number>, char <number>. Неизвестный синтаксический элемент — строка <число>, символ <число>
-104	335544709	dsql_agg_ref_err	-Invalid aggregate reference. Неверная ссылка на агрегат
-104	335544714	invalid_array_id	-Invalid blob id. Неверный идентификатор BLOB

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	335544730	cse_not_supported	-Client/Server Express not supported in this release. Client/Server Express не поддерживается в этом релизе
-104	335544743	token_too_long	-Token size exceeds limit. Размер синтаксического элемента превышает предел
-104	335544763	invalid_string_constant	-A string constant is delimited by double quotes. Строковая константа определена в кавычках
-104	335544764	transitional_date	-DATE must be changed to TIMESTAMP. DATE должно быть изменено в TIMESTAMP
-104	335544796	sql_dialect_datatype_unsupport	-Client SQL dialect <number> does not support reference to <string> datatype. SQL-диалект клиента <номер> не поддерживает ссылку на тип данных <строка>
-104	335544798	depend_on_uncommitted_rel	-You created an indirect dependency on uncommitted metadata. You must roll back the current transaction. Вы создали непрямую зависимость на неподтвержденные метаданные. Вы должны отменить текущую транзакцию

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	335544821	dsql_column_pos_err	-Invalid column position used in the <string> clause. В предложении <строка> используется неверная позиция столбца
-104	335544822	dsql_agg_where_err	-Cannot use an aggregate function in a WHERE clause, use HAVING instead. Невозможно использовать агрегирующую функцию в предложении WHERE, используйте вместо этого HAVING
-104	335544823	dsql_agg_group_err	-Cannot use an aggregate function in a GROUP BY clause. Невозможно использовать агрегирующую функцию в предложении GROUP BY
-104	335544824	dsql_agg_column_err	-Invalid expression in the <string> (not contained in either an aggregate function or the GROUP BY clause). Неверное выражение в <строка> (не содержится ни в агрегирующей функции, ни в предложении GROUP BY)
-104	335544825	dsql_agg_having_err	-Invalid expression in the <string> (neither an aggregate function nor a part of the GROUP BY clause). Неверное выражение в <строка> (не агрегирующая функция, не предложение GROUP BY)

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-104	335544826	dsql_agg_nested_err	-Nested aggregate functions are not allowed. Вложенные агрегирующие функции недопустимы
-104	336003075	dsql_transitional_numeric	-Precision 10 to 18 changed from DOUBLE PRECISION in SQL dialect 1 to 64-bit scaled integer in SQL dialect 3. Точность от 10 до 18 в диалекте SQL 1 изменена для DOUBLE PRECISION до 64-битового масштабируемого целого в диалекте SQL 3
-104	336003077	sql_db_dialect_dtype_unsupport	-Database SQL dialect <number> does not support reference to <string> datatype. База данных диалекта SQL <номер> не поддерживает ссылку на тип данных <строка>
-104	336003087	dsql_invalid_label	-Label <string> <string> in the current scope. Метка <строка> <строка> находится в текущей области видимости
-104	336003088	dsql_datatypes_not_comparable	-Datatypes <string>are not comparable in expression <string>. Тип данных <строка> не сравним в выражении <строка>
-105	335544702	like_escape_invalid	-Invalid ESCAPE sequence. Неверная ESCAPE-последовательность

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-105	335544789	extract_input_mismatch	-Specified EXTRACT part does not exist in input datatype. Заданная часть EXTRACT не существует во входном типе данных
-150	335544360	read_only_rel	-Attempted update of read-only table. Попытка изменить таблицу только для чтения
-150	335544362	read_only_view	-Cannot update read-only view <string>. Невозможно изменить представление только для чтения <строка>
-150	335544446	non_updatable	-Not updatable. Не изменяется
-150	335544546	constaint_on_view	-Cannot define constraints on views. Нельзя определить ограничения для представления
-151	335544359	read_only_field	-Attempted update of read-only column. Попытка изменить столбец только для чтения
-155	335544658	dsql_base_table	-<string> is not a valid base table of the specified view. <строка> не является допустимой базовой таблицей для указанного представления
-157	335544598	specify_field_err	-Must specify column name for view select expression. Требуется задать имя столбца для выражения SELECT представления

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-158	335544599	num_field_err	-Number of columns does not match select list. Номера столбцов не соответствуют списку SELECT
-162	335544685	no_dbkey	-Dbkey not available for multi-table views. Ключ базы данных недоступен для многотабличных представлений
-170	335544512	prcmismat	-Parameter mismatch for procedure <string>. Несоответствие параметра для процедуры <строка>
-170	335544619	extern_func_err	-External functions cannot have more than 10 parameters. Внешняя функция не может иметь более 10 параметров
-171	335544439	funmismat	-Function <string> could not be matched. Функции <строка> нельзя найти соответствие
-171	335544458	invalid_dimension	-Column not array or invalid dimensions (expected <number>, encountered <number>). Столбец не является массивом или неверная размерность (ожидается <номер>, встретилось <номер>)
-171	335544618	return_mode_err	-Return mode by value not allowed for this data type. Вариант возвращаемого значения недоступен для этого типа данных
-172	335544438	funnotdef	-Function <string> is not defined. Функция <строка> не определена

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-203	335544708	dyn_fld_ambiguous	-Ambiguous column reference. Неоднозначная ссылка на столбец
-204	335544463	gennotdef	-Generator <string> is not defined. Генератор <строка> не определен
-204	335544502	stream_not_defined	-Reference to invalid stream number. Ссылка на неверный номер потока
-204	335544509	charset_not_found	-CHARACTER SET <string> is not defined. Набор символов <строка> не определен
-204	335544511	prcnotdef	-Procedure <string> is not defined. Процедура <строка> не определена
-204	335544515	codnotdef	-Status code <string> unknown. Неизвестный код состояния <строка>
-204	335544516	xcpnotdef	-Exception <string> not defined. Не определено исключение <строка>
-204	335544532	ref_cnstrnt_notfound	-Name of Referential Constraint not defined in constraints table. Имя ссылочного ограничения не определено в таблице ограничений
-204	335544551	grant_obj_notfound	-Could not find table/procedure for GRANT. Невозможно найти таблицу/процедуру для GRANT

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-204	335544568	text_subtype	-Implementation of text subtype <number> not located. Реализация текстового подтипа <номер> не обнаружена
-204	335544573	dsql_datatype_err	-Data type unknown. Неизвестный тип данных
-204	335544580	dsql_relation_err	-Table unknown. Неизвестная таблица
-204	335544581	dsql_procedure_err	-Procedure unknown. Неизвестная процедура
-204	335544588	collation_not_found	-COLLATION <string> is not defined. Порядок сортировки <строка> не определен
-204	335544589	collation_not_for_charset	-COLLATION <string> is not valid for specified CHARACTER SET. Порядок сортировки <строка> неверен для указанного набора символов
-204	335544595	dsql_trigger_err	-Trigger unknown. Неизвестный триггер
-204	335544620	alias_conflict_err	-Alias <string> conflicts with an alias in the same statement. Алиас <строка> конфликтует с алиасом в том же операторе
-204	335544621	procedure_conflict_error	-Alias <string> conflicts with a procedure in the same statement. Алиас <строка> конфликтует с процедурой в том же операторе

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-204	335544622	relation_conflict_err	-Alias <string> conflicts with a table in the same statement. Алиас <строка> конфликтует с таблицей в том же операторе
-204	335544635	dsql_no_relation_alias	-There is no alias or table named <string> at this scope level. Не существует указанного алиаса или таблицы <строка> на этом уровне видимости
-204	335544636	indexname	-There is no index <string> for table <string>. Не существует индекса <строка> для таблицы <строка>
-204	335544640	collation_requires_text	-Invalid use of CHARACTER SET or COLLATE. Неверное использование набора символов или порядка сортировки
-204	335544662	dsql_blob_type_unknown	-BLOB SUB_TYPE <string> is not defined. Подтип BLOB <строка> не определен
-204	335544759	bad_default_value	-Can not define a not null column with NULL as default value. Невозможно определить непустой столбец (NOT NULL) вместе со значением по умолчанию NULL
-204	335544760	invalid_clause	-Invalid clause— '<string>'. Неверное предложение — '<строка>'

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-204	335544800	too_many_contexts	-Too many Contexts of Relation/Procedure/Views. Maximum allowed is 127. Слишком большой контекст в отношении/процедуре/представлении. Допустимо максимум 127
-204	335544817	bad_limit_param	-Invalid parameter to FIRST. Only integers >= 0 are allowed. Неверный параметр для FIRST. Допустимы только целые >= 0
-204	335544818	bad_skip_param	-Invalid parameter to SKIP. Only integers >= 0 are allowed. Неверный параметр для SKIP. Допустимы только целые >= 0
-204	336003085	dsql_ambiguous_field_name	-Ambiguous field name between <string> and <string>. Двусмысленность имени поля между <строка> и <строка>
-205	335544396	fldnotdef	-Column <string> is not defined in table <string>. Столбец <строка> не определен в таблице <строка>
-205	335544552	grant_fld_notfound	-Could not find column for GRANT. Невозможно найти столбец для GRANT
-206	335544578	dsql_field_err	-Column unknown. Неизвестный столбец

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-206	335544587	dsql_blob_err	-Column is not a BLOB. Столбец не является BLOB
-206	335544596	dsql_subselect_err	-Subselect illegal in this context. Вложенный оператор SELECT неверен в данном контексте
-208	335544617	order_by_err	-Invalid ORDER BY clause. Неверное предложение ORDER BY
-219	335544395	relnotdef	-Table <string> is not defined. Таблица <строка> не определена
-239	335544691	cache_too_small	-Insufficient memory to allocate page buffer cache. Недостаточно памяти для выделения кэша под буфер страницы
-260	335544690	cache_redef	-Cache redefined. Переопределение кэша
-281	335544637	no_stream_plan	-Table <string> is not referenced in plan. Таблица <строка> не указана в плане
-282	335544638	stream_twice	-Table <string> is referenced more than once in plan; use aliases to distinguish. На таблицу <строка> осуществляются ссылки более одного раза в плане; используйте алиасы для различения

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-282	335544643	dsql_self_join	-The table <string> is referenced twice; use aliases to differentiate. На таблицу <строка> в плане осуществляются ссылки дважды; используйте алиасы для разыменования
-282	335544659	duplicate_base_table	-Table <string> is referenced twice in view; use an alias to distinguish. На таблицу <строка> в представлении осуществляются ссылки дважды; используйте алиас для различения
-282	335544660	view_alias	-View <string> has more than one base table; use aliases to distinguish. Представление <строка> имеет более одной базовой таблицы; используйте алиасы для различения
-282	335544710	complex_view	-Navigational stream <number> references a view with more than one base table. Поток навигации <строка> ссылается на представление с более чем одной базовой таблицей
-283	335544639	stream_not_found	-Table <string> is referenced in the plan but not the from list. На таблицу <строка> есть ссылки в плане, однако она не указана в списке

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-284	335544642	-index_unused	-Index <string> cannot be used in the specified plan. Индекс <строка> не может быть использован в указанном плане
-291	335544531	primary_key_notnull	-Column used in a PRIMARY KEY constraint must be NOT NULL. Столбец, используемый в ограничении первичного ключа, должен быть NOT NULL
-292	335544534	ref_cnstrnt_update	-Cannot update constraints (RDB$REF_CONSTRAINTS). Нельзя изменять ограничения (RDB$REF_CONSTRAINTS)
-293	335544535	check_cnstrnt_update	-Cannot update constraints (RDB$CHECK_CONSTRAINTS). Нельзя изменять ограничения (RDB$REF_CONSTRAINTS)
-294	335544536	check_cnstrnt_del	-Cannot delete CHECK constraint entry (RDB$CHECK_CONSTRAINTS) Нельзя удалять запись ограничения CHECK (RDB$REF_CONSTRAINTS)
-295	335544545	rel_cnstrnt_update	-Cannot update constraints (RDB$RELATION_CONSTRAINTS). Нельзя изменять ограничения (RDB$RELATION_CONSTRAINTS)

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-296	335544547	invld_cnstrnt_type	-Internal gds software consistency check (invalid RDB$CONSTRAINT_TYPE). Проверка внутреннего программного обеспечения gds (неверный RDB$CONSTRAINT_YYPE)
-297	335544558	check_constraint	-Operation violates CHECK constraint <string> on view or table <string> Операция нарушает ограничение CHECK <строка> для представления или таблицы <строка>
-313	335544669	dsql_count_mismatch	-Count of column list and variable list do not match. Количество столбцов в списке и список переменных не соответствуют друг другу
-314	335544565	transliteration_failed	-Cannot transliterate character between character sets. Невозможно выполнить транслитерацию символов между наборами символов
-315	336068815	dyn_dtype_invalid	-Cannot change datatype for column <string>. Changing datatype is not supported for BLOB or ARRAY columns. Невозможно изменить тип данных для столбца <строка>. Изменение типа данных не поддерживается для столбцов BLOB и массивов

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-383	336068814	dyn_dependency_exists	-Column <string> from table <string> is referenced in <string>. На столбец <строка> из таблицы <строка> есть ссылка в <строка>
-401	335544647	invalid_operator	-Invalid comparison operator for find operation. Неверный оператор сравнения для операции
-402	335544368	segstr_no_op	-Attempted invalid operation on a BLOB. Попытка использования неверной операции для BLOB
-402	335544414	blobnotsup	-BLOB and array data types are not supported for <string> operation. Типы данных BLOB и массивы не поддерживаются для операции <строка>
-402	335544427	datnotsup	-Data operation not supported. Операция с данными не поддерживается
-406	335544457	out_of_bounds	-Subscript out of bounds. Выход за пределы диапазона
-407	335544435	nullsegkey	-Null segment of UNIQUE KEY. Пустой сегмент для уникального ключа

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-413	335544334	convert_error	-Conversion error from string "<string>". Ошибка преобразования для строки "<строка>"
-413	335544454	nofilter	-Filter not found to convert type <number> to type <number>. Не найден фильтр для преобразования типа <номер> в тип <номер>
-501	335544327	bad_req_handle	-Invalid request handle. Неверный запрос дескриптора
-501	335544577	dsql_cursor_close_err	-Attempt to reclose a closed cursor. Попытка повторного закрытия закрытого курсора
-502	-335544574	-dsql_decl_err	-Declared cursor already exists. Объявляемый курсор уже существует
-502	-335544576	-dsql_cursor_open_err	-Attempt to reopen an open cursor. Попытка повторного открытия открытого курсора
-504	-335544572	-dsql_cursor_err	-Cursor unknown. Неизвестный курсор
-508	-335544348	-no_cur_rec	-No current record for fetch operation. Нет текущей записи для операции загрузки
-510	-335544575	-dsql_cursor_update_err	-Cursor not updatable. Курсор не является изменяемым
-518	-335544582	-dsql_request_err	-Request unknown. Неизвестный запрос

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-519	-335544688	-dsql_open_cursor_request	-The prepare statement identifies a prepare statement with an open cursor. Оператор подготовки идентифицировал оператор подготовки для открытого курсора
-530	-335544466	-foreign_key	-Violation of FOREIGN KEY constraint "<string>" on table "<string>". Нарушение ограничения внешнего ключа "<строка>" для таблицы "<строка>"
-531	-335544597	-dsql_crdb_prepare_err	-Cannot prepare a CREATE DATABASE/SCHEMA statement. Невозможно подготовить оператор CREATE DATABASE / SCHEMA
-532	-335544469	-trans_invalid	-Transaction marked invalid by I/O error. Транзакция отмечена как ошибочная из-за ошибки ввода-вывода
-551	-335544352	-no_priv	-No permission for <string> access to <string> <string>. Не существует полномочий для доступа <строка> к <строка>
-551	-335544790	-insufficient_svc_privileges	-Service <string> requires SYSDBA permissions. Reattach to the Service Manager using the SYSDBA account. Сервис <строка> требует полномочий SYSDBA. Соединитесь с Менеджером Сервисов как пользователь SYSDBA

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-552	-335544550	-not_rel_owner	-Only the owner of a table may reassign ownership. Только владелец таблицы может переназначать владение
-552	-335544553	-grant_nopriv	-User does not have GRANT privileges for operation . Пользователь не имеет назначенных привилегий для операции
-552	-335544707	-grant_nopriv_on_base	-User does not have GRANT privileges on base table/view for operation. Пользователь не имеет назначенных привилегий к базовой таблице/представлению для операции
-553	-335544529	-existing_priv_mod	-Cannot modify an existing user privilege. Невозможно изменить существующую привилегию пользователя
-595	-335544645	-stream_crack	-The current position is on a crack. Текущая позиция разрушена
-596	-335544644	-stream_bof	-Illegal operation when at beginning of stream. Неверная операция при начале потока
-597	-335544632	-dsql_file_length_err	-Preceding file did not specify length, so <string> must include starting page number. Предыдущий файл не содержит длины, следовательно, <строка> должен включать начальный номер страницы

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-598	-335544633	-dsql_shadow_number_err	-Shadow number must be a positive integer. Номер оперативной копии должен быть положительным целым числом
-599	-335544607	-node_err	-Gen.c: node not supported. Узел Gen.c не поддерживается
-599	-335544625	-node_name_err	-A node name is not permitted in a secondary, shadow, cache or log file name. Имя узла не разрешено во вторичном файле, файле оперативной копии, кэше или файле протокола
-600	-335544680	-crrp_data_err	-Sort error: corruption in data structure. Ошибка сортировки: разрушение структуры данных
-601	-335544646	-db_or_file_exists	-Database or file exists. Существует база данных или файл
-604	-335544593	-dsql_max_arr_dim_exceeded	-Array declared with too many dimensions. Объявлен массив с слишком большой размерностью
-604	-335544594	-dsql_arr_range_error	-Illegal array dimension range. Неверный диапазон размерности массива
-605	-335544682	-dsql_field_ref	-Inappropriate self-reference of column. Несоответствующая ссылка столбца на самого себя

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-607	-335544351	-no_meta_update	-Unsuccessful metadata update. Ошибочное изменение метаданных
-607	-335544549	-systrig_update	-Cannot modify or erase a system trigger. Невозможно изменить или удалить системный триггер
-607	-335544657	-dsql_no_blob_array	-Array/BLOB/DATE data types not allowed in arithmetic. Типы данных массив/BLOB недопустимы в арифметических операциях
-607	-335544746	-reftable_requires_pk	-"REFERENCES table" without "(column)" requires PRIMARY KEY on referenced table. "REFERENCES таблица" без указания столбца требует первичного ключа в таблицу, на которую осуществляется ссылка
-607	-335544815	-generator_name	-GENERATOR <string>. Генератор <строка>
-607	-335544816	-udf_name	-UDF <string>. UDF <строка>
-607	-336003074	-dsql_dbkey_from_non_table	-Cannot SELECT RDB$DB_KEY from a stored procedure. Невозможно выполнить SELECT RDB$DB_KEY из хранимой процедуры

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-607	-336003086	-dsql_udf_return_pos_err	-External function should have return position between 1 and <number>. Внешняя функция должна иметь позицию возвращаемого значения между 1 и <номер>
-612	-336068812	-dyn_domain_name_exists	-Cannot rename domain <string> to <string>. A domain with that name already exists. Невозможно переименовать домен <строка> в <строка>. Домен с этим именем уже существует
-612	-336068813	-dyn_field_name_exists	-Cannot rename column <string> to <string>. A column with that name already exists in table <string>. Невозможно переименовать столбец <строка> в <строка>. Столбец с этим именем уже существует в таблице <строка>
-615	-335544475	-relation_lock	-Lock on table <string> conflicts with existing lock. Блокировка таблицы <строка> конфликтует с существующей блокировкой
-615	-335544476	-record_lock	-Requested record lock conflicts with existing lock. Запрашиваемая блокировка записи конфликтует с существующей блокировкой

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-615	-335544507	`-range_in_use`	`-Refresh range number <number> already in use.` Обновляемый диапазон <номер> уже находится в использовании
-616	-335544530	`-primary_key_ref`	`-Cannot delete PRIMARY KEY being used in FOREIGN KEY definition.` Невозможно удалить первичный ключ, используемый в определении внешнего ключа
-616	-335544539	`-integ_index_del`	`-Cannot delete index used by an Integrity Constraint.` Невозможно удалить индекс, используемый в ограничении целостности
-616	-335544540	`-integ_index_mod`	`-Cannot modify index used by an Integrity Constraint.` Невозможно изменить индекс, используемый в ограничении целостности
-616	-335544541	`-check_trig_del`	`-Cannot delete trigger used by a CHECK Constraint.` Невозможно удалить триггер, используемый в ограничении `CHECK`
-616	-335544543	`-cnstrnt_fld_del`	`-Cannot delete column being used in an Integrity Constraint.` Невозможно удалить столбец, используемый в ограничении целостности
-616	-335544630	`-dependency`	`-There are <number> dependencies.` Существуют зависимости <номер>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-616	-335544674	-del_last_field	-Last column in a table cannot be deleted. Последний столбец в таблице не может быть удален
-616	-335544728	-integ_index_deactivate	-Cannot deactivate index used by an Integrity Constraint. Невозможно деактивировать триггер, используемый в ограничении целостности
-616	-335544729	-integ_deactivate_primary	-Cannot deactivate primary index. Невозможно деактивировать первичный индекс
-617	-335544542	-check_trig_update	-Cannot update trigger used by a CHECK Constraint. Невозможно обновить триггер, используемый в ограничении CHECK
-617	-335544544	-cnstrnt_fld_rename	-Cannot rename column being used in an Integrity Constraint. Невозможно переименовать столбец, используемый в ограничении целостности
-618	-335544537	-integ_index_seg_del	-Cannot delete index segment used by an Integrity Constraint. Невозможно удалить сегмент индекса, используемый в ограничении целостности

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-618	-335544538	-integ_index_seg_mod	-Cannot update index segment used by an Integrity Constraint. Невозможно изменить сегмент индекса, используемый в ограничении целостности
-625	-335544347	-not_valid	-Validation error for column <string>, value "<string>". Ошибка проверки для столбца <строка>, значение "<строка>"
-637	-335544664	-dsql_duplicate_spec	-Duplicate specification of <string>- not supported. Дублирование спецификации для <строка> не поддерживается
-660	-335544533	-foreign_key_notfound	-Non-existent PRIMARY or UNIQUE KEY specified for FOREIGN KEY. Не существует первичный или уникальный ключ, указанный для внешнего ключа
-660	-335544628	-idx_create_err	-Cannot create index <string>. Невозможно создать индекс <строка>
-663	-335544624	-idx_seg_err	-Segment count of 0 defined for index <string>. Счетчик сегментов 0 определен для индекса <строка>
-663	-335544631	-idx_key_err	-Too many keys defined for index <string>. Слишком много ключей определено для индекса <строка>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-663	-335544672	-key_field_err	-Too few key columns found for index <string> (incorrect column name?). Слишком мало ключевых столбцов найдено для индекса <строка> (неверное имя столбца?)
-664	-335544434	-keytoobig	-Key size exceeds implementation restriction for index "<string>". Размер ключа превышает ограничения реализации для индекса "<строка>"
-677	-335544445	-ext_err	-<string> extension error/ Ошибка расширения <строка>
-685	-335544465	-bad_segstr_type	-Invalid BLOB type for operation. Неверный тип BLOB для операции
-685	-335544670	-blob_idx_err	-Attempt to index BLOB column in index <string>. Попытка включить столбец BLOB в индекс <строка>
-685	-335544671	-array_idx_err	-Attempt to index array column in index <string>. Попытка включить столбец массива в индекс <строка>
-689	-335544403	-badpagtyp	-Page <number> is of wrong type (expected <number>, found <number>). Страница <номер> имеет неверный тип (ожидается <номер>, найдено <номер>)

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-689	-335544650	-page_type_err	-Wrong page type. Неверный тип страницы
-690	-335544679	-no_segments_err	-Segments not allowed in expression index <string>. Сегменты недоступны в индексном выражении <строка>
-691	-335544681	-rec_size_err	-New record size of <number> bytes is too big. Новая запись размера <число> байтов слишком велика
-692	-335544477	-max_idx	-Maximum indexes per table (<number>) exceeded. Превышено максимальное количество индексов для таблицы (<число>)
-693	-335544663	-req_max_clones_exceeded	-Too many concurrent executions of the same request. Слишком много выполнений того же самого запроса
-694	-335544684	-no_field_access	-Cannot access column <string> in view <string>. Невозможен доступ к столбцу <строка> в представлении <строка>
-802	-335544321	-arith_except	-Arithmetic exception, numeric overflow, or string truncation. Арифметическое исключение, числовое переполнение или усечение строки

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-803	-335544349	-no_dup	-Attempt to store duplicate value (visible to active transactions) in unique index "<string>". Попытка сохранения дубликата значения (видимого в активной транзакции) в уникальном индексе <строка>
-803	-335544665	-unique_key_violation	-Violation of PRIMARY or UNIQUE KEY constraint "<string>" on table "<string>". Нарушение ограничения "<строка>" для первичного или уникального ключа для таблицы "<строка>"
-804	-335544380	-wronumarg	-Wrong number of arguments on call. Неверное количество аргументов при вызове
-804	-335544583	-dsql_sqlda_err	-SQLDA missing or incorrect version, or incorrect number/type of variables. SQLDA отсутствует или имеет неверную версию или неверное количество или тип переменных
-804	-335544586	-dsql_function_err	-Function unknown. Неизвестная функция
-804	-335544713	-dsql_sqlda_value_err	-Incorrect values within SQLDA structure. Неверные значения в структуре SQLDA

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-806	-335544600	`-col_name_err`	`-Only simple column names permitted for VIEW WITH CHECK OPTION.` Только простые имена столбцов допустимы в VIEW WITH CHECK OPTION
-807	-335544601	`-where_err`	`-No WHERE clause for VIEW WITH CHECK OPTION.` Нет предложения `WHERE` для VIEW WITH CHECK OPTION
-808	-335544602	`-table_view_err`	`-Only one table allowed for VIEW WITH CHECK OPTION.` Только одна таблица допустима для VIEW WITH CHECK OPTION
-809	-335544603	`-distinct_err`	`-DISTINCT, GROUP or HAVING not permitted for VIEW WITH CHECK OPTION.` Не разрешены `DISTINCT`, `GROUP` или `HAVING` в VIEW WITH CHECK OPTION
-810	-335544605	`-subquery_err`	`-No subqueries permitted for VIEW WITH CHECK OPTION.` Не позволены подзапросы для VIEW WITH CHECK OPTION
-811	-335544652	`-sing_select_err`	`-Multiple rows in singleton select.` Множество строк в одиночном `SELECT`

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-816	-335544651	`-ext_readonly_err`	`-Cannot insert because the file is readonly or is on a read only medium.` Невозможно добавление, потому что файл является файлом только для чтения или располагается на устройстве только для чтения
-816	-335544715	`-extfile_uns_op`	`-Operation not supported for EXTERNAL FILE table <string>.` Операция не поддерживается для таблицы внешнего файла <строка>
-817	-335544361	`-read_only_trans`	`-Attempted update during read-only transaction.` Попытка изменения при транзакции только для чтения
-817	-335544371	`-segstr_no_write`	`-Attempted write to read-only BLOB.` Попытка записи в `BLOB` только для чтения
-817	-335544444	`-read_only`	`-Operation not supported.` Операция не поддерживается
-817	-335544765	`-read_only_database`	`-Attempted update on read-only database.` Попытка изменения базы данных только для чтения
-817	-335544766	`-must_be_dialect_2_and_up`	`-SQL dialect <string> is not supported in this database.` Диалект SQL <строка> не поддерживается в этой базе данных

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-817	-335544793	-ddl_not_allowed_by_db_sql_dial	-Metadata update statement is not allowed by the current database SQL dialect <number>. Оператор изменения метаданных недопустим в текущем диалекте SQL базы данных <строка>
-817	-336003079	-isc_sql_dialect_conflict_num	-DB dialect <number> and client dialect <number> conflict with respect to numeric precision <number> Диалект базы данных <число> и диалект клиента <число> конфликтуют в отношении точности чисел <строка>
-820	-335544356	-obsolete_metadata	-Metadata is obsolete. Устаревшие метаданные
-820	-335544379	-wrong_ods	-Unsupported on-disk structure for file <string>; found <number>, support <number>. Неподдерживаемая структура на диске (ODS) для файла <строка>; найдено <номер>, поддерживается <номер>
-820	-335544437	-wrodynver	- Wrong DYN version. Неверная версия DYN
-820	-335544467	-high_minor	-Minor version too high found <number> expected <number>. Найдена слишком высокая минимальная версия <номер>, ожидается <номер>
-823	-335544473	-invalid_bookmark	-Invalid bookmark handle. Неверный дескриптор закладки

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-824	-335544474	-bad_lock_level	-Invalid lock level <number>. Неверный уровень блокировки <номер>
-825	-335544519	-bad_lock_handle	-Invalid lock handle. Неверный дескриптор блокировки
-826	-335544585	-dsql_stmt_handle	-Invalid statement handle. Неверный дескриптор оператора
-827	-335544655	-invalid_direction	-Invalid direction for find operation. Неверное направление для операции поиска
-827	-335544718	-invalid_key	-Invalid key for find operation. Неверный ключ для операции поиска
-828	-335544678	-inval_key_posn	-Invalid key position. Неверная позиция ключа
-829	-335544616	-field_ref_err	-Invalid column reference. Неверная ссылка на столбец
-829	-336068816	-dyn_char_fld_too_small	-New size specified for column <string> must be at least <number> characters. Указанный новый размер для столбца <строка> должен иметь, по меньшей мере, <номер> символов

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-829	-336068817	-dyn_invalid_dtype_conversion	-Cannot change datatype for <string>. Conversion from base type <string> to <string> is not supported. Невозможно изменить тип данных для <строка>. Преобразование из базового типа <строка> в <строка> не поддерживается
-829	-336068818	-dyn_dtype_conv_invalid	-Cannot change datatype for column <string> from a character type to a non-character type. Невозможно изменить тип данных для столбца <строка> из набора символов в тип, не имеющий набора символов
-830	-335544615	-field_aggregate_err	-Column used with aggregate. Столбец используется в агрегате
-831	-335544548	-primary_key_exists	-Attempt to define a second PRIMARY KEY for the same table. Попытка определения второго первичного ключа для той же таблицы
-832	-335544604	-key_field_count_err	-FOREIGN KEY column count does not match PRIMARY KEY. Количество столбцов внешнего ключа не соответствует первичному ключу
-833	-335544606	-expression_eval_err	-Expression evaluation not supported. Вычисление выражения не поддерживается

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-833	-335544810	-date_range_exceeded	-Value exceeds the range for valid dates. Значение превышает диапазон допустимых дат
-834	-335544508	-range_not_found	-Refresh range number <number> not found. Номер диапазона обновления <номер> не найден
-835	-335544649	-bad_checksum	-Bad checksum. Ошибочная контрольная сумма
-836	-335544517	-except	-Exception <number>. Исключение <номер>
-837	-335544518	-cache_restart	-Restart shared cache manager. Повторный запуск менеджера совместно используемого кэша
-838	-335544560	-shutwarn	-Database <string> shutdown in <number> seconds. База данных <строка> остановлена на <номер> секунд
-841	-335544677	-version_err	-Too many versions. Слишком много версий
-842	-335544697	-precision_err	-Precision must be from 1 to 18. Точность должна быть между 1 и 18
-842	-335544698	-scale_nogt	-Scale must be between zero and precision. Масштаб должен быть между нулем и точностью

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-842	-335544699	-expec_short	-Short integer expected. Ожидается короткое целое
-842	-335544700	-expec_long	-Long integer expected. Ожидается длинное целое
-842	-335544701	-expec_ushort	-Unsigned short integer expected. Ожидается беззнаковое короткое целое
-842	-335544712	-expec_positive	-Positive value expected. Ожидается положительное значение
-901	-335544322	-bad_dbkey	-Invalid database key. Неверный ключ базы данных
-901	-335544326	-bad_dpb_form	-Unrecognized database parameter block. Нераспознанный блок параметров базы данных
-901	-335544328	-bad_segstr_handle	Invalid BLOB handle. Неверный дескриптор BLOB
-901	-335544329	-bad_segstr_id	Invalid BLOB ID. Неверный идентификатор BLOB
-901	-335544330	-bad_tpb_content	-Invalid parameter in transaction parameter block. Неверный параметр в блоке параметров транзакции
-901	-335544331	-bad_tpb_form	-Invalid format for transaction parameter block. Неверный формат блока параметров транзакции

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544332	`-bad_trans_handle`	`-Invalid transaction handle (expecting explicit transaction start).` Неверный дескриптор транзакции (ожидается явный запуск транзакции)
-901	-335544337	`-excess_trans`	`-Attempt to start more than <number> transactions.` Попытка запуска более чем <номер> транзакций
-901	-335544339	`-infinap`	`-Information type inappropriate for object specified.` Информационный тип не соответствует указанному объекту
-901	-335544340	`-infona`	`-No information of this type available for object specified.` Никакой информационный тип не доступен для указанного объекта
-901	-335544341	`-infunk`	`-Unknown information item.` Неизвестный информационный элемент
-901	-335544342	`-integ_fail`	`-Action cancelled by trigger (<number>) to preserve data integrity.` Отменено действие в триггере (<номер>) для сохранения целостности данных

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544345	-lock_conflict	-Lock conflict on no wait transaction. Конфликт блокировки для транзакции NO WAIT
-901	-335544350	-no_finish	-Program attempted to exit without finishing database. Программа пытается завершиться без закрытия базы данных
-901	-335544353	-no_recon	-Transaction is not in limbo. Транзакция не является зависшей
-901	-335544355	-no_segstr_close	-BLOB was not closed. BLOB не был закрыт
-901	-335544357	-open_trans	-Cannot disconnect database with open transactions (<number> active). Невозможно отключиться от базы данных при наличии открытой транзакции (активная <номер>)
-901	-335544358	-port_len	-Message length error (encountered <number>, expected <number>). Ошибка длины сообщения (встречена <число>, ожидается <число>)
-901	-335544363	-req_no_trans	-No transaction for request. Для запроса нет транзакции
-901	-335544364	-req_sync	-Request synchronization error. Ошибка синхронизации запроса

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544365	-req_wrong_db	-Request referenced an unavailable database. Запрос ссылается на недоступную базу данных
-901	-335544369	-segstr_no_read	-Attempted read of a new, open BLOB. Попытка чтения нового, открытого BLOB
-901	-335544370	-segstr_no_trans	-Attempted action on blob outside transaction. Попытка действий с BLOB за пределами транзакции
-901	-335544372	-segstr_wrong_db	-Attempted reference to BLOB in unavailable database. Попытка ссылки на BLOB в недоступной базе данных
-901	-335544376	-unres_rel	-Table <string> was omitted from the transaction reserving list. Таблица <строка> была опущена в зарезервированном списке транзакции
-901	-335544377	-uns_ext	-Request includes a DSRI extension not supported in this implementation. Запрос включает расширение DSRI, не поддерживаемое в этой реализации
-901	-335544378	-wish_list	-Feature is not supported. Возможность не поддерживается
-901	-335544382	-random	-<string>. <строка>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544383	-fatal_conflict	-Unrecoverable conflict with limbo transaction <number>. Неперекрываемый конфликт с зависшей транзакцией <число>
-901	-335544392	-bdbincon	-Internal error. Внутренняя ошибка
-901	-335544407	-dbbnotzer	-Database handle not zero. Дескриптор базы данных не ноль
-901	-335544408	-tranotzer	-Transaction handle not zero. Дескриптор транзакции не ноль
-901	-335544418	-trainlim	-Transaction in limbo. Зависшая транзакция
-901	-335544419	-notinlim	-Transaction not in limbo. Транзакция не зависшая
-901	-335544420	-traoutsta	-Transaction outstanding. Ожидающая выполнения транзакция
-901	-335544428	-badmsgnum	-Undefined message number. Неопределенный номер сообщения
-901	-335544431	-blocking_signal	-Blocking signal has been received. Был получен сигнал блокировки
-901	-335544442	-noargacc_read	-Database system cannot read argument <number>. Система базы данных не может читать аргумент <число>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544443	-noargacc_write	-Database system cannot write argument <number>. Система базы данных не может писать аргумент <число>
-901	-335544450	-misc_interpreted	-<string>. <строка>
-901	-335544468	-tra_state	-Transaction <number> is <string>. Транзакция <число> является <строка>
-901	-335544485	-bad_stmt_handle	-Invalid statement handle. Неверный дескриптор оператора
-901	-335544510	-lock_timeout	-Lock time-out on wait transaction. Истечение времени ожидания блокировки для транзакции WAIT
-901	-335544559	-bad_svc_handle	-Invalid service handle. Неверный дескриптор сервиса
-901	-335544561	-wrospbver	-Wrong version of service parameter block. Неверная версия блока параметра сервиса
-901	-335544562	-bad_spb_form	-Unrecognized service parameter block. Нераспознанный блок параметров сервиса
-901	-335544563	-svcnotdef	-Service <string> is not defined. Сервис <строка> не определен
-901	-335544609	-index_name	-INDEX <string>. Индекс <строка>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544610	`-exception_name`	`-EXCEPTION <string>.` Исключение <строка>
-901	-335544611	`-field_name`	`-COLUMN <string>.` Столбец <строка>
-901	-335544613	`-union_err`	`-Union not supported.` Объединение не поддерживается
-901	-335544614	`-dsql_construct_err`	`-Unsupported DSQL construct.` Неподдерживаемая конструкция DSQL
-901	-335544623	`-dsql_domain_err`	`-Illegal use of keyword VALUE.` Неверное использование ключевого слова `VALUE`
-901	-335544626	`-table_name`	`-TABLE <string>.` Таблица <строка>
-901	-335544627	`-proc_name`	`-PROCEDURE <string>.` Процедура <строка>
-901	-335544641	`-dsql_domain_not_found`	`-Specified domain or source column <string> does not exist.` Указанный домен или исходный столбец <строка> не существует
-901	-335544656	`-dsql_var_conflict`	`-Variable <string> conflicts with parameter in same procedure.` Переменная <строка> конфликтует с параметром в той же процедуре
-901	-335544666	`-srvr_version_too_old`	`-Server version too old to support all CREATE DATABASE options.` Версия сервера слишком старая для поддержки всех режимов `CREATE DATABASE`

Таблица П3.1 *(продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544673	-no_delete	-Cannot delete. Удаление невозможно
-901	-335544675	-sort_err	-Sort error. Ошибка сортировки
-901	-335544703	-svcnoexe	-Service <string> does not have an associated executable. Сервис <строка> не имеет связанного исполняемого модуля
-901	-335544704	-net_lookup_err	-Failed to locate host machine. Ошибка в локализации хост-машины
-901	-335544705	-service_unknown	-Undefined service <string>/<string>. Не определен сервис <строка>/<строка>
-901	-335544706	-host_unknown	-The specified name was not found in the hosts file or Domain Name Services. Указанное имя не было найдено в файле hosts или в сервисе имен доменов
-901	-335544711	-unprepared_stmt	-Attempt to execute an unprepared dynamic SQL statement. Попытка выполнения неподготовленного оператора динамического SQL
-901	-335544716	-svc_in_use	-Service is currently busy: <string>. В настоящий момент сервис занят

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544740	-udf_exception	-A fatal exception occurred during the execution of a user defined function. Возникло фатальное исключение в процессе выполнения функции, определенной пользователем
-901	-335544741	-lost_db_connection	-Connection lost to database. Потеряно соединение с базой данных
-901	-335544742	-no_write_user_priv	-User cannot write to RDB$USER_PRIVILEGES. Пользователь не может писать в RDB$USER_PRIVILEGES
-901	-335544767	-blob_filter_exception	-A fatal exception occurred during the execution of a blob filter. Возникло фатальное исключение в процессе выполнения фильтра BLOB
-901	-335544768	-exception_access_violation	-Access violation. The code attempted to access a virtual address without privilege to do so. Нарушение доступа. Код пытается получить доступ к виртуальному адресу без соответствующих привилегий

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544769	—	-Datatype misalignment. The attempted to read or write a value that was not stored on a memory boundary. Неверное выравнивание типа данных. Попытка читать или писать значение, которое не было сохранено в нужных границах памяти
-901	-335544770	—	-Array bounds exceeded. The code attempted to access an array element that is out of bounds. Превышены размеры массива. Код пытается получить доступ к элементу массива, который находится за пределами его границ
-901	-335544771	—	-Float denormal operand. One of the floating-point operands is too small to represent a standard float value. Ненормализованный операнд для числа с плавающей точкой. Один из операндов, ссылающихся на число с плавающей точкой, слишком мал для представления стандартного значения с плавающей точкой
-901	-335544772	-exception_float_divide_by_zero	-Floating-point divide by zero. The code attempted to divide a floating-point value by zero. Деление числа с плавающей точкой на ноль. Код пытается разделить значение с плавающей точкой на ноль

Таблица П3.1 *(продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544773	`-exception_float_inexact_result`	`-Floating-point inexact result. The result of a floating-point operation cannot be represented as a decimal fraction.` Неточный результат для числа с плавающей точкой. Результат операции для чисел с плавающей точкой не может быть представлен в виде десятичной дробной части
-901	-335544774	`-exception_float_invalid_operand`	`-Floating-point invalid operand. An indeterminant error occurred during a floating-point operation.` Неверный операнд в операции с плавающей точкой. Неопределенная ошибка возникает в процессе операции с числами с плавающей точкой
-901	-335544775	`-exception_float_overflow`	`-Floating-point overflow. The exponent of a floating-point operation is greater than the magnitude allowed.` Переполнение числа с плавающей точкой. Экспонента операции с числами с плавающей точкой больше, чем доступные размеры

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544776	-exception_float_stack_check	-Floating-point stack check. The stack overflowed or underflowed as the result of a floating-point operation. Проверка стека чисел с плавающей точкой. Переполнение стека или потеря значащих разрядов является результатом операции с числами с плавающей точкой
-901	-335544777	-exception_float_underflow	-Floating-point underflow. The exponent of a floating-point operation is less than the magnitude allowed. Потеря значащих разрядов числа с плавающей точкой. Экспонента операции с плавающей точкой меньше, чем допустимый размер
-901	-335544778	—	-Integer divide by zero. The code attempted to divide an integer value by an integer divisor of zero. Деление целого на ноль. Код пытается разделить целое значение на целый делитель, который является нулем
-901	-335544779	-exception_integer_overflow	-Integer overflow. The result of an integer operation caused the most significant bit of the result to carry. Целочисленное переполнение. Результат операции над целыми числами дает больше знаков, чем может храниться

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544780	-exception_unknown	-An exception occurred that does not have a description. Exception number %X. Появилось исключение, не имеющее дескриптора. Исключение с номером %X
-901	-335544781	-exception_stack_overflow	-Stack overflow. The resource requirements of the runtime stack have exceeded the memory available to it. Переполнение стека. Требуемые для стека ресурсы во время выполнения исчерпали доступную память
-901	-335544782	-exception_sigsegv	-Segmentation Fault. The code attempted to access memory without privileges. Ошибка сегментирования. Код пытается получить доступ к памяти без привилегий
-901	-335544783	-exception_sigill	-Illegal Instruction. The Code attempted to perform an illegal operation. Неверная операция. Код пытается выполнить неверную операцию
-901	-335544784	-exception_sigbus	-Bus Error. The Code caused a system bus error. Ошибка канала. Выполнение кода приводит к системной ошибке канала

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544785	-exception_sigfpe	-Floating Point Error. The Code caused an Arithmetic Exception or a floating point exception. Ошибка десятичной точки. Код выдает арифметическое исключение или исключение десятичной точки
-901	-335544786	-ext_file_delete	-Cannot delete rows from external files. Невозможно удалить строки из внешних файлов
-901	-335544787	-ext_file_modify	-Cannot update rows in external files. Невозможно изменить строки во внешних файлах
-901	-335544788	-adm_task_denied	-Unable to perform operation. You must be either SYSDBA or owner of the database. Невозможно выполнить операцию. Вы должны быть или пользователем SYSDBA, либо владельцем базы данных
-901	-335544794	-cancelled	-Operation was cancelled. Операция была отменена
-901	-335544797	-svcnouser	-User name and password are required while attaching to the services manager. Требуются имя пользователя и пароль при подключении к Менеджеру Сервисов

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335544801	-datype_notsup	-Data type not supported for arithmetic. Тип данных не поддерживается для арифметических действий
-901	-335544803	-dialect_not_changed	-Database dialect not changed. Диалект базы данных не был изменен
-901	-335544804	-database_create_failed	-Unable to create database <string>. Невозможно создать базу данных <строка>
-901	-335544805	-inv_dialect_specified	-Database dialect <number> is not a valid dialect. Диалект базы данных <число> не является правильным диалектом
-901	-335544806	-valid_db_dialects	-Valid database dialects are <string>. Допустимыми диалектами базы данных являются <строка>
-901	-335544811	-inv_client_dialect_specified	-Passed client dialect <number> is not a valid dialect. Переданный клиентом диалект <число> не является верным диалектом
-901	-335544812	-valid_client_dialects	-Valid client dialects are <string>. Допустимыми клиентскими диалектами являются <строка>
-901	-335544814	-service_not_supported	-Services functionality will be supported in a later version of the product Функциональность сервисов будет поддерживаться на более поздних версиях продукта

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335740929	`-gfix_db_name`	`-Data base file name (<string>) already given.` Имя файла базы данных (<строка>) уже предоставлено
-901	-335740930	`-gfix_invalid_sw`	`-Invalid switch <string>.` Неверный переключатель <строка>
-901	-335740932	`-gfix_incmp_sw`	`-Incompatible switch combination.` Несовместимая комбинация переключателей
-901	-335740933	`-gfix_replay_req`	`-Replay log pathname required.` Требуется путь к протоколу
-901	-335740934	`-gfix_pgbuf_req`	`-Number of page buffers for cache required.` Требуется буфер страниц для кэша
-901	-335740935	`-gfix_val_req`	`-Numeric value required.` Требуется числовое значение
-901	-335740936	`-gfix_pval_req`	`-Positive numeric value required.` Требуется положительное числовое значение
-901	-335740937	`-gfix_trn_req`	`-Number of transactions per sweep required.` Требуется номер транзакции для чистки
-901	-335740940	`-gfix_full_req`	`-"full" or "reserve" required` Требуются "full" или "reserve"

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335740941	-gfix_usrname_req	-User name required. Требуется имя пользователя
-901	-335740942	-gfix_pass_req	-Password required. Требуется пароль
-901	-335740943	-gfix_subs_name	-Subsystem name. Имя подсистемы
-901	-335740945	-gfix_sec_req	-Number of seconds required. Требуется число или секунды
-901	-335740946	-gfix_nval_req	-Numeric value between 0 and 32767 inclusive required. Требуется числовое значение между 9 и 32 767, включительно
-901	-335740947	-gfix_type_shut	-Must specify type of shutdown. Должен быть указан тип останова
-901	-335740948	-gfix_retry	-Please retry, specifying an option. Пожалуйста, повторите, указав режим
-901	-335740951	-gfix_retry_db	-Please retry, giving a database name. Пожалуйста, повторите, задав имя базы данных
-901	-335740991	-gfix_exceed_max	-Internal block exceeds maximum size. Внутренний блок превышает максимальный размер
-901	-335740992	-gfix_corrupt_pool	-Corrupt pool. Разрушен пул

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-335740993	-gfix_mem_exhausted	-Virtual memory exhausted. Исчерпана виртуальная память
-901	-335740994	-gfix_bad_pool	-Bad pool id. Неверный идентификатор пула
-901	-335740995	-gfix_trn_not_valid	-Transaction state <number> not in valid range. Состояние транзакции <строка> не находится в допустимом диапазоне
-901	-335741012	-gfix_unexp_eoi	-Unexpected end of input. Неверное завершение ввода
-901	-335741018	-gfix_recon_fail	-Failed to reconnect to a transaction in database <string>. Ошибки при повторном соединении с транзакцией в базе данных <строка>
-901	-335741036	-gfix_trn_unknown	-Transaction description item unknown. Неизвестное описание элемента транзакции
-901	-335741038	-gfix_mode_req	-"read_only" or "read_write" required. Требуется "только для чтения" или "чтение и запись"
-901	-336068796	-dyn_role_does_not_exist	-SQL role <string> does not exist. Не существует роль SQL <строка>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336068797	-dyn_no_grant_admin_opt	-User <string> has no grant admin option on SQL role <string>. Пользователь <строка> не получил от администратора роли SQL <строка>
-901	-336068798	-dyn_user_not_role_member	-User <string> is not a member of SQL role <string>. Пользователь <строка> не является участником роли SQL <строка>
-901	-336068799	-dyn_delete_role_failed	-<string> is not the owner of SQL role <string>. <строка> не является владельцем роли SQL <строка>
-901	-336068800	-dyn_grant_role_to_user	-<string> is a SQL role and not a user. <строка> является ролью SQL, а не пользователем
-901	-336068801	-dyn_inv_sql_role_name	-User name <string> could not be used for SQL role. Имя пользователя <строка> не может быть использовано в качестве роли SQL
-901	-336068802	-dyn_dup_sql_role	-SQL role <string> already exists. Роль SQL <строка> уже существует
-901	-336068803	-dyn_kywd_spec_for_role	-Keyword <string> can not be used as a SQL role name. Ключевое слово <строка> не может быть использовано в качестве имени роли SQL

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336068804	-dyn_roles_not_supported	-SQL roles are not supported in on older versions of the database. A backup and restore of the database is required. Роли SQL не поддерживаются в старых версиях базы данных. Требуется копирование и восстановление базы данных
-901	-336068820	-dyn_zero_len_id	-Zero length identifiers are not allowed. Недопустима нулевая длина идентификатора
-901	-336330753	-gbak_unknown_switch	-Found unknown switch. Найден неизвестный переключатель
-901	-336330754	-gbak_page_size_missing	-Page size parameter missing. Отсутствует параметр размера страницы
-901	-336330755	-gbak_page_size_toobig	-Page size specified (<number>) greater than limit (8192 bytes). Указанный размер страницы (<число>) больше ограничения (8192 байт)
-901	-336330756	-gbak_redir_ouput_missing	-Redirect location for output is not specified. Не задано перенаправление вывода
-901	-336330757	-gbak_switches_conflict	-Conflicting switches for backup/restore. Конфликт переключателей для копирования/восстановления
-901	-336330758	-gbak_unknown_device	-Device type <string> not known. Тип устройства <строка> не известен

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336330759	-gbak_no_protection	-Protection is not there yet. Защита пока не установлена
-901	-336330760	-gbak_page_size_not_allowed	-Page size is allowed only on restore or create. Размер страницы нужен только при восстановлении или при создании
-901	-336330761	-gbak_multi_source_dest	-Multiple sources or destinations specified. Указано множество источников или результатов
-901	-336330762	-gbak_filename_missing	-Requires both input and output filenames. Требуются имена входного и выходного файлов
-901	-336330763	-gbak_dup_inout_names	-Input and output have the same name. Disallowed. Вход и выход имеют одинаковые имена. Отвергаются
-901	-336330764	-gbak_inv_page_size	-Expected page size, encountered "<string>". Ожидается размер страницы, появилось "<строка>"
-901	-336330765	-gbak_db_specified	-REPLACE specified, but the first file <string> is a database. Указано REPLACE, однако первый файл <строка> является базой данных

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336330766	-gbak_db_exists	-Database <string> already exists. To replace it, use the -R switch. База данных <строка> уже существует. Для ее замены используйте переключатель R
-901	-336330767	-gbak_unk_device	-Device type not specified. Не задан тип устройства
-901	-336330772	-gbak_blob_info_failed	-Gds_$blob_info failed. Ошибка в gds_$blob_info
-901	-336330773	-gbak_unk_blob_item	-Do not understand BLOB INFO item <number>. Неизвестный элемент BLOB INFO <число>
-901	-336330774	-gbak_get_seg_failed	-Gds_$get_segment failed. Ошибка в gds_$get_segment
-901	-336330775	-gbak_close_blob_failed	-Gds_$close_blob failed. Ошибка в gds_$close_blob
-901	-336330776	-gbak_open_blob_failed	-Gds_$open_blob failed. Ошибка в gds_$open_blob
-901	-336330777	-gbak_put_blr_gen_id_failed	-Failed in put_blr_gen_id. Ошибка в put_blr_gen_id
-901	-336330778	-gbak_unk_type	-Data type <number> not understood. Тип данных <число> не известен

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336330779	-gbak_comp_req_failed	-Gds_$compile_request failed. Ошибка в gds_$compile_request
-901	-336330780	-gbak_start_req_failed	-Gds_$start_request failed. Ошибка в gds_$start_request
-901	-336330781	-gbak_rec_failed	- gds_$receive failed. Ошибка в gds_$receive
-901	-336330782	-gbak_rel_req_failed	-Gds_$release_request failed. Ошибка в gds_$release_request
-901	-336330783	-gbak_db_info_failed	-gds_$database_info failed. Ошибка в gds_$database_info
-901	-336330784	-gbak_no_db_desc	-Expected database description record. Ожидается запись описания базы данных
-901	-336330785	-gbak_db_create_failed	-Failed to create database <string>. Ошибка при создании базы данных <строка>
-901	-336330786	-gbak_decomp_len_error	-RESTORE: decompression length error. RESTORE: ошибка в длине декомпрессии
-901	-336330787	-gbak_tbl_missing	-Cannot find table <string>. Невозможно найти таблицу <строка>
-901	-336330788	-gbak_blob_col_missing	-Cannot find column for BLOB. Невозможно найти столбец для BLOB

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336330789	-gbak_create_blob_failed	-Gds_$create_blob failed. Ошибка в gds_$create_blob
-901	-336330790	-gbak_put_seg_failed	-Gds_$put_segment failed. Ошибка в gds_$put_segment
-901	-336330791	-gbak_rec_len_exp	-Expected record length. Ожидается длина записи
-901	-336330792	-gbak_inv_rec_len	-Wrong length record, expected <number> encountered <number>. Неверная длина записи, ожидается <число>, встретилось <число>
-901	-336330793	-gbak_exp_data_type	-Expected data attribute. Ожидается атрибут данных
-901	-336330794	-gbak_gen_id_failed	-Failed in store_blr_gen_id. Ошибка в store_blr_gen_id
-901	-336330795	-gbak_unk_rec_type	-Do not recognize record type <number>. Не распознан тип записи <число>
-901	-336330796	-gbak_inv_bkup_ver	-Expected backup version 1, 2, or 3. Found <number>. Ожидается версия копии 1, 2 или 3. Найдено <число>
-901	-336330797	-gbak_missing_bkup_desc	-Expected backup description record. Ожидается запись описания копии
-901	-336330798	-gbak_string_trunc	-String truncated. Усечение строки

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336330799	-gbak_cant_rest_record	-Warning—record could not be restored. Предупреждение — запись не может быть восстановлена
-901	-336330800	-gbak_send_failed	-Gds_$send failed. Ошибка в gds_$send
-901	-336330801	-gbak_no_tbl_name	-No table name for data. Для данных нет имени таблицы
-901	-336330802	-gbak_unexp_eof	-Unexpected end of file on backup file. Конец файла в файле копии
-901	-336330803	-gbak_db_format_too_old	-Database format <number> is too old to restore to. Формат базы данных <число> слишком старый для восстановления
-901	-336330804	-gbak_inv_array_dim	-Array dimension for column <string> is invalid. Неверная размерность массива для столбца <строка>
-901	-336330807	-gbak_xdr_len_expected	-Expected XDR record length. Ожидается длина записи
-901	-336330817	-gbak_open_bkup_error	-Cannot open backup file <string>. Невозможно открыть файл копии <строка>
-901	-336330818	-gbak_open_error	-Cannot open status and error output file <string>. Невозможно открыть состояние и ошибка вывода в файл <строка>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336330934	-gbak_missing_block_fac	-Blocking factor parameter missing. Отсутствует параметр коэффициент блокирования
-901	-336330935	-gbak_inv_block_fac	-Expected blocking factor, encountered "<string>". Ожидается коэффициент блокирования, встречено "<строка>"
-901	-336330936	-gbak_block_fac_specified	-A blocking factor may not be used in conjunction with device CT. Коэффициент блокирования не может использоваться вместе с устройством CT
-901	-336330940	-gbak_missing_username	-User name parameter missing. Отсутствует параметр имя пользователя
-901	-336330941	-gbak_missing_password	-Password parameter missing. Отсутствует параметр пароль
-901	-336330952	-gbak_missing_skipped_bytes	-Missing parameter for the number of bytes to be skipped. Отсутствует параметр количества пропускаемых байтов
-901	-336330953	-gbak_inv_skipped_bytes	-Expected number of bytes to be skipped, encountered "<string>". Ожидается количество пропускаемых байтов, встречено "<строка>"
-901	-336330965	-gbak_err_restore_charset	-Bad attribute for RDB$CHARACTER_SETS. Неверный атрибут для RDB$CHARACTER_SETS

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336330967	gbak_err_restore_collation	-Bad attribute for RDB$COLLATIONS. Неверный атрибут для RDB$COLLATIONS
-901	-336330972	gbak_read_error	-Unexpected I/O error while reading from backup file. Ошибка ввода-вывода при чтении из файла копии
-901	-336330973	gbak_write_error	-Unexpected I/O error while writing to backup file. Ошибка ввода-вывода при записи в файл копии
-901	-336330985	gbak_db_in_use	-Could not drop database <string> (database might be in use). Невозможно удалить базу данных <строка> (возможно база данных используется)
-901	-336330990	gbak_sysmemex	-System memory exhausted. Исчерпана системная память
-901	-336331002	gbak_restore_role_failed	-Bad attributes for restoring SQL role. Неверный атрибут для восстановления роли SQL
-901	-336331005	gbak_role_op_missing	-SQL role parameter missing. Отсутствует параметр роли SQL
-901	-336331010	gbak_page_buffers_missing	-Page buffers parameter missing. Отсутствует параметр буферов страниц

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336331011	-gbak_page_buffers_wrong_param	-Expected page buffers, encountered "<string>". Ожидаются буферы страниц, встречено "<строка>"
-901	-336331012	-gbak_page_buffers_restore	-Page buffers is allowed only on restore or create. Буферы страниц допустимы только при восстановлении или создании
-901	-336331014	-gbak_inv_size	-Size specification either missing or incorrect for file <string>. Указание размера отсутствует или неверное для файла <строка>
-901	-336331015	-gbak_file_outof_sequence	-File <string> out of sequence. Файл <строка> не задан в последовательности
-901	-336331016	-gbak_join_file_missing	-Can't join—one of the files missing. Соединение невозможно — один из файлов отсутствует
-901	-336331017	-gbak_stdin_not_supptd	-Standard input is not supported when using join operation. Стандартный ввод не поддерживается при использовании операции соединения
-901	-336331018	-gbak_stdout_not_supptd	-Standard output is not supported when using split operation. Стандартный вывод не поддерживается при использовании операции разделения

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336331019	-gbak_bkup_corrupt	-Backup file <string> might be corrupt. Возможно файл копии <строка> разрушен
-901	-336331020	-gbak_unk_db_file_spec	-Database file specification missing. Отсутствует указание файла базы данных
-901	-336331021	-gbak_hdr_write_failed	-Can’t write a header record to file <string>. Невозможно записать заголовочную запись в файл <строка>
-901	-336331022	-gbak_disk_space_ex	-Free disk space exhausted. Исчерпано свободное дисковое пространство
-901	-336331023	-gbak_size_lt_min	-File size given (<number>) is less than minimum allowed (<number>). Заданный размер файла (<число>) меньше минимально допустимого (<число>)
-901	-336331025	-gbak_svc_name_missing	-Service name parameter missing. Отсутствует параметр имени сервиса
-901	-336331026	-gbak_not_ownr	-Cannot restore over current database, must be SYSDBA or owner of the existing database. Невозможно восстановление поверх текущей базы данных, должен быть пользователь SYSDBA или владелец существующей базы данных

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336331031	-gbak_mode_req	-"read_only" or "read_write" required. Требуется "read_only" или "read_write"
-901	-336331033	-gbak_just_data	-Just data ignore all constraints, etc. Данные игнорируют все ограничения
-901	-336331034	-gbak_data_only	-Restoring data only ignoring foreign key, unique, not null & other constraints. Восстановление данных только при игнорировании ограничений внешнего ключа, уникального ключа, NOT NULL и других ограничений
-901	-336723983	-gsec_cant_open_db	-Unable to open database. Невозможно открыть базу данных
-901	-336723984	-gsec_switches_error	-Error in switch specifications. Ошибка в задании переключателя
-901	-336723985	-gsec_no_op_spec	-No operation specified. Не указана операция
-901	-336723986	-gsec_no_usr_name	-No user name specified. Не задано имя пользователя
-901	-336723987	-gsec_err_add	-Add record error. Ошибка добавления записи
-901	-336723988	-gsec_err_modify	-Modify record error. Ошибка изменения записи
-901	-336723989	-gsec_err_find_mod	-Find/modify record error. Ошибка поиска/изменения записи

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336723990	-gsec_err_rec_not_found	-Record not found for user: <string>. Не найдена запись для пользователя <строка>
-901	-336723991	-gsec_err_delete	-Delete record error. Ошибка удаления записи
-901	-336723992	-gsec_err_find_del	-Find/delete record error. Ошибка поиска/удаления записи
-901	-336723996	-gsec_err_find_disp	-Find/display record error. Ошибка поиска/отображения записи
-901	-336723997	-gsec_inv_param	-Invalid parameter, no switch defined. Неверный параметр, не определено переключателей
-901	-336723998	-gsec_op_specified	-Operation already specified. Операция уже задана
-901	-336723999	-gsec_pw_specified	-Password already specified. Пароль уже задан
-901	-336724000	-gsec_uid_specified	-Uid already specified. UID уже задано
-901	-336724001	-gsec_gid_specified	-Gid already specified. GID уже задано
-901	-336724002	-gsec_proj_specified	-Project already specified. Проект уже задан
-901	-336724003	-gsec_org_specified	-Organization already specified. Организация уже задана

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336724004	-gsec_fname_specified	-First name already specified. Имя уже задано
-901	-336724005	-gsec_mname_specified	-Middle name already specified. Второе имя уже задано
-901	-336724006	-gsec_lname_specified	-Last name already specified. Фамилия уже задана
-901	-336724008	-gsec_inv_switch	-Invalid switch specified. Задан неверный переключатель
-901	-336724009	-gsec_amb_switch	-Ambiguous switch specified. Задан неоднозначный переключатель
-901	-336724010	-gsec_no_op_specified	-No operation specified for parameters. Для параметров не задана операция
-901	-336724011	-gsec_params_not_allowed	-No parameters allowed for this operation. Параметры недопустимы для этой операции
-901	-336724012	-gsec_incompat_switch	-Incompatible switches specified. Заданы несовместимые переключатели
-901	-336724044	-gsec_inv_username	-Invalid user name (maximum 31 bytes allowed). Неверное имя пользователя (допустимо максимум 31 байт)

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336724045	-gsec_inv_pw_length	-Warning—maximum 8 significant bytes of password used. Предупреждение — в пароле используется максимум 8 значащих байтов
-901	-336724046	-gsec_db_specified	-Database already specified. База данных уже задана
-901	-336724047	-gsec_db_admin_specified	-Database administrator name already specified. Имя администратора базы данных уже задано
-901	-336724048	-gsec_db_admin_pw_specified	-Database administrator password already specified. Пароль администратора базы данных уже задан
-901	-336724049	-gsec_sql_role_specified	-SQL role name already specified. Имя роли SQL уже задано
-901	-336920577	-gstat_unknown_switch	-Found unknown switch. Найден неизвестный переключатель
-901	-336920578	-gstat_retry	-Please retry, giving a database name. Пожалуйста, повторите, задав имя базы данных
-901	-336920579	-gstat_wrong_ods	-Wrong ODS version, expected <number>, encountered <number>. Неверная версия ODS, ожидается <число>, встречено <число>
-901	-336920580	-gstat_unexpected_eof	-Unexpected end of database file. Конец файла базы данных

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-901	-336920605	-gstat_open_err	-Can't open database file <string>. Невозможно открыть файл базы данных <строка>
-901	-336920606	-gstat_read_err	-Can't read a database page. Невозможно прочесть страницу базы данных
-901	-336920607	-gstat_sysmemex	-System memory exhausted. Исчерпана системная память
-902	-335544333	-bug_check	-Internal gds software consistency check (<string>). Проверка достоверности внутреннего программного продукта gds (<строка>)
-902	-335544335	-db_corrupt	-Database file appears corrupt (<string>). Разрушение файла базы данных (<строка>)
-902	-335544344	-io_error	-I/O error for file %.0s"<string>". Ошибка ввода-вывода для файла "<строка>"
-902	-335544346	-metadata_corrupt	-Corrupt system table. Разрушена системная таблица
-902	-335544373	-sys_request	-Operating system directive <string> failed. Ошибочная директива операционной системы <строка>
-902	-335544384	-badblk	-Internal error. Внутренняя ошибка

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	-335544385	-invpoolcl	-Internal error. Внутренняя ошибка
-902	-335544387	-relbadblk	-Internal error. Внутренняя ошибка
-902	-335544388	-blktoobig	-Block size exceeds implementation restriction. Размер блока превышает ограничение реализации
-902	-335544394	-badodsver	-Incompatible version of on-disk structure. Несовместимая версия для ODS
-902	-335544397	-dirtypage	-Internal error. Внутренняя ошибка
-902	-335544398	-waifortra	-Internal error. Внутренняя ошибка
-902	-335544399	-doubleloc	-Internal error. Внутренняя ошибка
-902	-335544400	-nodnotfnd	-Internal error. Внутренняя ошибка
-902	-335544401	-dupnodfnd	-Internal error. Внутренняя ошибка
-902	-335544402	-locnotmar	-Internal error. Внутренняя ошибка
-902	-335544404	-corrupt	-Database corrupted. База данных разрушена
-902	-335544405	-badpage	-Checksum error on database page <number>. Ошибка контрольной суммы для страницы базы данных <число>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	-335544406	-badindex	-Index is broken. Индекс разрушен
-902	-335544409	-trareqmis	-Transaction—request mismatch (synchronization error). Транзакция — несогласованный запрос (ошибка синхронизации)
-902	-335544410	-badhndcnt	-Bad handle count. Ошибочный счетчик дескриптора
-902	-335544411	-wrotpbver	-Wrong version of transaction parameter block. Неверная версия блока параметров транзакции
-902	-335544412	-wroblrver	-Unsupported BLR version (expected <number>, encountered <number>). Неподдерживаемая версия BLR (ожидается <число>, встречено <число>)
-902	-335544413	-wrodpbver	-Wrong version of database parameter block. Неверная версия блока параметров базы данных
-902	-335544415	-badrelation	-Database corrupted. База данных разрушена
-902	-335544416	-nodetach	-Internal error. Внутренняя ошибка
-902	-335544417	-notremote	-Internal error. Внутренняя ошибка
-902	-335544422	-dbfile	-Internal error. Внутренняя ошибка

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	-335544423	-orphan	-Internal error. Внутренняя ошибка
-902	-335544432	-lockmanerr	-Lock manager error. Ошибка менеджера блокировок
-902	-335544436	-sqlerr	-SQL error code = <number>. Код ошибки SQL = <число>
-902	-335544448	-bad_sec_info	—
-902	-335544449	-invalid_sec_info	—
-902	-335544470	-buf_invalid	-Cache buffer for page <number> invalid. Неверный буфер кэша для страницы <число>
-902	-335544471	-indexnotdefined	-There is no index in table <string> with id <number> Не существует индексов для таблицы <строка> с идентификатором <число>
-902	-335544472	-login	-Your user name and password are not defined. Ask your database administrator to set up a Firebird login. Не определены ваше имя и пароль. Чтобы установить соединение с Firebird, обратитесь к администратору базы данных
-902	-335544506	-shutinprog	-Database <string> shutdown in progress. Выполняется останов базы данных <строка>

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	-335544528	-shutdown	-Database <string> shutdown. База данных <строка> остановлена
-902	-335544557	-shutfail	-Database shutdown unsuccessful. Неуспешный останов базы данных
-902	-335544569	-dsql_error	-Dynamic SQL Error. Ошибка динамического SQL
-902	-335544653	-psw_attach	-Cannot attach to password database. Невозможно соединиться с базой данных пароля
-902	-335544654	-psw_start_trans	-Cannot start transaction for password database. Невозможно стартовать транзакцию для базы данных пароля
-902	-335544717	-err_stack_limit	-Stack size insufficient to execute current request. Размер стека недостаточен для выполнения текущего запроса
-902	-335544721	-network_error	-Unable to complete network request to host "<string>". Невозможно завершить сетевой запрос на хост "<строка>"
-902	-335544722	-net_connect_err	-Failed to establish a connection. Ошибка при установлении соединения

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	-335544723	-net_connect_listen_err	-Error while listening for an incoming connection. Ошибка при прослушивании входного соединения
-902	-335544724	-net_event_connect_err	-Failed to establish a secondary connection for event processing. Ошибка при установлении вторичного соединения для обработки события
-902	-335544725	-net_event_listen_err	-Error while listening for an incoming event connection request. Ошибка при прослушивании запроса события соединения
-902	-335544726	-net_read_err	-Error reading data from the connection. Ошибка чтения данных из соединения
-902	-335544727	-net_write_err	-Error writing data to the connection. Ошибка записи данных в соединение
-902	-335544732	-unsupported_network_drive	-Access to databases on file servers is not supported. Доступ к базам данных в файловых серверах не поддерживается
-902	-335544733	-io_create_err	-Error while trying to create file. Ошибка при попытке создания файла
-902	-335544734	-io_open_err	-Error while trying to open file. Ошибка при попытке открытия файла

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	-335544735	-io_close_err	-Error while trying to close file. Ошибка при попытке закрытия файла
-902	-335544736	-io_read_err	-Error while trying to read from file. Ошибка при попытке чтения из файла
-902	-335544737	-io_write_err	-Error while trying to write to file. Ошибка при попытке записи в файл
-902	-335544738	-io_delete_err	-Error while trying to delete file. Ошибка при попытке удаления файла
-902	-335544739	-io_access_err	-Error while trying to access file. Ошибка при попытке доступа к файлу
-902	-335544745	-login_same_as_role_name	-Your login <string> is same as one of the SQL role name. Ask your database administrator to set up a valid Firebird login. Ваше регистрационное имя <строка> то же, что и имя роли SQL. Попросите вашего администратора базы данных установить допустимое регистрационное имя Firebird
-902	-335544791	-file_in_use	-The file <string> is currently in use by another process. Try again later. Файл <строка> в настоящее время используется другим процессом. Попытайтесь позже

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-902	-335544795	`-unexp_spb_form`	`-Unexpected item in service parameter block, expected <string>.` Неопределенный элемент в блоке параметров сервиса, ожидается <строка>
-902	-335544809	`-extern_func_dir_error`	`-Function <string> is in <string>, which is not in a permitted directory for external functions.` Функция <строка> находится в <строка>, что не является доступным каталогом для внешних функций
-902	-335544819	`-io_32bit_exceeded_err`	`-File exceeded maximum size of 2GB. Add another database file or use a 64 bit I/O version of Firebird.` Файл превысил максимальный размер 2 Гбайт. Добавьте другой файл базы данных или используйте 64-битовую версию Firebird
-902	-335544820	`-invalid_savepoint`	`-Unable to find savepoint with name <string> in transaction context.` Невозможно найти точку сохранения с именем <строка> в контексте транзакции
-902	-335544831	`-conf_access_denied`	`-Access to <string> "<string>" is denied by server administrator.` Доступ к <строка> "<строка>" отвергнут администратором сервера

Таблица П3.1 *(продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	-335544324	-bad_db_handle	-Invalid database handle (no active connection). Неверный дескриптор базы данных (нет активных соединений)
-904	-335544375	-unavailable	-Unavailable database. Недоступная база данных
-904	-335544381	-imp_exc	-Implementation limit exceeded. Исчерпан лимит выполнения
-904	-335544386	-nopoolids	-Too many requests. Слишком много запросов
-904	-335544389	-bufexh	-Buffer exhausted. Исчерпан буфер
-904	-335544391	-bufinuse	-Buffer in use. Буфер используется
-904	-335544393	-reqinuse	-Request in use. Запрос используется
-904	-335544424	-no_lock_mgr	-No lock manager available. Нет доступного менеджера блокировок
-904	-335544430	-virmemexh	-Unable to allocate memory from operating system. Невозможно выделить память в операционной системе
-904	-335544451	-update_conflict	-Update conflicts with concurrent update. Изменение конфликтует с текущим изменением
-904	-335544453	-obj_in_use	-Object <string> is in use. Объект <строка> используется

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	-335544455	-shadow_accessed	-Cannot attach active shadow file. Невозможно соединиться с активным файлом оперативной копии
-904	-335544460	-shadow_missing	-A file in manual shadow <number> is unavailable. Файл в ручной оперативной копии <число> недоступен
-904	-335544661	-index_root_page_full	-Cannot add index, index root page is full. Невозможно добавить индекс, корневая страница индексов заполнена
-904	-335544676	-sort_mem_err	-Sort error: not enough memory. Ошибка сортировки: нет достаточно памяти
-904	-335544683	-req_depth_exceeded	-Request depth exceeded. (Recursive definition?) Превышена глубина запроса (рекурсивное определение?)
-904	-335544758	-sort_rec_size_err	-Sort record size of <number> bytes is too big. Размер записи сортировки в <число> байтов слишком велик
-904	-335544761	-too_many_handles	-Too many open handles to database. Слишком много открытых дескрипторов базы данных
-904	-335544792	-service_att_err	-Cannot attach to services manager. Невозможно подключиться к менеджеру сервисов

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-904	-335544799	-svc_name_missing	-The service name was not specified. Не указано имя сервиса
-904	-335544813	-optimizer_between_err	-Unsupported field type specified in BETWEEN predicate. Указан неподдерживаемый тип поля в предикате BETWEEN
-904	-335544827	-exec_sql_invalid_arg	-Invalid argument in EXECUTE STATEMENT- cannot convert to string. Неверный аргумент в EXECUTE STATEMENT — невозможно конвертировать в строку
-904	-335544828	-exec_sql_invalid_req	-Wrong request type in EXECUTE STATEMENT '<string>'. Ошибочный тип запроса в EXECUTE STATEMENT '<строка>'
-904	-335544829	-exec_sql_invalid_var	-Variable type (position <number>) in EXECUTE STATEMENT '<string>' INTO does not match returned column type. Тип переменной (позиция <число> в EXECUTE STATEMENT '<строка>' INTO не соответствует возвращаемому типу столбца
-904	-335544830	-exec_sql_max_call_exceeded	-Too many recursion levels of EXECUTE STATEMENT. Слишком много уровней рекурсии в EXECUTE STATEMENT

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-906	-335544744	-max_att_exceeded	-Maximum user count exceeded. Contact your database administrator. Превышен максимум счетчика пользователей
-909	-335544667	-drdb_completed_with_errs	-Drop database completed with errors. Удаление базы данных завершилось с ошибками
-911	-335544459	-rec_in_limbo	-Record from transaction <number> is stuck in limbo. Запись транзакции <число> становится зависшей
-913	-335544336	-deadlock	-Deadlock. Взаимная блокировка
-922	-335544323	-bad_db_format	-File <string> is not a valid database. Файл <строка> не является допустимой базой данных
-923	-335544421	-connect_reject	-Connection rejected by remote interface. Соединение отменено удаленным интерфейсом
-923	-335544461	-cant_validate	-Secondary server attachments cannot validate databases. Вторичные подключения к серверу не могут проверять базы данных
-923	-335544464	-cant_start_logging	-Secondary server attachments cannot start logging. Вторичные подключения к серверу не могут запускать соединения

***Таблица П3.1** (продолжение)*

SQLCODE	GDSCODE	Символ	Текст сообщения
-924	-335544325	-bad_dpb_content	-Bad parameters on attach or create database. Неверные параметры при подключении или создании базы данных
-924	-335544441	-bad_detach	-Database detach completed with errors. Отключение от базы данных завершилось с ошибками
-924	-335544648	-conn_lost	-Connection lost to pipe server. Потеря соединения с каналом сервера
-926	-335544447	-no_rollback	-No rollback performed. Не выполнен откат транзакции
-999	-335544689	-ib_error	-Firebird error. Ошибка Firebird

ПРИЛОЖЕНИЕ 4

Структура некоторых системных таблиц

В это отдельное приложение я вынес описания некоторых системных таблиц и программ, отображающих в достаточно удобном виде содержимое этих таблиц. Не знаю, насколько эти сведения понадобятся вам в вашей практической деятельности, но для расширения общего кругозора они явно не повредят.

При создании новой, "пустой" базы данных InterBase или Firebird создается множество системных таблиц, которые содержат *метаданные*, т. е. данные, описывающие данные базы данных. Все метаданные, как и обычные данные, хранятся в таблицах, но в таблицах системных. Как правило, имена системных таблиц начинаются с символов `RDB$`.

Мы уже рассматривали в *приложении 2* структуру системных таблиц, хранящих доступные наборы символов и порядок сортировки — `RDB$CHARACTER_SETS` и `RDB$COLLATIONS`. Здесь мы рассмотрим некоторые другие наиболее интересные и важные для нас системные таблицы, исследование которых поможет более глубоко изучить эти серверы баз данных.

Кроме этого, мы напишем несколько программ, которые дадут нам возможность отображать содержимое отдельных системных таблиц в удобном для нас виде.

Структуру системных таблиц я описываю несколько упрощенно, опуская столбцы, которые на сегодняшний день не используются.

П4.1. База данных, файлы, страницы

В этом разделе мы рассмотрим три таблицы, которые дают возможность нам просмотреть базу данных в целом.

П4.1.1. Структура системных таблиц

Системная таблица `RDB$DATABASE` хранит общие данные о базе данных. Ее сокращенная структура представлена в табл. П4.1.

Таблица П4.1. *Структура таблицы* RDB$DATABASE

Имя столбца	Тип	Описание
RDB$RELATION_ID	SMALLINT	Количество таблиц и представлений в базе данных
RDB$CHARACTER_SET_NAME	CHAR(31)	Набор символов по умолчанию для базы данных

В системной таблице RDB$FILES хранятся сведения о вторичных файлах базы данных и файлах оперативных копий (табл. П4.2). Ни то ни другое мы с вами не рассматривали. Если вам когда-нибудь потребуется создавать многофайловую базу данных или использовать средства оперативного копирования (shadow), вы легко найдете все необходимые сведения в книге Хелен Борри. А средства отображения этих данных у вас уже будут под рукой.

Таблица П4.2. *Структура таблицы* RDB$FILES

Имя столбца	Тип	Описание
RDB$FILE_NAME	VARCHAR(253)	Имя вторичного файла базы данных в многофайловой базе данных или файла оперативной копии
RDB$FILE_SEQUENCE	SMALLINT	Порядковый номер вторичного файла или номер файла в наборе оперативных копий
RDB$FILE_START	INTEGER	Начальный номер страницы
RDB$FILE_LENGTH	INTEGER	Длина файла в страницах базы данных
RDB$SHADOW_NUMBER	SMALLINT	Номер набора оперативных копий

Системная таблица RDB$PAGES хранит сведения о страницах базы данных (табл. П4.3).

Таблица П4.3. *Структура таблицы* RDB$PAGES

Имя столбца	Тип	Описание
RDB$PAGE_NUMBER	INTEGER	Уникальный номер страницы базы данных, которая была выделена физически
RDB$RELATION_ID	SMALLINT	Идентификатор таблицы, чьи данные хранятся на этой странице
RDB$PAGE_SEQUENCE	INTEGER	Последовательный номер страницы по отношению к другим страницам, выделенным для этой таблицы
RDB$PAGE_TYPE	SMALLINT	Идентифицирует тип данных, хранящихся на этой странице (данные таблицы, индекса)

Для отображения этих сведений базы данных напишем простую программу.

П4.1.2. Программа с использованием компонентов FIBPlus

Создайте новый проект. Положите на форму компоненты, как показано на рис. П4.1.

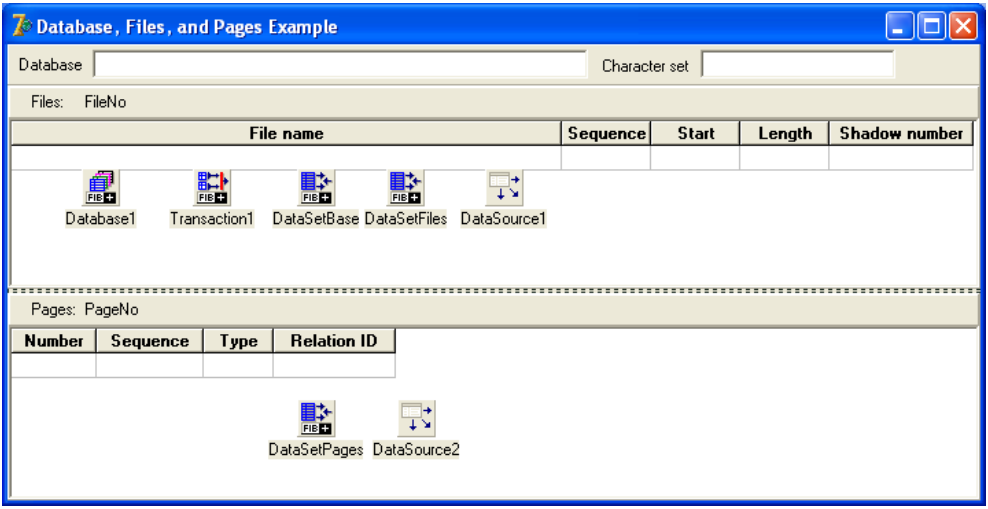


Рис. П4.1. Программа отображения характеристик базы данных, файлов и страниц

Компонент базы данных свяжите с нашей базой данных, как обычно.

Для компонента набора данных `DataSetBase`, работающего с системной таблицей базы данных, задайте только оператор `SELECT`:

```
SELECT * FROM RDB$DATABASE
```

Остальные операторы не нужны, поскольку мы не собираемся вручную корректировать данные системных таблиц. Для некоторых версий InterBase это и невозможно.

Для компонента `DataSetFiles` (список файлов) укажите следующий оператор `SELECT`:

```
SELECT * FROM RDB$FILES
```

И для компонента `DataSetPages` (список страниц) задайте `SELECT`:

```
SELECT * FROM RDB$PAGES
```

Компоненты свяжите друг с другом, как обычно мы делаем в наших программах.

Форма должна иметь приблизительно следующий вид — рис. П4.1.

Напишите обработчик события `OnShow` для формы (листинг П4.1).

Листинг П4.1. Обработчик события `OnShow` для формы

```
procedure TFormMain.FormShow(Sender: TObject);
begin
  Database1.Connected := True;
  DataSetBase.Active := True;
  DataSetFiles.Active := True;
  DataSetPages.Active := True;
  DatabasePath.Text := Database1.DBName;
  CharacterSet.Text :=
    DataSetBase.FieldByName('RDB$CHARACTER_SET_NAME').AsString;
  FileNo.Caption := IntToStr(DataSetFiles.RecordCount);
  PageNo.Caption := IntToStr(DataSetPages.RecordCount);
end;
```

Здесь мы соединяемся с нашей базой данных, открываем все три набора данных, помещаем в поля редактирования путь к базе данных и имя набора символов по умолчанию, а также выводим количество записей в таблице файлов и таблице страниц.

При завершении работы программы мы закрываем наборы данных и отсоединяемся от базы данных — листинг П4.2.

Листинг П4.2. Обработчик события `OnClose` для формы

```
procedure TFormMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  DataSetBase.Active := False;
  DataSetFiles.Active := False;
  DataSetPages.Active := False;
  Database1.Connected := False;
end;
```

Больше ничего в этой программе делать не нужно.

Запустите программу на выполнение. Вы увидите путь к базе данных, набор символов по умолчанию для базы данных и список страниц базы данных. Сведений о файлах не будет, потому что мы не создавали вторичных файлов и оперативных копий.

П4.1.3. Программа с использованием компонентов IBX

Никаких сложностей по созданию программы с IBX не будет. Заменяем компоненты FIBPlus на соответствующие компоненты IBX.

Для каждого компонента набора данных вручную вводим соответствующие операторы `SELECT` (в списке доступных таблиц их также не будет).

В обработчик события формы `OnShow` не забудем добавить обращения к методу наборов данных для файлов и страниц `FetchAll`, чтобы были считаны все записи из таблиц, и мы отобразили бы правильное их количество.

В остальном программа работает аналогичным образом.

> ***Замечание***
>
> Если вам позволяет размер монитора, в этих двух программах удобнее было бы разместить сетки просмотра наборов данных не горизонтально, а вертикально. У себя я так и сделал.

П4.2. Отношения (таблицы, представления)

Сейчас мы рассмотрим наиболее сложную и важную группу системных таблиц, связанных с хранением метаданных о таблицах нашей базы данных.

П4.2.1. Структура системных таблиц

Вообще говоря, можно построить довольно сложную и интересную программу для практически полного просмотра описания наших таблиц и представлений — в каком виде это хранится в базе данных. Мы ограничимся двумя системными таблицами. Как вы понимаете, мне хочется дать вам домашнее задание на написание соответствующей программы.

В области метаданных есть две основные таблицы, описывающие отношения (напомню, что отношениями в реляционных базах данных являются таблицы и представления). Это таблицы `RDB$RELATIONS` (табл. П4.4), которая хранит сведения об отношениях, и `RDB$RELATION_FIELDS` (табл. П4.5), содержащая сведения о столбцах.

Таблица П4.4. *Структура таблицы* `RDB$RELATIONS`

Имя столбца	Тип	Описание
`RDB$RELATION_ID`	`SMALLINT`	Внутренний идентификатор таблицы
`RDB$SYSTEM_FLAG`	`SMALLINT`	Указывает, создана ли таблица пользователем (0) или системой (1 или выше)
`RDB$FIELD_ID`	`SMALLINT`	Не совсем понятное название столбца. На самом деле содержит количество полей в отношении
`RDB$RELATION_NAME`	`CHAR(31)`	Имя таблицы или представления
`RDB$OWNER_NAME`	`VARCHAR(31)`	Имя пользователя — владельца (создателя) таблицы или представления

Таблица П4.5. *Структура таблицы* `RDB$RELATION_FIELDS`

Имя столбца	Тип	Описание
`RDB$FIELD_NAME`	`CHAR(31)`	Имя столбца
`RDB$RELATION_NAME`	`CHAR(31)`	Имя таблицы или представления
`RDB$FIELD_SOURCE`	`CHAR(31)`	Если столбец основан на домене, то хранит имя домена
`RDB$FIELD_ID`	`SMALLINT`	Внутренний номер идентификатора
`RDB$SYSTEM_FLAG`	`SMALLINT`	Определено пользователем (0) или системой (1 или выше)

Напоминаю, что здесь я представил резко упрощенные структуры этих системных таблиц. Полные структуры вы можете посмотреть у Хелен Борри или в документации по InterBase.

> ***Замечание***
>
> В таблицах для столбцов, содержащих имена объектов базы данных (генераторы, таблицы, столбцы таблиц и т. д.) указан размер 31. Это верно для Firebird и InterBase версии 6.*x* и более ранних. В InterBase, начиная с версии 7.*x*, максимальный размер имен доведен до 67 символов.

П4.2.2. Программы просмотра отношений

Не хочу тратить много времени на подробное описание программ просмотра таблиц и представлений в системных таблицах. Все подробности нам с вами

хорошо известны. Здесь у нас классический случай отношения "главная/подчиненная".

Свойствам компонентов базы данных и транзакции устанавливаются обычные значения.

В двух компонентах наборов данных нужно задать только операторы `SELECT`.

У набора данных `DSRelations` для работы с таблицей `RDB$RELATIONS` нужен следующий оператор:

```
SELECT * FROM RDB$RELATIONS
```

В наборе данных `DSField`, работающем с таблицей `RDB$RELATION_FIELDS`, задаем оператор:

```
SELECT *
  FROM RDB$RELATION_FIELDS
WHERE
  RDB$RELATION_NAME = ?RDB$RELATION_NAME
```

Здесь в предложении `WHERE` задается условие связи с главной таблицей. Напомню, что для этого набора данных нужно свойству `DataSource` присвоить значение `DataSource1`.

Внешний вид формы с использованием компонентов FIBPlus представлен на рис. П4.2.

Как установить значения и связи компонентов, вы теперь знаете не хуже меня. Создайте правильную программу (две программы, включая ту, которая использует компоненты IBX).

В обеих таблицах есть столбцы флагов, которые содержат признаки, указывающие, определен ли объект системой или пользователем. Чтобы выводить осмысленные тексты вместо цифр 0 или 1, нужно для компонента `DBGrid` написать обработчик события `DrawColumnCell` (прорисовка ячейки) — листинг П4.3.

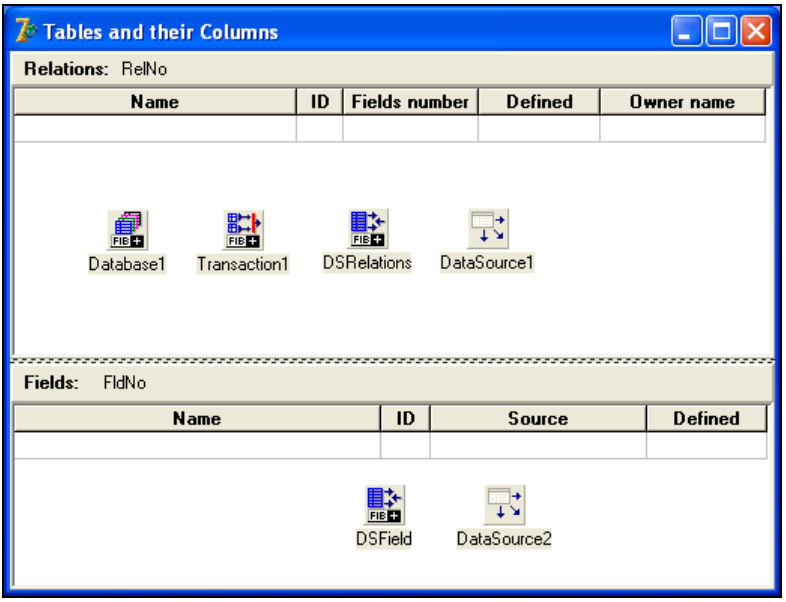


Рис. П4.2. Программа отображения таблиц и представлений

Листинг П4.3. Прорисовка столбца `Defined`

```
procedure TFormMain.DBGridRelationsDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var S: String;
begin
  if Column.FieldName = 'RDB$SYSTEM_FLAG_0' then
  begin
    if DSRelations.FieldByName('RDB$SYSTEM_FLAG').AsString = '1' then
      S := 'System defined'
    else
      S := 'User defined';
    DBGridRelations.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
  end;
end;
```

Запустите программу на выполнение. Вы увидите в списке системных таблиц описания созданных вами таблиц и представлений.

Предложение особо одаренным

Не поленитесь, напишите программу (программы) отображения более полных сведений о таблицах и их столбцах. Для этого вам придется разобраться и с другими системными таблицами, имеющими отношение к данному вопросу, и рассмотреть полные структуры указанных таблиц. Жду ваших гениальных программ.

П4.3. Триггеры

Сведения о триггерах, используемых в базе данных, хранятся в системной таблице `RDB$TRIGGERS`.

П4.3.1. Структура системной таблицы

В табл. П4.6 приведена несколько упрощенная структура этой таблицы.

Таблица П4.6. *Структура таблицы* `RDB$TRIGGERS`

Имя столбца	Тип	Описание
`RDB$TRIGGER_NAME`	`CHAR(31)`	Имя триггера
`RDB$RELATION_NAME`	`CHAR(31)`	Имя таблицы или представления, для которого используется триггер
`RDB$TRIGGER_SEQUENCE`	`SMALLINT`	Последовательность (позиция) триггера
`RDB$TRIGGER_TYPE`	`SMALLINT`	Тип триггера (событие и фаза события): 1 — `BEFORE INSERT`, 2 — `AFTER INSERT`, 3 — `BEFORE UPDATE`, 4 — `AFTER UPDATE`, 5 — `BEFORE DELETE`, 6 — `AFTER DELETE`
`RDB$TRIGGER_SOURCE`	`BLOB TEXT`	Исходный код триггера. Содержит текст определения триггера, включая комментарии
`RDB$TRIGGER_BLR`	`BLOB BLR`	Представление триггера в двоичном коде (в формате BLR)
`RDB$TRIGGER_INACTIVE`	`SMALLINT`	Указывает, является ли триггер активным: 0 — `INACTIVE`, 1 — `ACTIVE`
`RDB$SYSTEM_FLAG`	`SMALLINT`	Указывает, определен триггер пользователем (0) или системой (1)

П4.3.2. Программы просмотра описаний триггеров

Это, пожалуй, самая простая из наших программ для просмотра системных таблиц.

В случае использования компонентов FIBPlus создайте новый проект. Разместите на форме необходимые компоненты.

Форма должна быть приблизительно следующей — рис. П4.3.

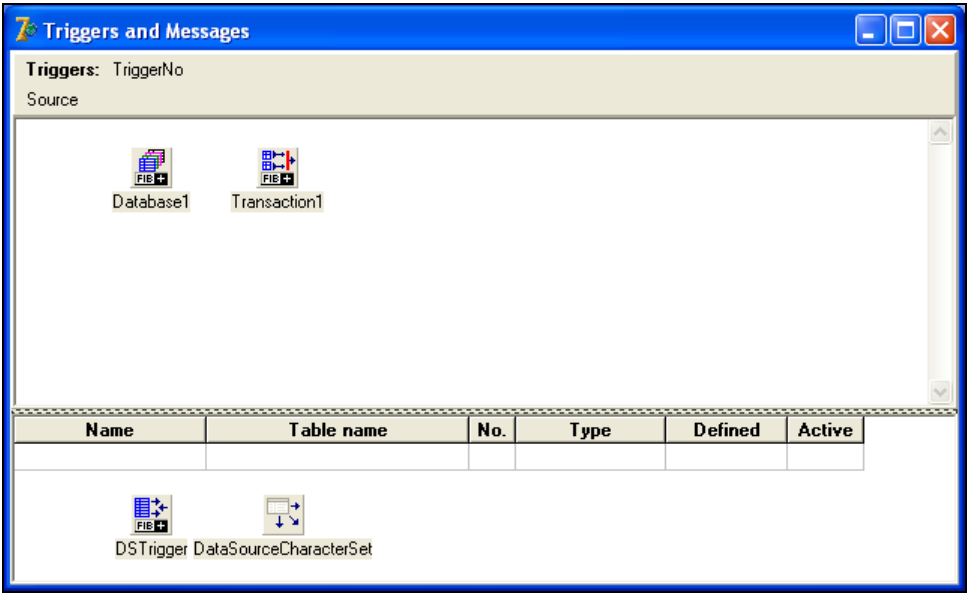


Рис. П4.3. Программа отображения триггеров

Параметры базы данных и транзакции задаются обычным образом.

Для набора данных нужно задать только один оператор `SELECT`:

```
SELECT * FROM RDB$TRIGGERS
```

При отображении формы (событие `OnShow`) необходимо как обычно соединиться с базой данных, открыть набор данных и вывести количество записей в метку `TriggerNo`.

При закрытии формы (`OnClose`) закрывается набор данных и выполняется отключение от базы данных.

В поле `Memo` мы должны выводить значение столбца `RDB$TRIGGER_SOURCE`. Это удобно делать в обработчике события `AfterScroll`, которое возникает при любом перемещении по набору данных, выполняемому пользователем с помощью мыши или клавиатуры (листинг П4.4).

Листинг П4.4. Обработка события `AfterScroll` набора данных

```
procedure TFormMain.DSTriggerAfterScroll(DataSet: TDataSet);
begin
  Memo1.Lines.Text :=
    DSTrigger.FieldByName('RDB$TRIGGER_SOURCE').AsString;
end;
```

Самый объемный код получается для прорисовки ячеек сетки, отображающей набор данных. Нам нужно сформировать тексты, является ли триггер определенным системой, является ли он активным, а также указать, при каком событии срабатывает триггер (листинг П4.5).

Листинг П4.5. Прорисовка столбцов сетки

```
procedure TFormMain.DBGridTriggerDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var S: String;
begin
  if Column.FieldName = 'RDB$SYSTEM_FLAG_0' then
  begin
  if DSTrigger.FieldByName('RDB$SYSTEM_FLAG').AsString = '1' then
    S := 'System defined'
  else
    S := 'User defined';
  DBGridTrigger.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
  end;
  if Column.FieldName = 'RDB$TRIGGER_TYPE0' then
  begin
    if DSTrigger.FieldByName('RDB$TRIGGER_TYPE').AsString = '1' then
      S := 'BEFORE INSERT';
    if DSTrigger.FieldByName('RDB$TRIGGER_TYPE').AsString = '2' then
      S := 'AFTER INSERT';
    if DSTrigger.FieldByName('RDB$TRIGGER_TYPE').AsString = '3' then
      S := 'BEFORE UPDATE';
    if DSTrigger.FieldByName('RDB$TRIGGER_TYPE').AsString = '4' then
```

```
      S := 'AFTER UPDATE';
    if DSTrigger.FieldByName('RDB$TRIGGER_TYPE').AsString = '5' then
      S := 'BEFORE DELETE';
    if DSTrigger.FieldByName('RDB$TRIGGER_TYPE').AsString = '6' then
      S := 'AFTER DELETE';
    DBGridTrigger.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
  end;
  if Column.FieldName = 'RDB$TRIGGER_INACTIVE0' then
  begin
    if DSTrigger.FieldByName('RDB$TRIGGER_INACTIVE').AsString = '0' then
      S := 'Active'
    else
      S := 'Inactive';
    DBGridTrigger.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
  end;
end;
```

Программа с использованием компонентов IBX создается так же просто.

Запустите программу на выполнение. Вы увидите в списке системные и созданные вами триггеры. В поле `Memo` будут появляться тексты триггеров, однако только для созданных пользователем триггеров. Созданные системой триггеры будут содержать свой код в компилированном виде в формате BLR. Средств просмотра такого кода у нас с вами под рукой сейчас нет. Используя IBExpert, вы можете просматривать тексты и в этом формате, правда, пользы от этого я не вижу.

> ***Замечание***
>
> В этих программах я несколько намудрил с количеством панелей, на которых располагаются основные визуальные компоненты. Дело в том, что за основу я взял достаточно давно написанную мною программу (опять же, для англоязычных ребят). Там все было немного сложнее. Уберите все лишние панели, хотя и с ними все работает совершенно нормально.

П4.4. Хранимые процедуры

Последняя группа системных таблиц, которые мы с вами здесь рассмотрим, относится к хранимым процедурам.

Сведения о хранимых процедурах, используемых в базе данных, находятся в системной таблице `RDB$PROCEDURES`. Параметры хранимых процедур, входные и выходные, описаны в системной таблице `RDB$PROCEDURE_PARAMETERS`.

П4.4.1. Структура системных таблиц

В табл. П4.7 и П4.8 приведены несколько упрощенные структуры этих таблиц.

Таблица П4.7. *Структура таблицы* RDB$PROCEDURES

Имя столбца	Тип	Описание
RDB$PROCEDURE_NAME	CHAR(31)	Имя процедуры
RDB$PROCEDURE_ID	SMALLINT	Идентификатор процедуры
RDB$PROCEDURE_INPUTS	SMALLINT	Указывает, существуют входные параметры (1) или нет (0)
RDB$PROCEDURE_OUTPUTS	SMALLINT	Указывает, существуют выходные параметры (1) или нет (0)
RDB$PROCEDURE_SOURCE	BLOB TEXT	Исходный код процедуры
RDB$PROCEDURE_BLR	BLOB BLR	Двоичное представление (BLR) кода процедуры
RDB$OWNER_NAME	VARCHAR(31)	Имя владельца процедуры
RDB$SYSTEM_FLAG	SMALLINT	Определена пользователем (0) или системой (1)

Таблица П4.8. *Структура таблицы* RDB$PROCEDURE_PARAMETERS

Имя столбца	Тип	Описание
RDB$PARAMETER_NAME	CHAR(31)	Имя параметра
RDB$PROCEDURE_NAME	CHAR(31)	Имя процедуры
RDB$PARAMETER_NUMBER	SMALLINT	Последовательный номер параметра
RDB$PARAMETER_TYPE	SMALLINT	Указывает, является ли параметр входным (0) или выходным (1)
RDB$FIELD_SOURCE	CHAR(31)	Сгенерированное системой уникальное имя столбца
RDB$SYSTEM_FLAG	SMALLINT	Указывает, определен параметр системой (1) или пользователем (0)

П4.4.2. Программы просмотра описаний хранимых процедур и их параметров

Напишем две программы отображения сведений о хранимых процедурах.

Здесь мы опять имеем отношение "главная/подчиненная". Главной является системная таблица, описывающая характеристики хранимой процедуры. Подчиненная — описание входных и выходных параметров процедуры.

При использовании компонентов FIBPlus создадим новый проект. Положим нужные компоненты, чтобы получить следующую форму — рис. П4.4.

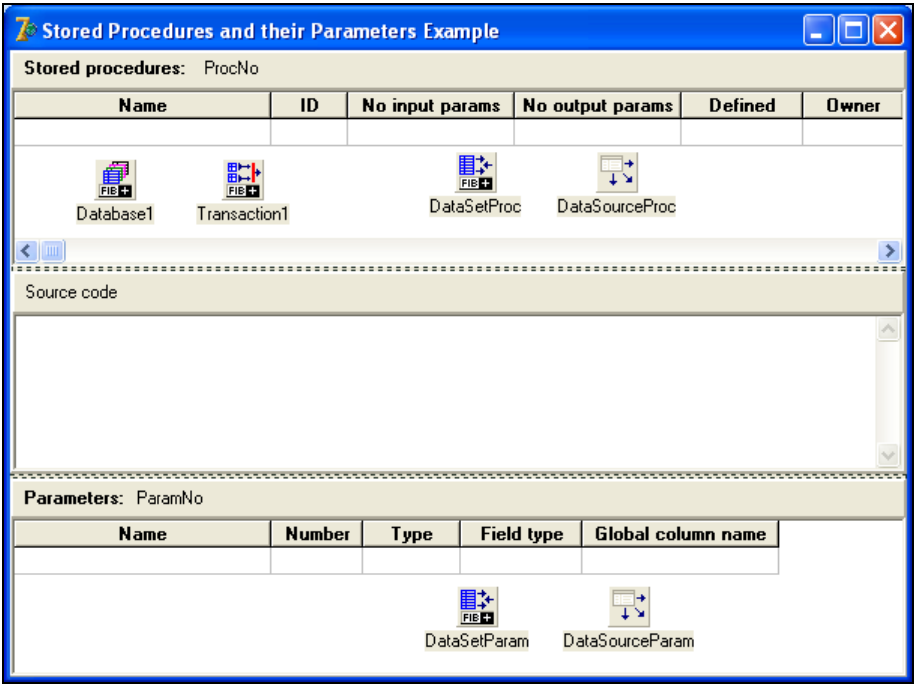


Рис. П4.4. Программа отображения хранимых процедур

Компоненты базы данных и транзакции имеют все те же характеристики, что и в других программах этого приложения.

Для компонента `DataSetProc` формируем оператор `SELECT`:

```
SELECT *
FROM RDB$PROCEDURES
  ORDER BY RDB$PROCEDURE_NAME
```

Здесь (наконец-то) я ввел и предложение `ORDER BY`, чтобы упорядочить список по именам процедур. Предыдущие программы не содержали этого пред-

ложения. Фактически мы получали списки в неявном хронологическом порядке формирования системой (или нами) объектов базы данных. Это тоже может быть интересным для целей исследования.

У компонента `DataSetParam` оператор `SELECT` должен быть таким:

```
SELECT *
  FROM RDB$PROCEDURE_PARAMETERS
WHERE
  RDB$PROCEDURE_NAME = ?RDB$PROCEDURE_NAME
ORDER BY RDB$PARAMETER_TYPE, RDB$PARAMETER_NUMBER
```

Помимо указания связи с родительской таблицей (предложение `WHERE`) здесь мы также задаем и упорядоченность набора данных по типу параметра. Это делается в предложении `ORDER BY` (вначале будут отображаться входные параметры, а затем выходные).

Для формирования осмысленных текстов (System defined или User defined) мы напишем соответствующий обработчик события прорисовки столбцов в первой сетке. Чтобы задавать вид параметра, мы должны написать похожий обработчик и для второй сетки (листинг П4.6).

Листинг П4.6. Прорисовка столбцов второй сетки

```
procedure TFormMain.DBGridParamDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
var S: String;
begin
  if Column.FieldName = 'RDB$SYSTEM_FLAG0' then
  begin
    if DataSetParam.FieldByName('RDB$SYSTEM_FLAG').AsString >= '1' then
      S := 'System defined'
    else
      S := 'User defined';
    DBGridParam.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
  end;
  if Column.FieldName = 'RDB$PARAMETER_TYPE0' then
  begin
    if DataSetParam.FieldByName('RDB$PARAMETER_TYPE').AsString = '0' then
      S := 'Input'
    else
      S := 'Output';
    DBGridParam.Canvas.TextOut(Rect.Left + 3, Rect.Top + 2, S);
  end;
end;
```

Для отображения количества параметров у текущей процедуры нужно написать обработчик события `AfterOpen` для набора данных параметров `DataSetParam` (листинг П4.7).

Листинг П4.7. Обработчик события `AfterOpen`

```
procedure TFormMain.DataSetParamAfterOpen(DataSet: TDataSet);
begin
  ParamNo.Caption := IntToStr(DataSetParam.RecordCount);
end;
```

И, наконец, чтобы в поле `Memo` отображался текст хранимой процедуры, нужен обработчик события `AfterScroll` для набора данных процедур `DataSetProc` (листинг П4.8).

Листинг П4.8. Обработчик события `AfterScroll`

```
procedure TFormMain.DataSetProcAfterScroll(DataSet: TDataSet);
begin
  Memo1.Lines.Text :=
    DataSetProc.FieldByName('RDB$PROCEDURE_SOURCE').AsString;
end;
```

Запустите программу на выполнение. Посмотрите на созданные нами хранимые процедуры.

Создание программы с компонентами IBX также достаточно простое. Все компоненты имеют похожие значения, при этом не забывайте добавлять после открытия набора данных оператор чтения всех записей: `FetchAll`. Необходимо также выполнять открытие и набора данных параметров `DataSetParam` в обработчике события формы `OnShow`.

***

Существует много других системных таблиц. Назначение некоторых понятно. Смысл остальных остается загадкой. Посмотрите в документации по InterBase описания системных таблиц и некоторых полезных представлений (view). Это находится в приложении в документе *Language Reference (Справочник по языкам)*.

ПРИЛОЖЕНИЕ 5

Дополнительные материалы

С сайта БХВ-Петербург вы можете скачать файл в формате zip, в котором содержатся дополнительные материалы, содержащие тексты программ, скрипты работы с базой данных и иконки, которые можно использовать для создания элементов меню и кнопок быстрого доступа в программах.

Все это можно использовать для полного освоения материала этой книги.

Самое простое — размещение написанных в процессе создания книги программ. Все программы находятся в каталоге Book. Далее следует номер главы или приложения. Внутренние каталоги имеют имена, обычно соответствующие именам программ.

В материалах присутствуют следующие каталоги:

- Appendix02. Содержит каталог CharacterSet, в котором присутствуют подкаталоги FIBPlus и IBX. В них находятся программы CharacterSet для просмотра наборов символов и порядков сортировки, существующих в ваших системах управления базами данных;
- Appendix04. В нем находятся каталоги Database, Relations, StoredProc и Triggers, которые содержат подкаталоги, где находятся программы, написанные с использованием компонентов FIBPlus и IBX;
- в каталоге Chapter02 присутствует четыре подкаталога: Backup, CreateDatabase, Restore и UserAccount, содержащие соответствующие программы;
- каталог Chapter06 содержит подкаталоги DisplayDelete, InsertUpdate, MasterDetail и Order, в каждом из которых присутствуют программы, описанные в книге;
- в каталоге Chapter08 находятся подкаталоги SavePoint и Transaction1, содержащие исходные тексты и исполняемые модули соответствующих программ;
- в каталоге Chapter10 находятся подкаталоги Country, MultiBase, People и RichView. В каждом присутствуют тексты нужных программ;

- каталог Icons содержит два десятка иконок в формате BMP, в которых содержатся рисунки, используемые при формировании элементов меню и кнопок быстрого доступа наших программ;
- каталог Scripts содержит все скрипты, необходимые для создания учебной базы данных, всех ее объектов и для заполнения базы данных.

В файле ReadMe.rtf еще раз описывается структура данных дополнительных материалов.

Предметный указатель