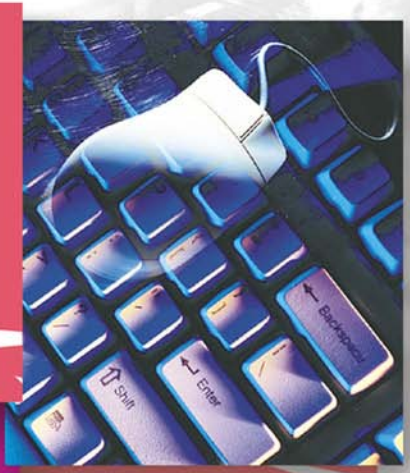
АНДРЕЙ ШКРЫЛЬ

bhv

РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ В DELPHI



СУБД: MS SQL SERVER 2000, Firebird И InterBase

РАБОТА С ГЕНЕРАТОРАМИ ОТЧЕТОВ: QReport, RaveReports, FastReport

АВТОМАТИЗАЦИЯ МОДЕЛИРОВАНИЯ, РАЗРАБОТКИ И ПОДДЕРЖКИ БАЗЫ ДАННЫХ В СРЕДЕ ERwin

ОСОБЕННОСТИ РАЗРАБОТКИ ПРИЛОЖЕНИЙ В DELPHI 7 И DELPHI 2005

ОБЗОР ДОПОЛНИТЕЛЬНЫХ КОМПОНЕНТОВ

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+ ВИДЕОКУРС

Андрей Шкрыль

РАЗРАБОТКА КЛИЕНТ-СЕРВЕРНЫХ ПРИЛОЖЕНИЙ В DELPHI

Санкт-Петербург
«БХВ-Петербург»
2006

УДК 681.3.06
ББК 32.973.26-018.2
Ш66

Шкрыль А. А.

Ш66 Разработка клиент-серверных приложений в Delphi. — СПб.: БХВ-Петербург, 2006. — 480 с.: ил.

ISBN 5-94157-761-3

Рассмотрены практические вопросы по разработке клиент-серверных приложений в среде Delphi 7 и Delphi 2005 с использованием СУБД MS SQL Server 2000, InterBase и Firebird. Приведена информация о теории построения реляционных баз данных и языке SQL. Освещены вопросы эксплуатации и администрирования СУБД. Большое внимание уделено различным генераторам отчетов QReport, RaveReports и FastReport. Описано использование системы проектирования, разработки и поддержки баз данных ERwin. Рассмотрены дополнительные компоненты для разработки клиент-серверных приложений, а также даны ответы на часто задаваемые вопросы. Материал излагается по принципу "от простого к сложному" и сопровождается иллюстрациями, практическими примерами и видеороликами.

На компакт-диске содержатся исходные коды, видеоролики по созданию приложений, а также дополнительные компоненты и инструменты для работы с БД.

Для разработчиков клиент-серверных приложений

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	*Руслан Абдрахманов*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Наталия Першакова*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 19.10.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 38,7.
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 5-94157-761-3

Оглавление

Благодарности

Хочу сказать спасибо своим родителям, благодаря которым я появился на свет, которые привили мне любовь к получению новых знаний и упорство в достижении целей. Брату Антону и сестре Ольге за то, что позволяли пользоваться своим компьютером для написания отдельных глав книги.

Выражаю большую признательность своей жене Наташе, самой умной, доброй и красивой женщине на свете, за ее поддержку и помощь в поиске новых идей. А также благодарю ее за терпение.

Огромная благодарность Чубаркину Артуру Владимировичу, который помог мне не затеряться в этой жизни, благодаря которому я открыл в себе силы написать эту книгу.

Особая благодарность Фленову Михаилу, при содействии которого я совершенствовался как специалист, который помогал мне своими советами и делился опытом. Спасибо тебе, Михаил, без тебя даже не зародилась бы идея написания этой книги.

Хочу поблагодарить отделение Сбербанка России, Автозаводское отделение № 8213 города Тольятти. Особенно отдел информатизации и автоматизации банковских работ, где начиналась моя практика и где я впервые познакомился с MS SQL Server 2000.

Признательность Плахову Сергею, благодаря которому я узнал, про IBExpert и возможности написания серьезных баз данных при помощи FireBird.

Благодарность Кропотову Максиму и Обенко Николаю за их моральную поддержку в процессе моей работы.

Особая благодарность Шишигину Игорю Владимировичу — специалисту с большой буквы — за его оперативные ответы на мои вопросы, профессиональное ведение проекта по написанию книги и просто человеческую поддержку. Также хочется поблагодарить Абдрахманова Руслана за помощь на завершающем этапе работы по созданию книги и всех сотрудников издательства "БХВ-Петербург", которые приняли участие в данном проекте.

Введение

Идея написания данной книги зародилась внезапно. Я давно увлекаюсь темой разработки базы данных, и меня всегда удивлял тот факт, что по данной теме очень мало качественной литературы, где бы доступным и понятным языком объяснялись основы и давались конкретные и, что очень важно, рабочие примеры.

Очень часто в книгах по Delphi в разделе "Работа с базами данных" даются типовые примеры работы с BDE — технологией, которая в настоящий момент уже не поддерживается Borland, — и работы с InterBase. Я не понимаю, зачем продолжать выпускать такие книги, когда в сети очень много информации по этим темам, и к тому же она бесплатная. А как же все остальное, ведь базы данных это не только Paradox и InterBase.

Совершенно случайно, путешествуя по просторам Сети, я нашел информацию о книге Михаила Фленова "Программирование в Delphi глазами хакера", почитал отзывы, и решил ее заказать. И именно с этой книги началась моя карьера программиста и появилась вера в себя как специалиста. Первой моей мыслью было: "Наконец-то появился автор, который объясняет вещи простым и доступным языком". Из этой же книги я узнал, что у Михаила есть собственный сайт **http://www.vr-online.ru**, на котором он выкладывает свои статьи, посвященные компьютерной тематике. И у меня зародилась идея попробовать свои силы и написать собственную статью, так как к тому времени я уже неплохо владел InterBase и MS SQL Server. Первая моя статьей была посвящена работе с InterBase. Причем, что очень меня удивило, статью скачали очень большое количество людей, она стала даже более популярна, чем статьи, которые появились раньше, чем моя. Это натолкнуло меня на мысль, что я не один блуждаю в поисках информации. И я решил продолжить и написал вторую статью, посвященную работе с MS SQL Server.

Вот таким образом, можно сказать, появилась идея написания этой книги.

Я отдаю себе отчет, что многих вопросов не удалось коснуться. Но в книге нельзя рассмотреть абсолютно все, а тем более в книге, охватывающей широкий круг тем, таких как работа с различными СУБД, построение отчетов, проектирование баз данных.

После прочтения данной книги у читателя могут возникнуть дополнительные вопросы, посвященные темам, рассмотренным в ней. Проявление интереса к этим темам, поиск ответов на вопросы и новой информации в рамках этих тем создаст отличные условия для развития читателя как

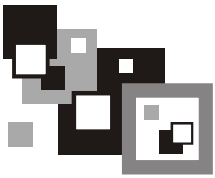
специалиста в области программирования и разработки баз данных. Приобретение дополнительных знаний — необходимое условие для перехода на новый уровень развития с возможностью в дальнейшем самостоятельно отвечать на собственные вопросы и на вопросы других, начинающих.

Хорошая книга в моем понимании — это книга, после прочтения которой читатель понял что-то новое и смог применить эти знания на практике. К тому же, если у читателя возникли конкретные вопросы — это отлично. Значит, он будет развиваться в данном направлении, и время на прочтение книги было потрачено не зря. Часто читателям приходится, как настоящим археологам, сидеть над каждой строчкой кода и сравнивать то, что написано в книге, с теми примерами, поставляемыми на диске к книге, искать ошибки. А все дело оказывается в том, что упущены какие-то мелочи, несущественные для автора, но для читателя имеющие огромное значение. Предусматривая это, я решил для каждого примера и для каждой версии Delphi, рассмотренных в книге (Delphi 7 и Delphi 2005), записать видеоролик, чтобы читатель мог просмотреть, если ему вдруг непонятен какой-то момент, а ни в книге, ни в исходных текстах нет объяснений.

Вообще меня поражает тот факт, что в наш век высоких технологий, когда мультимедиа давно используется во всех сферах деятельности, к книгам это до сих пор не применяется. Я считаю, что большое преимущество этой книги в том, что есть наглядные материалы (видеоролики), а также существует форум, куда читатель может обратиться, и ему обязательно помогут. Таким образом, диалог автора с читателем после прочтения книги не заканчивается, он может продолжиться, если у читателя появится желание сделать это во Всемирной сети, где я и остальные участники форума, а также члены команды VR-TEAM, с радостью поделятся своими знаниями и опытом.

Приятного прочтения.

Глава 1



Введение в реляционные базы данных

1.1. Вступление

Реляционные базы данных имеют мощный теоретический фундамент, основанный на математической теории отношений. Появление теории реляционных баз данных дало толчок к разработке языка запросов SQL.

В реляционной модели объекты реального мира и взаимосвязи между ними представляются с помощью совокупности связанных между собой таблиц или отношений.

Свое название реляционная теория заимствовала из математики. Основатель реляционной теории — доктор Э. Ф. Кодд.

1.2. База данных и ее составные части

Базы данных используются в тех случаях, когда возникает необходимость манипулировать большими объемами данных. У базы данных есть имя. Она состоит из таблиц, в которых хранятся данные, и из дополнительных элементов, поддерживающих работоспособность базы данных, а также достоверность содержащейся в ней информации (рис. 1.1).

Рассмотрим более подробно элементы базы данных.

- *Таблицы* — это набор обычных двухмерных таблиц, знакомых всем из школьного курса. У таблицы есть два параметра — *строки* и *столбцы*. Строки по реляционной теории называются *кортежами*, но, как правило, в настоящее время используют более распространенный термин — *запись*. Столбцы называются *атрибутами*. У каждого поля (атрибута) есть свой *тип данных*. *Тип данных* — набор значений, который может принимать атрибут. Например, числовой тип позволяет использовать только числа; логический — да, нет; текстовый — символы и т. д.

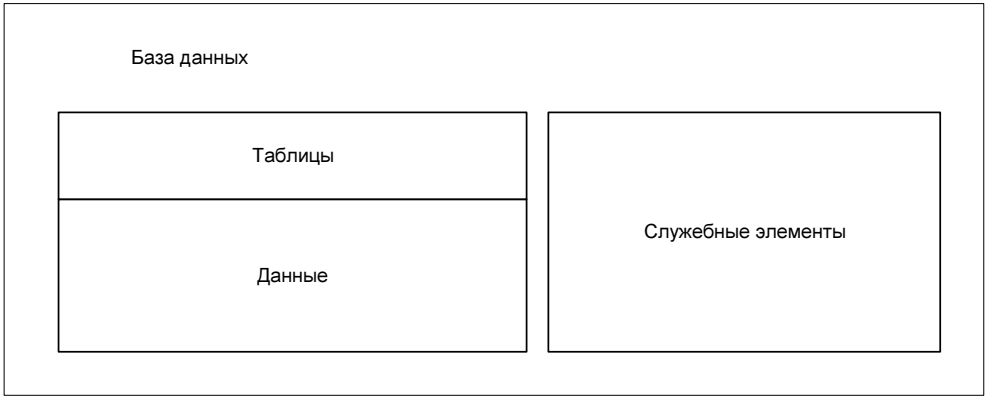


Рис. 1.1. Структура базы данных

У каждой таблицы есть имя, также имя есть и у каждого атрибута таблицы. Таким образом, можно легко обратиться к нужному элементу базы данных, как показано на рис. 1.2 и 1.3.

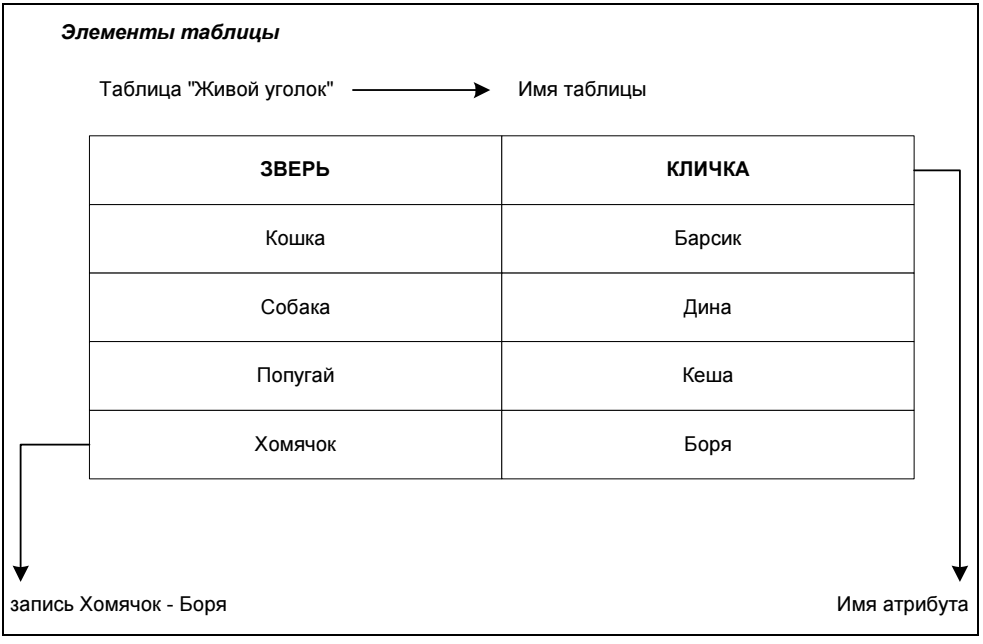


Рис. 1.2. Элементы таблицы

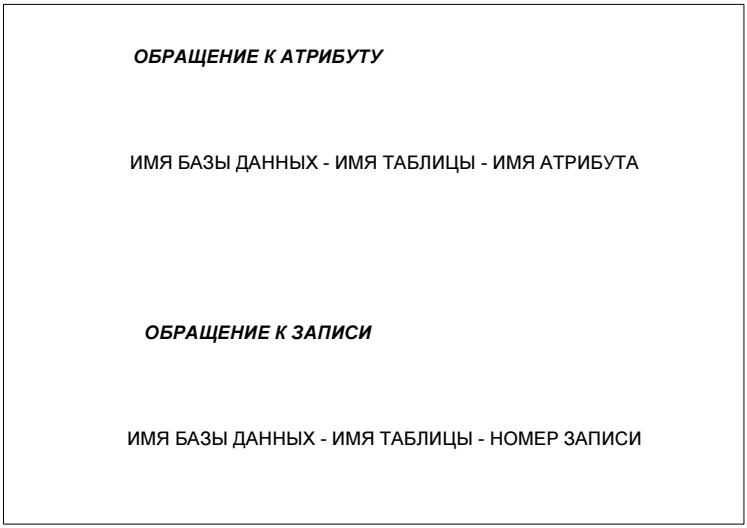


Рис. 1.3. Обращение к элементам таблицы

Вывод

Таблицы и атрибуты одной и той же таблицы не должны иметь одинаковые имена, иначе возникнет неоднозначность.

- *Данные* — это та информация, которую внес пользователь и которая хранится в таблицах. При проектировании таблиц важно обеспечить непротиворечивость хранящейся в ней информации. Рассмотрим это на примере (табл. 1.1).

Таблица 1.1. *Таблица "Персонал"*

Имя	**Отчество**	**Фамилия**
Вася	Васильевич	Васечкин
Вася	Васильевич	Васечкин
Миша	Андреевич	Кузькин

Существует некая таблица "Персонал", в ней содержатся имена людей, работающих в небольшой фирме. Предположим, что в этой фирме работают отец и сын с одинаковыми именами. Как определить, глядя на эту таблицу, какая запись кому соответствует?

Для решения этой проблемы в реляционной теории введено понятие *первичного ключа* — атрибута или комбинации атрибутов, являющейся уникальной

для всех записей таблицы. Если ключ содержит только одно поле, то это поле называется *ключевым*. Предназначением ключевого поля для данных в табл. 1.2 является обеспечение уникальности каждой записи.

Таблица 1.2. *Таблица "Персонал" с ключевым полем*

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин

В данном примере информацию уже можно различить, так как было добавлено дополнительное поле. Например, можно сказать, что запись с номером 1 соответствует отцу, а запись с номером 2 — сыну.

Таблица 1.3. *Таблица "Персонал" с новыми записями*

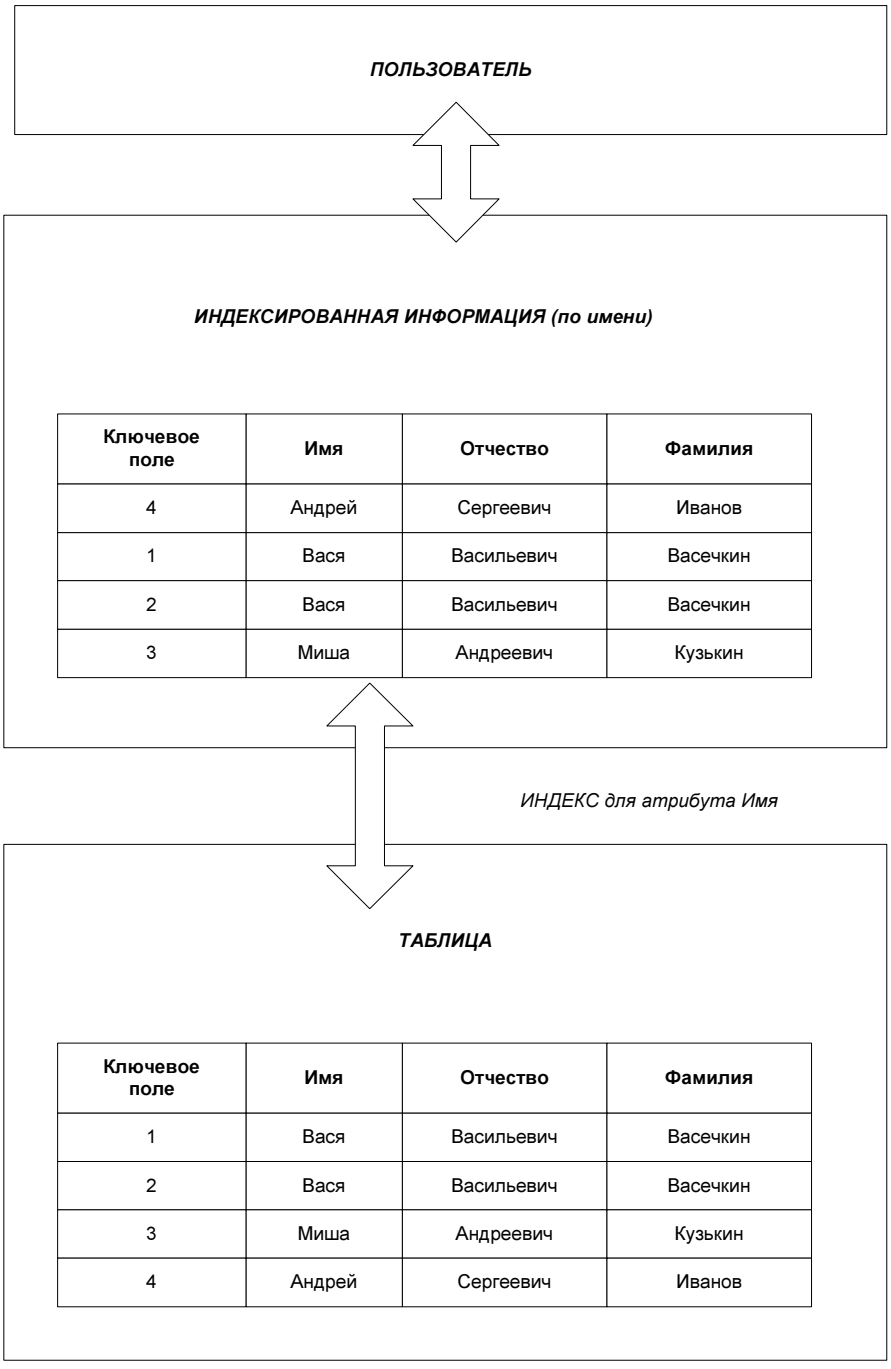
Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Когда данные табл. 1.3 будут отображаться пользователю, ему будет намного удобней, если данные предварительно будут упорядочены, например, по имени сотрудника.

В реляционной теории существует понятие *индексов*. Индекс является промежуточным звеном между пользователем и таблицей и обеспечивает более быстрый доступ к данным. Один из самых распространенных вариантов использования индексов при разработке клиентского приложения в Delphi — это упорядочивание информации, то есть для того поля, значения которого будут упорядочиваться (производиться сортировка), создается индекс (рис. 1.4).

Индексы могут быть *первичными* и *вторичными*.

- Первичный индекс — это ключевое поле, оно автоматически индексируется. Первичный индекс может быть:
  - простым — первичный индекс состоит из одного атрибута (ключевое поле — яркий пример простого первичного индекса);



Ключевое поле	Имя	Отчество	Фамилия
4	Андрей	Сергеевич	Иванов
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Рис. 1.4. Принцип работы индекса при использовании его для сортировки данных

- составным — первичный индекс состоит из нескольких атрибутов (можно создать составной первичный индекс, состоящий из атрибутов Имя — Отчество — Фамилия).

❒ Вторичный индекс — это любое неключевое поле (набор полей), для которого создан индекс. Вторичный индекс, так же как и поле, которому он принадлежит, имеет имя.

1.3. Создание связей между таблицами

Уже упоминалось о том, что база данных это набор таблиц. Так вот, сами по себе таблицы представляют небольшой интерес. Всю мощь использования баз данных можно оценить, если извлекать информацию из нескольких таблиц одновременно. Как правило, для этого таблицы предварительно *связывают,* хотя это не является обязательным условием.

При организации связи одна из таблиц будет главной, а другая — второстепенной. Для того чтобы связать две таблицы, используется ключевое поле (его еще называют ключом) и индекс.

Рассмотрим таблицу "Сотрудники" (табл. 1.4) и таблицу "Зарплата" (табл. 1.5).

Таблица 1.4. *Таблица "Сотрудники"*

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Таблица 1.5. *Таблица "Зарплата"*

Ключевое поле	Дата получения	Сумма	Кто получил (код сотрудника)
1	10.10.2005	10 000	2
2	13.10.2005	5000	1
3	15.10.2005	7000	2
4	11.10.2005	18 000	3

Зарплата не может существовать сама по себе, она всегда выплачивается кому-то, а ситуация, когда сотрудник не получает зарплату, вполне реальная.

На основании этого можно сделать вывод, что таблица "Сотрудники" — главная, а таблица "Зарплата" — второстепенная.

Для организации связи в таблицу "Зарплата" добавляется атрибут "Кто получил", который содержит код сотрудника, полученный из атрибута "Ключевое поле" таблицы "Сотрудники". Таким образом, можно сказать, что некий сотрудник Вася Васильевич Васечкин с кодом 2 получал зарплату дважды, а сотрудник с кодом 4 — Андрей Сергеевич Иванов — не получил еще ни разу (рис. 1.5).

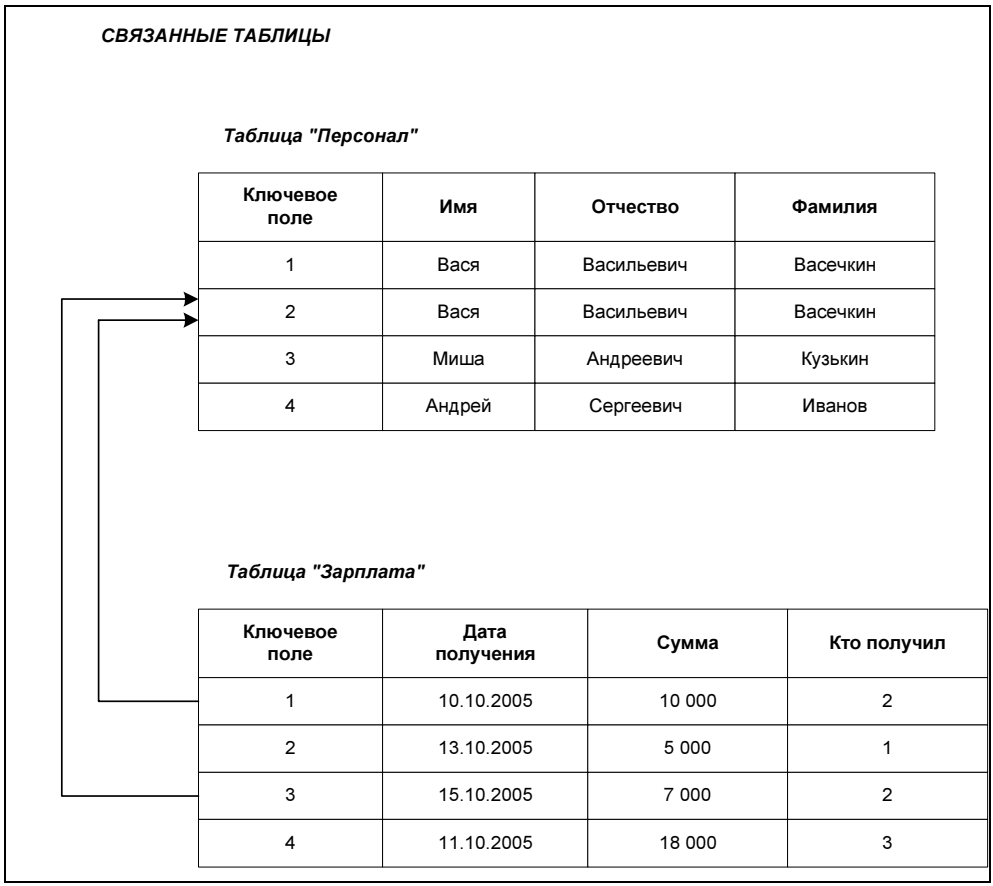


Рис. 1.5. Связанные таблицы

Так как таблицы отражают объекты реального мира, а, как известно, объекты могут взаимодействовать друг с другом, то можно сказать, что между таблицами есть *отношения* (связь).

В реляционной теории существует три типа отношений (связей):

- Один-к-одному — наиболее редко встречающийся тип, когда одной записи в главной таблице соответствует одна запись в подчиненной (рис. 1.6).

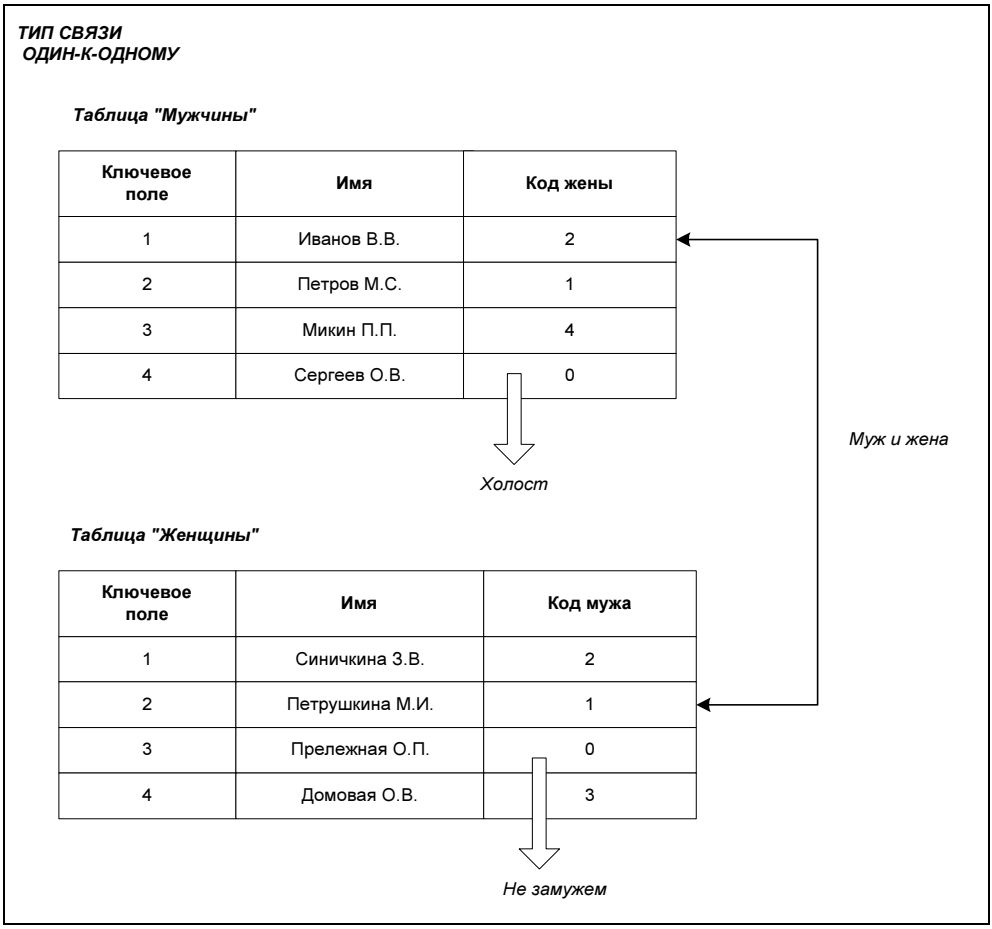


Рис. 1.6. Тип связи один-к-одному

В России многоженство запрещено, поэтому у одного мужчины может быть в один момент времени только одна жена, также и у женщины в один момент может быть только один муж. Связь между таблицами реализуется добавлением в обе таблицы по дополнительному атрибуту. В таблицу "Мужчины" — атрибута "Код жены", в таблицу "Женщины" — атрибута "Код мужа". Также данный тип связи можно реализовать, доба-

вив дополнительную таблицу "Отношения", в которой будут присутствовать два атрибута: "Код мужчины" и "Код женщины".

- Один-ко-многим — наиболее распространенный тип связи, когда одной записи в главной таблице соответствуют одна или несколько записей во второстепенной таблице (рис. 1.7).

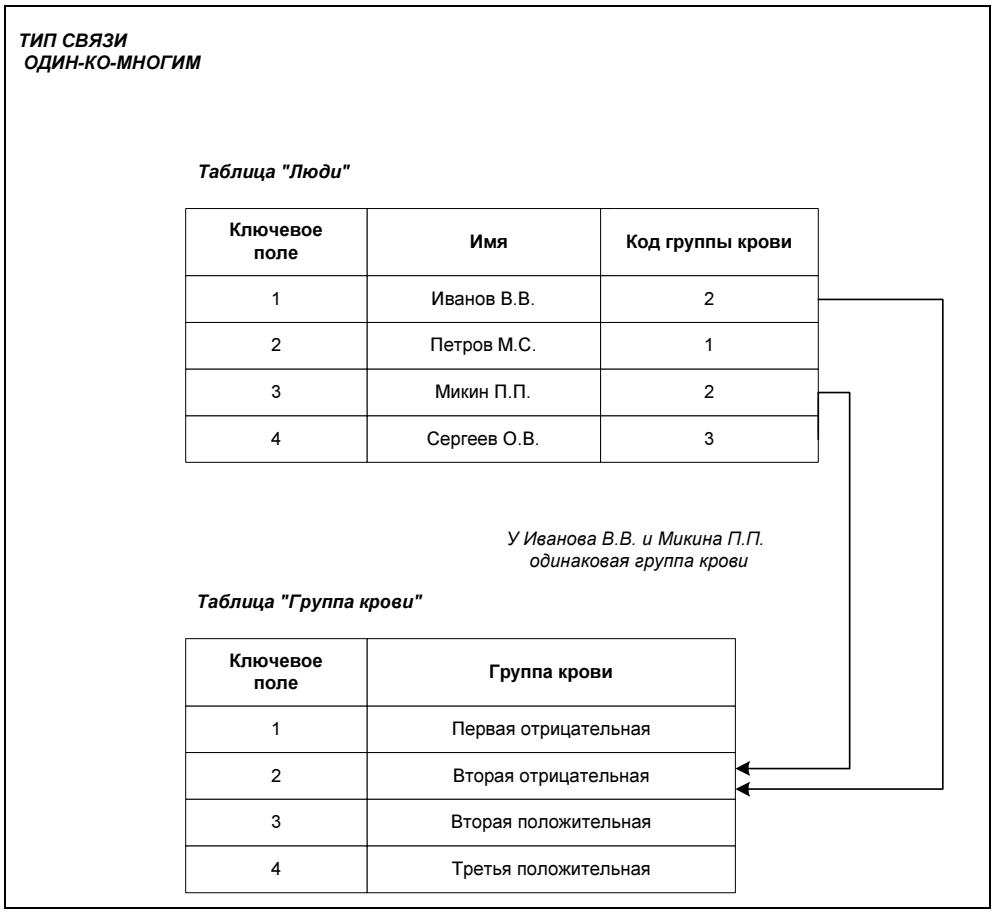


Рис. 1.7. Тип связи один-ко-многим

У человека может быть только одна группа крови, но одна группа крови может одновременно принадлежать множеству людей. Для организации данного типа связи в таблицу "Люди" добавляется дополнительный атрибут "Код группы крови", содержащий ссылку на запись в таблице "Группа крови".

- Многие-ко-многим — тип связи, когда множеству записей в главной таблице соответствует множество записей из второстепенной таблицы. Этот тип связи самый сложный, и для его реализации вводят дополнительную таблицу. Для примера рассмотрим отношения, складывающиеся между пациентом и врачом в больнице (рис. 1.8).

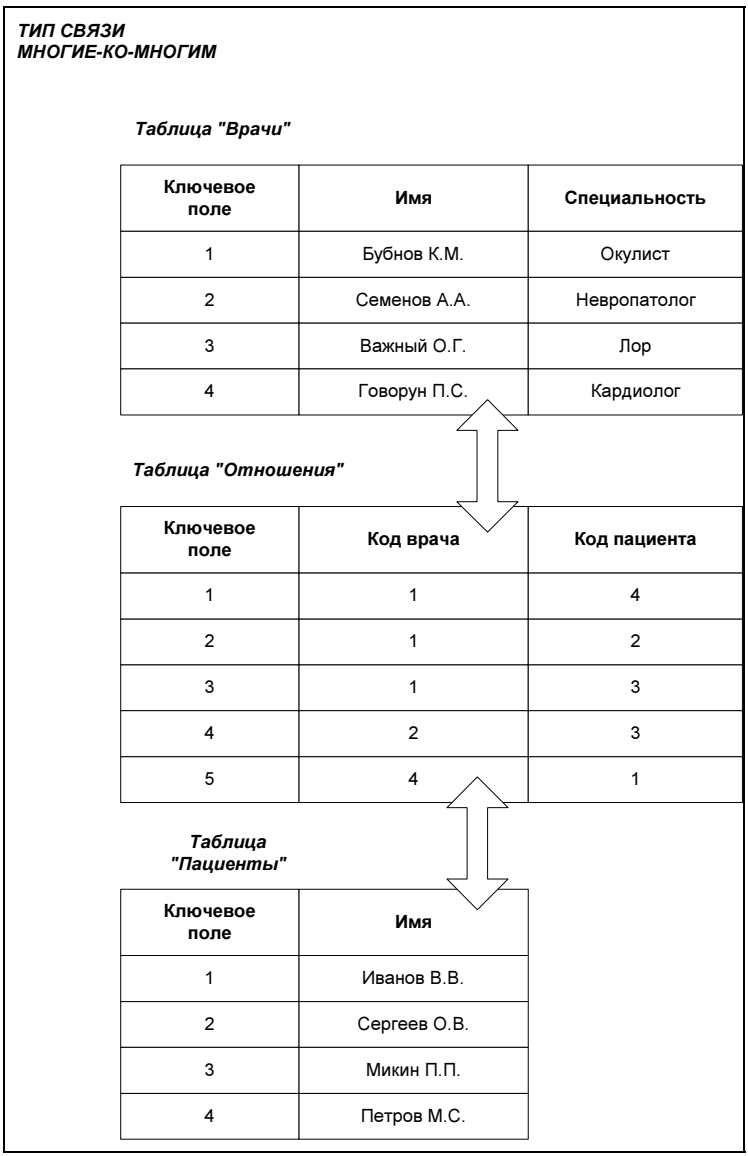


Рис. 1.8. Тип связи многие-ко-многим

Один пациент может записаться на прием к нескольким врачам, также аналогичная ситуация и с врачами. К одному врачу может быть записано множество пациентов.

Таким образом, получается, что множество пациентов может быть записано к множеству врачей, или у множества врачей может быть множество пациентов.

Для того чтобы реализовать эти отношения, нужно ввести дополнительную таблицу "Отношения", в которой будут прописываться отношения между пациентом и врачом. Например, по поводу рис. 1.8 можно сказать следующее:

- к окулисту записано трое пациентов: Сергеев О. В., Петров М. С., Микин П. П.;
- к невропатологу записан один пациент: Микин П. П., записанный также и к окулисту;
- к кардиологу записан один пациент: Иванов В. В.;
- к лору не записан никто.

1.4. Служебные элементы базы данных

Теперь рассмотрим *служебные элементы* базы данных (у всех этих элементов, так же как у таблиц и индексов, есть имя).

- Представление — это виртуальная таблица, сама по себе не хранит информацию, а является результатом выполнения некоторого запроса (о запросах мы поговорим во *главе 2*). В основном используется для обеспечения безопасности (пользователю показывается только то, что нужно) и для формирования отчетов (рис. 1.9).
- Хранимая процедура — этот объект представлен набором команд (об этом более подробно в следующей главе). Хранимая процедура не содержит никакой информации из базы данных, вместо этого она содержит команды о том, что нужно сделать с данными (добавить, удалить, произвести обработку) или в каком виде представить их пользователю (рис. 1.10).
- Триггер — это та же самая процедура, только выполнение ее происходит автоматически при выполнении некоего события, связанного с таблицей. Например, при использовании СУБД InterBase (определение понятия СУБД будет дано ниже), когда происходит добавление записи в таблицу, возникнет событие `Before_Insert`.

  По аналогии с триггером работают продавцы в магазинах. Они продадут вам сигареты и спиртные напитки, если вам больше 18 лет, в противном случае в продаже вам будет отказано. В данном случае в роли события выступает желание покупателя сделать покупку, в роли триггера — продавец.

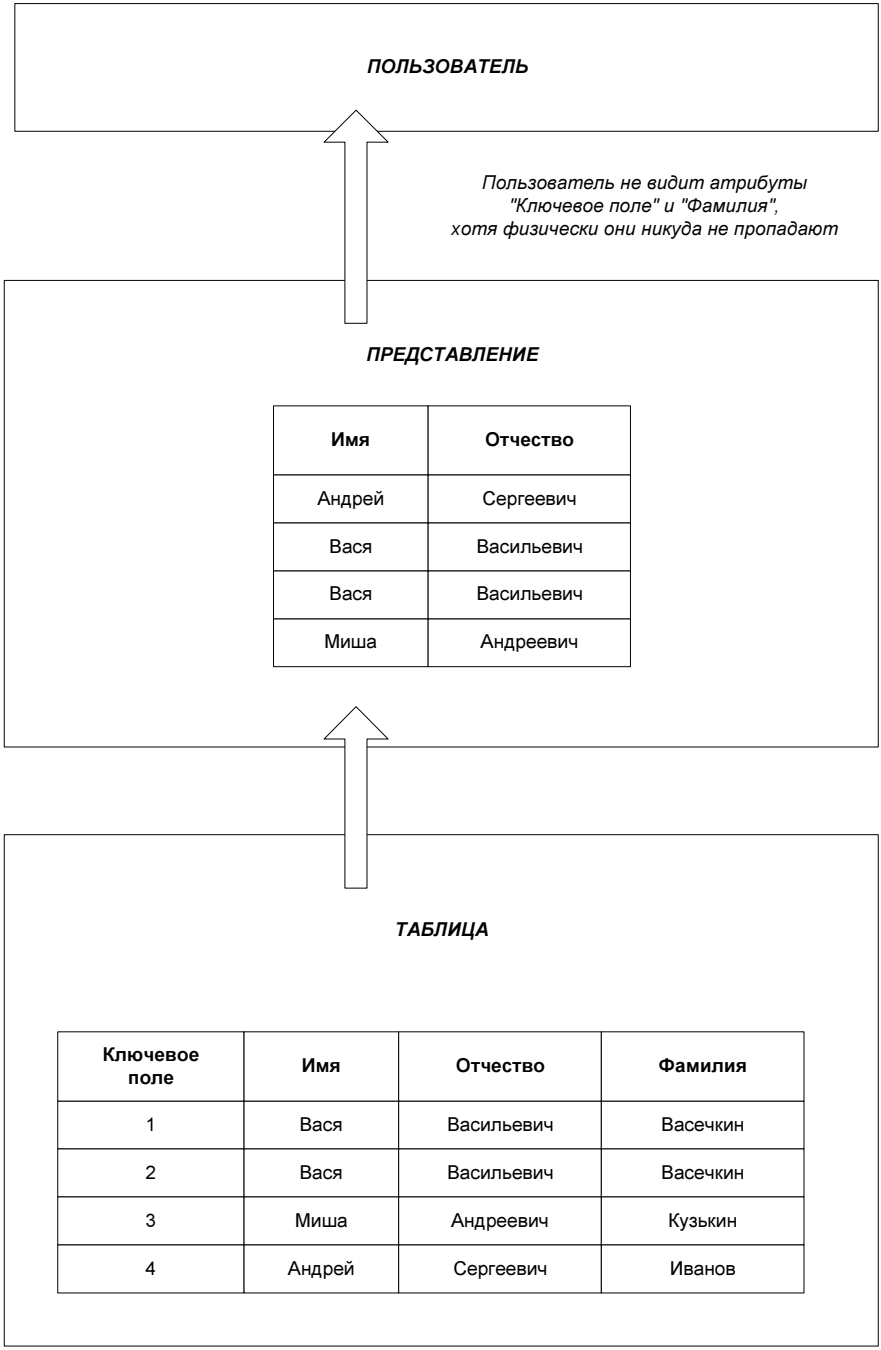


Рис. 1.9. Принцип работы представления

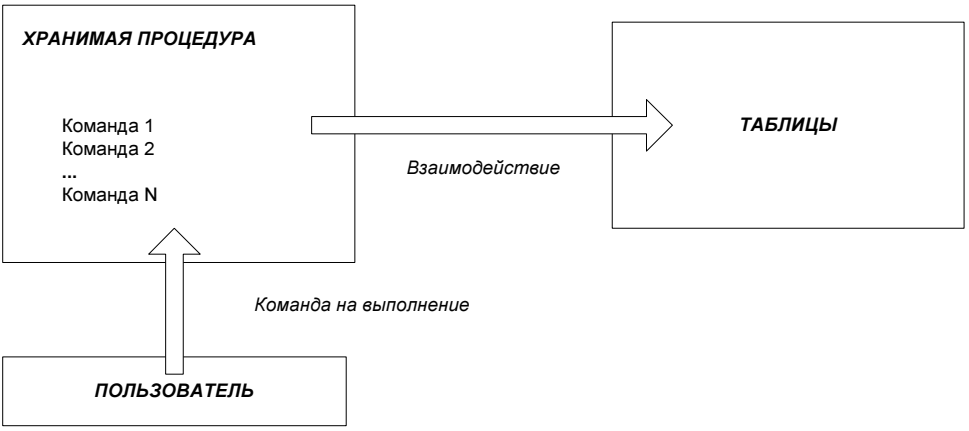


Рис. 1.10. Принцип работы хранимой процедуры

Опишем эту ситуацию в виде в виде *алгоритма*.

1. Покупатель хочет совершить покупку сигарет.
2. Возникает событие — покупка.
3. Управление передается продавцу (триггеру), который смотрит на покупателя и пытается определить, исполнилось ли ему 18 лет или нет.
4. Если исполнилось, то покупка осуществляется.
5. Если покупатель слишком молод, то продавец (триггер) ничего не продает.

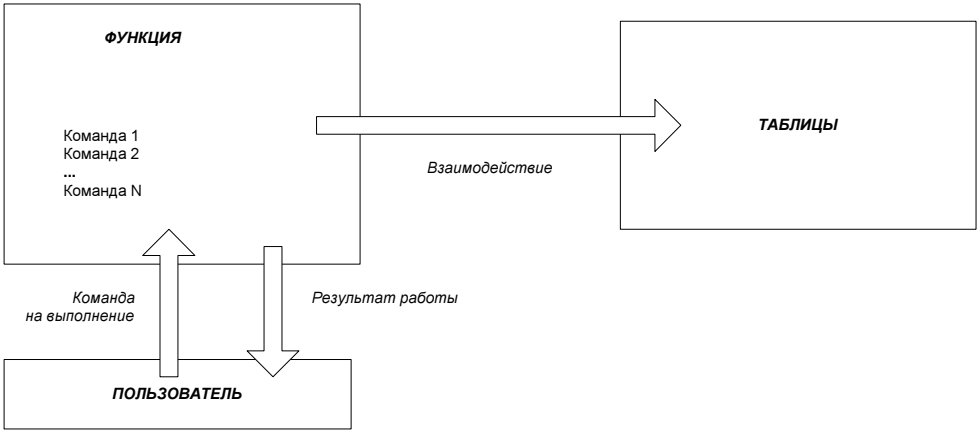


Рис. 1.11. Принцип работы функции

- Функция — предоставляет один из способов манипулирования данными. Функция — это, так же как и процедура, набор команд. Главное отличие, что функция всегда возвращает результат (рис. 1.11).
- Генератор — это элемент, предназначенный для построения *первичных ключей*. Генератор сначала создается, потом ему задается начальное значение, далее он присваивается атрибуту таблицы. Далее, чтобы генератор заработал, создается триггер для события "Вставка новой записи" (при использовании СУБД InterBase этим событием будет `Before_Insert`). И каждый раз, когда в таблицу добавляется новая запись, предварительно идет обращение к генератору, который в свою очередь выдает записи уникальный номер.

1.5. Основные принципы разработки базы данных

Теперь мы рассмотрим к чему разработчик или проектировщик базы данных должен стремиться в своей работе, это *основные правила* построения качественной базы данных.

- *Доступность*. База данных должна быть спроектирована так, чтобы данные, хранящиеся в ней, были доступны пользователю в любое время. База данных должна быть защищена от сбоев и непредвиденных ситуаций. Если в работе базы данных все же случился сбой, то он должен быть устранен в минимальные сроки и посредством специалиста или специалистов, обслуживающих данную базу.
- *Расширяемость*. База данных, как правило, отражает отношения чего-либо (товарно-денежные отношения людей) либо хранит информацию о каких-то объектах (данные по продажам, паспортные данные людей и т. д.). Со временем могут потребоваться изменения в базе данных, либо расширилось количество данных, которые требуется хранить, либо изменились правила отношений между людьми. Можно приводить кучу примеров причин, из-за которых возникает необходимость в изменении структуры базы данных. Принцип расширяемости регламентирует процесс внесения изменений в структуру базы данных. По времени он должен быть минимальным, а качество отраженных изменений — максимально.
- *Непротиворечивость*. База данных должна быть спроектирована таким образом, чтобы была исключена возможность возникновения в ней противоречивой информации. Например, если существует таблица "Персонал" (см. табл. 1.1) и в ней содержатся две абсолютно одинаковые записи (Вася Васильевич Васечкин), то можно утверждать, что условие непротиворечивости нарушено.

1.6. Транзакции

Транзакция — это неделимый блок действий (совокупность команд) с базой данных, который можно либо целиком выполнить, либо не выполнить вообще. Как правило, говорят, что транзакцию можно либо подтвердить, либо откатить.

Принцип работы транзакции можно описать следующим образом.

1. Необходимо внести данные в базу данных.
2. Начинаем транзакцию.
3. Вносим изменения.
4. Пытаемся завершить транзакцию.
   - Если завершить удалось, то все изменения успешно сохраняются.
   - Если произошла ошибка, то отказываемся от всех изменений.
5. Если передумали вносить изменения, то пропускаем пункт 4 и сразу откатываем транзакцию.

Например, у человека на кредитной карточке имеется 1000 рублей и карточка не может иметь отрицательный баланс (количество денежных средств с минусом). Предположим, что он хочет оплатить покупку в магазине на сумму 700 рублей и в это же время у него с карточки снимается 500 рублей за квартплату. Если не использовать транзакции, то баланс будет отрицательным (−200 рублей), а это не допустимо. Транзакции помогают защитить базу данных от ошибок. Предположим, что первой снялась сумма 500 рублей. Тогда человек не сможет оплатить покупку в магазине, потому что транзакция будет откатываться, так как условие отрицательного баланса будет вызывать ошибку.

1.7. Типы данных

По реляционной теории необходимо, чтобы типы используемых данных были простыми. Простые типы данных — это такие, которые не обладают внутренней структурой. К простым типам относятся:

- ❒ Логический
- ❒ Строковый
- ❒ Числовой

1.8. Нормализация

Нормализация — это процесс улучшения качества отношений между таблицами и, как следствие, свойств базы данных в целом. Основополагающим принципом является наложение некоторых ограничений и правил на структуру таблиц в базе данных.

В теории реляционных баз данных выделяются следующие *нормальные формы*:

1. Первая нормальная форма (1NF).
2. Вторая нормальная форма (2NF).
3. Третья нормальная форма (3NF).
4. Нормальная форма Бойса—Кодда (BCNF).
5. Четвертая нормальная форма (4NF).
6. Пятая нормальная форма (5NF).

Мы рассмотрим только первые три. Вызвано это тем, что приведение базы данных к третьей нормальной форме является достаточным условием для того, чтобы база данных соответствовала основным правилам построения качественной базы данных, которые были рассмотрены в *разд. 1.5* (доступность, расширяемость, непротиворечивость). Четвертая и пятая нормальные формы применяются редко и на данном этапе изучения для нас не представляют интерес.

Таблица находится в 1НФ, если все ее атрибуты содержат только данные простых типов, т. е. атрибут не может хранить данные структурированного типа (рис. 1.12).

ПЕРВАЯ НОРМАЛЬНАЯ ФОРМА

Имя	Отчество	Фамилия
Вася	Васильевич	Васечкин
Вася	Васильевич	Васечкин
Миша	Андреевич	Кузькин
Андрей	Сергеевич	Иванов

Рис. 1.12. Таблица "Персонал", приведенная к первой нормальной форме

В таблице "Персонал" (см. рис. 1.12) каждый атрибут содержит данные только простого типа (все атрибуты имеют строковый или символьный тип данных).

Таблица находится во 2НФ, если она находится в 1НФ и каждый ее *неключевой* атрибут (тот, который не входит в состав первичного ключа) полностью зависит от первичного ключа (рис. 1.13).

ВТОРАЯ НОРМАЛЬНАЯ ФОРМА

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Рис. 1.13. Таблица "Персонал", приведенная ко второй нормальной форме

На рис. 1.13 видно, что в таблицу "Персонал" добавлен дополнительный атрибут "Ключевое поле"; таким образом, мы исключаем ситуацию, когда в таблице будут присутствовать одинаковые записи и теперь каждый атрибут таблицы зависит от первичного ключа (атрибут "Ключевое поле"). Таким образом, 2НФ накладывает условие, что в каждой таблице должен быть первичный ключ.

Таблица находится в 3НФ, если она находится во 2НФ и каждый неключевой атрибут *нетранзитивно* зависит от первичного ключа.

Сейчас все разберем на примере. Имеется 2 таблицы (рис. 1.14). В таблице "Персонал" хранятся данные людей: фамилия, имя и отчество. В таблице "Отделы" хранятся данные по отделам и по персоналу, который в них работает.

Поговорим о недостатках этой базы данных (рис. 1.14). Во-первых, в таблице "Отделы" явно содержится избыточная информация. Получается, что имя, отчество и фамилия о человеке хранятся сразу в двух таблицах. Во-вторых, если возникнет ситуация, что у сотрудника изменилась фамилия, например, сменил он ее, то данные придется изменять в обеих таблицах. В-третьих, если создать запись в таблице "Отделы" без указания конкретного сотрудника, то атрибуты "Имя", "Отчество", "Фамилия" окажутся пустыми и

таким образом получится, что в отделе работает сотрудник, которого нет в таблице "Персонал". Другим словами, несмотря на то, что *неключевые атрибуты* обеих таблиц полностью зависят от первичного ключа (то есть таблицы приведены к 2НФ), атрибуты таблицы "Отделы" еще к тому же находятся в зависимости от атрибутов таблицы "Персонал". Такая зависимость называется транзитивной.

Таблица "Персонал"

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Таблица "Отдел"

Ключевое поле	Номер отдела	Имя	Отчество	Фамилия
1	отдел № 315	Вася	Васильевич	Васечкин
2	отдел № 808	Вася	Васильевич	Васечкин
3	отдел № 315	Миша	Андреевич	Кузькин
4	отдел № 905	Андрей	Сергеевич	Иванов
5	отдел № 111			

Сотрудник-невидимка

Рис. 1.14. Таблицы "Персонал" и "Отделы"

Приведем таблицы к 3НФ (рис. 1.15).

ТРЕТЬЯ НОРМАЛЬНАЯ ФОРМА

Таблица "Персонал"

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Таблица "Размещение сотрудников"

Ключевое поле	Код сотрудника	Код отдела
1	1	1
2	2	2
3	3	1
4	4	3

Таблица "Отделы"

Ключевое поле	Номер отдела
1	отдел № 315
2	отдел № 808
3	отдел № 905

Рис. 1.15. Пример 3НФ

В результате небольших изменений преодолены ограничения, накладываемые предыдущим вариантом базы данных (рис. 1.15). Могут быть как отделы без сотрудников, например, вновь отремонтированные помещения или

купленные. Так и сотрудники, не размещенные ни в одном из отделов, например, только что принятые. Таким образом, возможности этой базы данных расширились и исчезла транзитивная зависимость. Связь между таблицами "Персонал" и "Отделы" регламентирует таблица "Размещение сотрудников", в которой хранится информация о том, какой сотрудник и в каком отделе размещен.

1.9. Системы управления базами данных (СУБД)

СУБД (система управления базами данных) — это программное обеспечение, предоставляющее пользователю механизм создания и поддержки базы данных, а также возможность получать к ней доступ.

Процесс выбора типа СУБД называется *масштабированием*.

Существуют следующие модели баз данных.

- Автономная база хранит все данные в локальной файловой системе компьютера. Клиентское приложение и система управления базой расположены на одном компьютере. Этот тип базы данных используется для тех случаев, когда каждому пользователю нужно поддерживать отдельную базу, например, база "Адресная книга" — яркий пример автономной (локальной) базы данных.
- Файл-серверная база данных доступна одновременно нескольким пользователям через сеть. Физически база данных располагается на сервере. Для каждого пользователя, желающего поработать с базой, создается локальная копия данных, то есть сначала вся база копируется к пользователю на компьютер, а потом он уже начинает с ней работать. После того как пользователь поработал, происходит процесс сверки обоих баз. Все изменения помещаются в базу на сервер. При этом возникает проблема передачи данных. Чем больше данных в базе, тем больше информации приходится тащить по сети; таким образом, сеть вообще может умереть.
- Клиент-серверная база данных физически располагается на сервере. Клиент, он же пользователь, посылает базе данных команды (запросы) и просит предоставить информацию, которая ему необходима. При использовании этой технологии не возникает проблема загрузки сети, но возникает *проблема параллельного доступа*. Суть данной проблемы заключается в том, что несколько клиентов (пользователей) пытаются одновременно изменить одну и ту же запись. Данная проблема решается с помощью *блокировок*. При изменении определенной записи она блокируется до того времени, пока изменения не будут подтверждены; таким образом, в это время никто другой не сможет изменять данные, их можно будет только просмотреть.

- Трехзвенная архитектура состоит из трех уровней. Первый — сервер баз данных, к которому поступают запросы клиентов, который возвращает им информацию, следит за работоспособностью базы данных. Второй — это сервер приложений, а третий — пользовательские компьютеры.

  Основное преимущество данной архитектуры состоит в том, что пользователям необходимо только подключиться к серверу приложений и больше их ничего не должно беспокоить. Все драйвера доступа к базе данных находятся на сервере приложений. К тому же решается проблема обновления программного обеспечения, данную операцию достаточно произвести единожды на сервере приложений, и все пользователи будут работать с новой версией программы.

Важно понимать различия между моделями баз данных и СУБД.

- Модель баз данных — это теоретически обоснованная технология разработки и построения базы данных.
- СУБД — это практическая реализация выбранной модели базы данных.

1.10. Целостность данных

В системах управления базами данных есть такое понятие, как *целостность данных*. Оно подразумевает под собой установку правил для элементов базы данных, в том числе и для атрибутов связанных таблиц. При изменении данных атрибута одной таблицы эти изменения сразу сказываются на данных, содержащихся в атрибутах таблицы, участвующей в связи. Под изменением данных подразумевается создание новых записей, изменение или удаление уже существующих.

Наиболее распространены следующие способы поддержания ссылочной целостности данных:

- *каскадное обновление связанных полей* указывает на необходимость обновлять значение внешнего ключа каждый раз, когда изменяется соответствующее ему значение первичного ключа. Например, имеются таблица "Персонал" и таблица "Зарплата". В таблице "Зарплата" каждая запись привязывается к определенному сотруднику по его коду в таблице "Персонал". Если код сотрудника в таблице "Персонал" изменится, то автоматически этот код изменится и в таблице "Зарплата" для этого сотрудника; таким образом, данные о зарплате не потеряются (рис. 1.16).
- *Каскадное удаление связанных полей* указывает на необходимость удаления всех записей в таблице с внешним ключом при удалении соответствующей записи из таблицы с первичным ключом. Например, если мы удалим запись из таблицы "Персонал", то удалятся все записи по зарплате этого человека в таблице "Зарплата".
- Запрет на удаление записи из главной таблицы, если есть зависимые записи в другой таблице.

КАСКАДНОЕ ОБНОВЛЕНИЕ СВЯЗАННЫХ ПОЛЕЙ

Таблица "Персонал"

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Таблица "Зарплата"

Ключевое поле	Дата получения	Сумма	Кто получил
1	10.10.2005	10 000	2
2	13.10.2005	5 000	1
3	15.10.2005	7 000	2
4	11.10.2005	18 000	3

Вид таблиц после изменения записи с кодом 2

Таблица "Персонал"

Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
70	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

Изменение кода

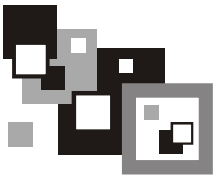
Таблица "Зарплата"

Ключевое поле	Дата получения	Сумма	Кто получил
1	10.10.2005	10 000	*70*
2	13.10.2005	5 000	1
3	15.10.2005	7 000	*70*
4	11.10.2005	18 000	3

Автоматическое изменение зависимых записей

Рис. 1.16. Каскадное обновление связанных таблиц

Глава 2



Язык SQL

2.1. Вступление

Различные СУБД хранят свои данные совершенно по-разному, для доступа к ним есть специальный язык — *SQL* (Structured Query Language — структурированный язык запросов). Он универсален и является как бы переводчиком между базой данных и клиентским приложением. Хотя в различных СУБД язык имеет свои тонкости, но основные команды SQL понимают все СУБД.

Язык SQL, в основном, ориентирован на конечный результат обработки данных. Отсюда такая универсальность и многофункциональность.

Все команды языка SQL делятся на:

- DDL (Data Definition Language) — язык определения данных, используется для создания объектов (создание базы данных, таблиц и служебных элементов). Как правило, состоит из следующих команд: `CREATE` (создание), `ALTER` (изменение), `DROP` (удаление).
- DML (Data Manipulation Language) — язык управления данными, используется для манипуляции данными, хранящимися в базе данных. Позволяет добавлять, изменять, удалять информацию, а также извлекать ее по условию. Как правило, состоит из следующих команд: `SELECT` (запросить данные), `INSERT` (добавить данные), `UPDATE` (изменить данные), `DELETE` (удалить данные).
- DCL (Data Control Language) — язык управления доступом к данным, используется для разрешения или запрещения доступа к объектам базы данных (команды `GRANT` и `REVOKE`).

SQL стандартизирован и поддерживается американским национальным институтом стандартов и международной организацией по стандартизации (ISO). Несмотря на это, некоторые производители баз данных вносят изменения и дополнения в этот язык.

В этой главе мы рассмотрим только основные операторы этого языка, то есть те, которые в первую очередь понадобятся в работе. К сожалению, рассмотреть все тонкости просто невозможно, на эту тему можно писать целую книгу. Если возникнет необходимость узнать больше, то советую почитать книгу Груббера "Понимание SQL", ее очень легко найти в Интернете.

Самая часто используемая команда SQL — `SELECT` — предназначена для выборки данных. Команда состоит из следующих блоков:

- команда `SELECT` указывает серверу, что клиент хочет получить данные;
- слово `FROM` указывает источник получения данных, им может быть таблица, представление и т. д.;
- слово `WHERE` позволяет указать условия выбора данных;
- слово `ORDER BY` указывает атрибуты, по которым будет произведена сортировка;
- дополнительные конструкции, предназначенные для построения сложных запросов, могут содержать также еще одну конструкцию `SELECT`.

Далее представлена общая схема построения запроса `SELECT`:

```
SELECT имя атрибута1, имя атрибута2, …, имя атрибутаN
       FROM таблица1, таблица 2, …, таблицаN
       WHERE атрибут1=условие1, атрибут2=условие2, …, атрибутN=условиеN
       ORDER BY атрибут1, атрибут2, …, атрибутN
```

2.2. Принцип построения запросов

Основой построения запроса является задание ключевого слова (команды), которое указывает на то, что мы хотим сделать. Каждая команда состоит из блоков, в которых необходимо указать дополнительные параметры.

Если мы хотим извлечь данные, то для этого используется команда `SELECT`, дальше нам надо указать, какие атрибуты нам нужны и из каких таблиц их брать (источник данных). Если мы хотим извлечь значения всех атрибутов, то, чтобы не перечислять все атрибуты, используется символ звездочка (`*`).

Если нам нужно произвести манипуляции с объектами базы данных, то, соответственно, используют другие команды, например, запрос на создание таблицы будет задаваться командой `CREATE TABLE`.

2.3. Команды управления данными (DML)

Для более глубокого понимания материала будем рассматривать его на примере. Предположим, есть база данных Фирма. Она состоит из двух таблиц:

"Персонал" (Personal) и "Зарплата" (Pay). Связь между таблицами — один-ко-многим. Главной таблицей является "Персонал" (табл. 2.1 и 2.2).

Таблица 2.1. *Таблица "Персонал" (Personal)*

Ключевое поле (PersonalID)	Имя (Name)	Отчество (SecondName)	Фамилия (Surname)	Профессия (Profession)
1	Вася	Васильевич	Васечкин	Водитель
2	Вася	Васильевич	Васечкин	Программист
3	Миша	Андреевич	Кузькин	Секретарь
4	Андрей	Сергеевич	Иванов	Директор
5	Сергей	Михайлович	Чуркин	уборщик

Таблица 2.2. *Таблица "Зарплата" (Pay)*

Ключевое поле (PayID)	Дата получения (Data)	Сумма (Summa)	Кто получил (PersonalKod)	Тип зарплаты (TypeOfPay)
1	10.10.2005	10 000	4	Оклад
2	13.10.2005	5000	1	Оклад
3	15.10.2005	7000	4	Премия
4	11.10.2005	1000	3	Аванс
5	11.10.2005	3000	3	Оклад
6	18.11.2005	4000	2	Оклад
7	20.12.2005	3000	5	Оклад

Теперь будем ставить различные задачи к базе данных, так как на примерах обучаться всегда легче.

Задача 1. Выбрать фамилии всех сотрудников, работающих в фирме.

```
SELECT Surname FROM Personal
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Surname` — атрибут таблицы Personal.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Personal` — имя таблицы-источника.

Результат выполнения этого запроса представлен в табл. 2.3.

Таблица 2.3

Surname
Васечкин
Васечкин
Кузькин
Иванов
Чуркин

Задача 2. Выбрать фамилии и должности всех сотрудников, работающих в фирме.

```
SELECT Surname, Profession FROM Personal
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Surname, Profession` — атрибуты таблицы Personal.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Personal` — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.4.

Таблица 2.4

Surname	Profession
Васечкин	Водитель
Васечкин	Программист
Кузькин	Секретарь
Иванов	Директор
Чуркин	Уборщик

Задача 3. Выбрать все данные таблицы Personal (первый способ).

```
SELECT PersonalID, Name, SecondName, Surname, Profession FROM Personal
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `PersonalID, Name, SecondName, Surname, Profession` — атрибуты таблицы Personal.

❒ `FROM` — ключевое слово, указывающее источник получения данных.

- `Personal` — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.5.

Таблица 2.5

PersonalID	Name	SecondName	Surname	Profession
1	Вася	Васильевич	Васечкин	Водитель
2	Вася	Васильевич	Васечкин	Программист
3	Миша	Андреевич	Кузькин	Секретарь
4	Андрей	Сергеевич	Иванов	Директор
5	Сергей	Михайлович	Чуркин	Уборщик

Задача 4. Выбрать все данные таблицы Personal (второй способ). В нем мы используем более короткую запись атрибутов, указывая вместо их названия символ `*`, который означает все атрибуты таблицы.

```
SELECT * FROM Personal
```

❒ `SELECT` — команда, указывающая серверу, что мы хотим получить данные.

- `*` — символ, который указывает, что будут выбираться все атрибуты таблицы.

❒ `FROM` — ключевое слово, указывающее источник получения данных.

- `Personal` — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.6.

Таблица 2.6

PersonalID	Name	SecondName	Surname	Profession
1	Вася	Васильевич	Васечкин	Водитель
2	Вася	Васильевич	Васечкин	Программист
3	Миша	Андреевич	Кузькин	Секретарь
4	Андрей	Сергеевич	Иванов	Директор
5	Сергей	Михайлович	Чуркин	Уборщик

Задача 5. Выбрать фамилии и должности всех сотрудников плюс дополнительный атрибут, который будет содержать пустые значения, под названием ForReport. Такой запрос может потребоваться при построении отчетов.

```
SELECT Surname, Profession, '' ForReport FROM Personal
```

Для InterBase и Firebird запрос будет следующим:

```
SELECT Surname, Profession, Null AS ForReport FROM Personal
```

- SELECT — команда, указывающая серверу, что мы хотим получить данные.
  - Surname, Profession — атрибуты таблицы Personal.
  - '' ForReport — дополнительный пустой атрибут (перед его именем необходимо поставить одинарные кавычки), физически он в таблицу не добавляется, а существует только во время отображения запроса.
- FROM — ключевое слово, указывающее источник получения данных.
  - Personal — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.7.

Таблица 2.7

Surname	Profession	ForReport
Васечкин	Водитель	
Васечкин	Программист	
Кузькин	Секретарь	
Иванов	Директор	
Чуркин	Уборщик	

Задача 6. Выбрать имена, отчества и фамилии всех сотрудников таким образом, чтобы в результате они отображались в одном атрибуте.

Вариант запроса представлен для MS SQL Server, потому что аналогичный запрос в InterBase или в Firebird не будет выполнен, так как данными СУБД выражения при указании атрибутов не поддерживаются.

```
SELECT Name+' '+SecondName+' '+Surname FROM Personal
```

- SELECT — команда, указывающая серверу, что мы хотим получить данные.
  - Name+' '+SecondName+' '+Surname — математическое выражение, аргументами которого являются атрибуты таблицы, знак суммирования и пробел, заключенный в одинарные кавычки (без пробела записи будут состоять из одного длинного слова).
- FROM — ключевое слово, указывающее источник получения данных.
  - Personal — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.8.

Таблица 2.8

Expr1000
Вася Васильевич Васечкин
Вася Васильевич Васечкин
Миша Андреевич Кузькин
Андрей Сергеевич Иванов
Сергей Михайлович Чуркин

Задача 7. Изменить запрос задачи 6 таким образом, чтобы имя атрибута было не Expr1000, а FIO.

Вариант запроса представлен для MS SQL Server, потому что аналогичный запрос в InterBase или в Firebird не будет выполнен, так как данными СУБД выражения при указании атрибутов не поддерживаются.

```
SELECT Name+' '+SecondName+' '+Surname AS FIO FROM Personal
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Name+' '+SecondName+' '+Surname` — математическое выражение, аргументами которого являются атрибуты таблицы, знак суммирования и пробел, заключенный в одинарные кавычки (без пробела записи будут состоять из одного длинного слова).
  - `AS` — указывает, что имя атрибута будет изменено.
  - `FIO` — имя атрибута в результате запроса, данное нами.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Personal` — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.9.

Таблица 2.9

FIO
Вася Васильевич Васечкин
Вася Васильевич Васечкин
Миша Андреевич Кузькин
Андрей Сергеевич Иванов
Сергей Михайлович Чуркин

Задача 8. Выбрать имя и профессию всех сотрудников таким образом, чтобы имена атрибутов были не `Name` и `Profession`, а `NameOfPersonal` и `ProfessionOfWorker`.

```
SELECT Name AS NameOfPersonal, Profession AS ProfessionOfWorker FROM
Personal
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Name AS NameOfPersonal` — сначала указывается атрибут таблицы, потом ключевое слово `AS`, затем новое имя атрибута, которое будет отображено в результате выполнения запроса.
  - `Profession AS ProfessionOfWorker` — сначала указывается атрибут таблицы, потом ключевое слово `AS`, затем новое имя атрибута, которое будет отображено в результате выполнения запроса.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Personal` — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.10.

Таблица 2.10

NameOfPersonal	ProfessionOfWorker
Вася	Водитель
Вася	Программист
Миша	Секретарь
Андрей	Директор
Сергей	Уборщик

Задача 9. Выбрать имена и фамилии всех сотрудников, работающих в фирме. Произвести сортировку по имени.

```
SELECT Name, Surname FROM Personal ORDER BY Name
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Name, Surname` — атрибуты таблицы Personal.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Personal` — имя таблицы-источника.
- `ORDER BY` — ключевое слово, указывающее, что будет осуществляться сортировка.
  - `Surname` — атрибут, по которому будет произведена сортировка.

Результат выполнения запроса представлен в табл. 2.11.

Таблица 2.11

Name	Surname
Андрей	Иванов
Вася	Васечкин
Вася	Васечкин
Миша	Кузькин
Сергей	Чуркин

Задача 10. Выбрать имена и фамилии всех сотрудников, работающих в фирме. Произвести сортировку по имени, сортировка должна быть в обратном порядке, то есть первые буквы алфавита в конце, а последние — в начале.

```
SELECT Name, Surname FROM Personal ORDER BY Name DESC
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Name, Surname` — атрибуты таблицы Personal.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Personal` — имя таблицы-источника.
- `ORDER BY` — ключевое слово, указывающее, что будет осуществляться сортировка.
  - `Surname` — атрибут, по которому будет произведена сортировка.
  - `DESC` — ключевое слово, указывающее, что сортировка для атрибута `Surname` будет произведена в обратном порядке.

Результат выполнения запроса представлен в табл. 2.12.

Таблица 2.12

Name	Surname
Сергей	Чуркин
Миша	Кузькин
Вася	Васечкин
Вася	Васечкин
Андрей	Иванов

Задача 11. Выбрать фамилии всех сотрудников таким образом, чтобы одинаковые фамилии не отображались.

```
SELECT DISTINCT Surname FROM Personal
```

- SELECT — команда, указывающая серверу, что мы хотим получить данные.
  - DISTINCT — ключевое слово, указывающее, что нужно возвращать только уникальные записи (без ключевого слова DISTINCT в запросе было бы две записи Васечкин).
  - Surname — атрибут таблицы Personal.
- FROM — ключевое слово, указывающее источник получения данных.
  - Personal — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.13.

Таблица 2.13

Surname
Васечкин
Иванов
Кузькин
Чуркин

Задача 12. Написать запрос, который посчитает количество записей в таблице "Персонал".

```
SELECT COUNT(Surname) FROM Personal
```

- SELECT — команда, указывающая серверу, что мы хотим получить данные.
  - COUNT — функция, которая считает количество записей в возвращенном результате запроса. В качестве параметра указывается имя атрибута в скобках.
- FROM — ключевое слово, указывающее источник получения данных.
  - Personal — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.14.

Таблица 2.14

Expr1000
5

Задача 13. Написать запрос, который посчитает общую сумму зарплаты, выданной сотрудникам (будем выбирать данные из таблицы "Зарплата").

```
SELECT SUM(Summa) FROM Pay
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `SUM` — функция, которая считает общую сумму значений, содержащихся в записях, возвращенных в результате запроса. В качестве параметра указывается имя атрибута в скобках, атрибут должен быть числового типа.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Pay` — имя таблицы-источника.

Результат выполнения запроса представлен в табл. 2.15.

Таблица 2.15

Expr1000
33 000,00 р.

Задача 14. Написать запрос, который посчитает сумму выданной зарплаты по каждому типу зарплаты.

```
SELECT SUM(Summa), TypeOfPay FROM Pay GROUP BY TypeOfPay
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `SUM(Summa)` — функция, которая считает общую сумму значений, содержащихся в записях, возвращенных в результате запроса. В качестве параметра указывается имя атрибута в скобках, атрибут должен быть числового типа.
  - `TypeOfPay` — атрибут таблицы Pay.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Pay` — имя таблицы-источника.
- `GROUP BY` — ключевое слово, указывающее, что будет производиться группировка (созданы группы) по значениям атрибута, указанного после него. В данном примере образуются следующие группы: Аванс, Оклад, Премия.

Результат выполнения запроса представлен в табл. 2.16.

Таблица 2.16

Expr1000	TypeOfPay
1000,00 р.	Аванс
25 000,00 р.	Оклад
7000,00 р.	Премия

Задача 15. Написать запрос, который посчитает сумму выданной зарплаты по датам выдачи.

```
SELECT SUM(Summa),Data FROM Pay GROUP BY Data
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `SUM(Summa)` — функция, которая считает общую сумму значений, содержащихся в записях, возвращенных в результате запроса. В качестве параметра указывается имя атрибута в скобках, атрибут должен быть числом.
  - `Data` — атрибут таблицы Pay.
- `FROM` — ключевое слово, указывающее источник получения данных.
  - `Pay` — имя таблицы-источника.
- `GROUP BY` — ключевое слово, указывает, что будет производиться группировка (созданы группы) по значениям атрибута, указанного после него.

Результат выполнения запроса представлен в табл. 2.17.

Таблица 2.17

Expr1000	Data
10 000,00 р.	10.10.2005
4000,00 р.	11.10.2005
5000,00 р.	13.10.2005
7000,00 р.	15.10.2005
4000,00 р.	18.11.2005
3000,00 р.	20.12.2005

Задача 16. Написать запрос, который выведет все данные из таблиц "Персонал" и "Зарплата". Результатом запроса будет каша из данных, который

очень сложно использовать практически, но в целях обучения данный запрос просто необходим.

```
SELECT Personal.Surname, Personal.Profession, Pay.Summa, Pay.TypeOfPay
FROM Personal,Pay
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Personal.Surname, Personal.Profession, Pay.Summa, Pay.TypeOfPay` — атрибуты таблиц, которые будут участвовать в запросе. Так как мы выбираем данные из двух таблиц, то нам надо указать, какой атрибут какой таблице принадлежит. Для этого используется конструкция: имя таблицы, точка, имя атрибута. Пример: `Personal.Surname` — атрибут `Surname` таблицы Personal.
- `FROM` — ключевое слово, указывающее источник получения данных (в данном случае их два).
  - `Personal, Pay` — имена таблиц-источников данных.

Результат выполнения запроса представлен в табл. 2.18.

Таблица 2.18

Surname	Profession	Summa	TypeOfPay
Васечкин	Водитель	10 000,00 р.	Оклад
Васечкин	Программист	10 000,00 р.	Оклад
Кузькин	Секретарь	10 000,00 р.	Оклад
Иванов	Директор	10 000,00 р.	Оклад
Чуркин	Уборщик	10 000,00 р.	Оклад
Васечкин	Водитель	5000,00 р.	Оклад
Васечкин	Программист	5000,00 р.	Оклад
Кузькин	Секретарь	5000,00 р.	Оклад
Иванов	Директор	5000,00 р.	Оклад
Чуркин	Уборщик	5000,00 р.	Оклад
Васечкин	Водитель	7000,00 р.	Премия
Васечкин	Программист	7000,00 р.	Премия
Кузькин	Секретарь	7000,00 р.	Премия
Иванов	Директор	7000,00 р.	Премия
Чуркин	Уборщик	7000,00 р.	Премия

Таблица 2.18 *(окончание)*

Surname	Profession	Summa	TypeOfPay
Васечкин	Водитель	1000,00 р.	Аванс
Васечкин	Программист	1000,00 р.	Аванс
Кузькин	Секретарь	1000,00 р.	Аванс
Иванов	Директор	1000,00 р.	Аванс
Чуркин	Уборщик	1000,00 р.	Аванс
Васечкин	Водитель	3000,00 р.	Оклад
Васечкин	Программист	3000,00 р.	Оклад
Кузькин	Секретарь	3000,00 р.	Оклад
Иванов	Директор	3000,00 р.	Оклад
Чуркин	Уборщик	3000,00 р.	Оклад
Васечкин	Водитель	4000,00 р.	Оклад
Васечкин	Программист	4000,00 р.	Оклад
Кузькин	Секретарь	4000,00 р.	Оклад
Иванов	Директор	4000,00 р.	Оклад
Чуркин	Уборщик	4000,00 р.	Оклад
Васечкин	Водитель	3000,00 р.	Оклад
Васечкин	Программист	3000,00 р.	Оклад
Кузькин	Секретарь	3000,00 р.	Оклад
Иванов	Директор	3000,00 р.	Оклад
Чуркин	Уборщик	3000,00 р.	Оклад

Замечание

Так как предыдущий вариант запроса вернул данные, которые неудобно использовать на практике, то, чтобы исправить эту ситуацию, надо явно указывать, как осуществляется связь между таблицами.

Задача 17. Написать запрос, который выведет фамилии сотрудников и полученные ими зарплаты. В запросе нужно учесть, что таблицы связаны отношением один-ко-многим и таблица "Персонал" — главная.

```
SELECT Personal.Surname, Pay.Summa FROM Personal, Pay
WHERE Pay.PersonalKod=Personal.PersonalID
```

- `SELECT` — команда, указывающая серверу, что мы хотим получить данные.
  - `Personal.Surname, Personal.Profession, Pay.Summa, Pay.TypeOfPay` — атрибуты таблиц, которые будут участвовать в запросе. Так как мы выбираем данные из двух таблиц, то нам надо указать, какой атрибут какой таблице принадлежит. Для этого используется конструкция: имя таблицы, точка, имя атрибута. Пример: `Personal.Surname` — атрибут `Surname` таблицы Personal.
- `FROM` — ключевое слово, указывающее источник получения данных (в данном случае их два).
  - `Personal, Pay` — имена таблиц — источников данных.
- `WHERE` — ключевое слово, после которого указываются условия выборки записей из источника (одной таблицы, нескольких таблиц и т. д.).
  - `Pay.PersonalKod=Personal.PersonalID` — условие выборки записей. В данном случае условие обозначает связь между таблицами Pay и Personal, по полям `PersonalKod` и `PersonalID`.

Результат выполнения запроса представлен в табл. 2.19.

Таблица 2.19

Surname	Summa
Иванов	10 000,00 р.
Васечкин	5000,00 р.
Иванов	7000,00 р.
Кузькин	1000,00 р.
Кузькин	3000,00 р.
Васечкин	4000,00 р.
Чуркин	3000,00 р.

Задача 18. Написать запрос, который выведет фамилии сотрудников и полученные ими зарплаты. В запросе учесть, что таблицы связаны отношением один-ко-многим и таблица Personal — главная. Результат запроса должен быть сгруппирован по фамилии; таким образом, будет получена общая сумма, полученная человеком за все время работы на фирме.

```
SELECT Personal.Surname, Sum(Pay.Summa) FROM Personal,Pay
       WHERE Pay.PersonalKod=Personal.PersonalID
       GROUP BY Personal.Surname
```

- SELECT — команда, указывающая серверу, что мы хотим получить данные.
  - Personal.Surname — атрибут Surname таблицы Personal.
  - Sum(Pay.Summa) — вычислить сумму для атрибута Summa таблицы Pay.
- FROM — ключевое слово, указывающее источник получения данных (в данном случае их два).
  - Personal, Pay — имена таблиц — источников данных.
- WHERE — ключевое слово, после которого указываются условия выборки записей из источника (одной таблицы, нескольких таблиц и т. д.).
  - Pay.PersonalKod=Personal.PersonalID — условие выборки записей. В данном случае условие обозначает связь между таблицами Pay и Personal по полям PersonalKod и PersonalID.
- GROUP BY — ключевое слово, указывающее, что будет производиться группировка по значениям атрибута, указанного после него (Personal.Surname).

Результат выполнения запроса представлен в табл. 2.20.

Таблица 2.20

Surname	Expr1001
Васечкин	9000,00 р.
Иванов	17 000,00 р.
Кузькин	4000,00 р.
Чуркин	3000,00 р.

Задача 19. Написать запрос, который выведет фамилии и зарплаты для сотрудников, получивших больше 5000.

```
SELECT Personal.Surname, Pay.Summa FROM Personal,Pay
      WHERE (Pay.PersonalKod=Personal.PersonalID) AND (Pay.Summa>5000)
```

- SELECT — команда, указывающая серверу, что мы хотим получить данные.
  - Personal.Surname, Personal.Profession, Pay.Summa, Pay.TypeOfPay — атрибуты таблиц, которые будут участвовать в запросе. Так как мы выбираем данные из двух таблиц, то нам надо указать, какой атрибут какой таблице принадлежит. Для этого используется конструкция: имя таблицы, точка, имя атрибута. Пример: Personal.Surname — атрибут Surname таблицы Personal.

- ❒ FROM — ключевое слово, указывающее источник получения данных (в данном случае их два).
  - Personal, Pay — имена таблиц — источников данных.
- ❒ WHERE — ключевое слово, после которого указываются условия выборки записей из источника (одной таблицы, нескольких таблиц и т. д.).
  - (Pay.PersonalKod=Personal.PersonalID) AND (Pay.Summa>5000) — условие. Когда необходимо в одном запросе использовать несколько условий, то условия берутся в скобки, а между ними ставится ключевое слово AND или OR. Если использовать AND, то это будет означать, что все условия обязательно должны выполниться (как в данном запросе). Если используется OR, то обязательно должно выполниться хотя бы одно из условий.

Результат выполнения запроса представлен в табл. 2.21.

Таблица 2.21

Surname	Summa
Иванов	10 000,00 р.
Иванов	7000,00 р.

Задача 20. Добавить нового сотрудника.

- ❒ Имя: Константин
- ❒ Отчество: Петрович
- ❒ Фамилия: Сидоров
- ❒ Профессия: Охранник

Выполнение задания будет состоять из трех частей. В первом запросе мы определим код, который надо присвоить сотруднику. Это нужно, чтобы код был уникальным. Во втором — добавим новую запись в таблицу. Также нам потребуется еще один запрос для проверки добавления новых данных.

Первый запрос:

```
SELECT MAX(PersonalID) FROM Personal
```

- ❒ SELECT — команда, указывающая серверу, что мы хотим получить данные.
  - MAX(PersonalID) — функция MAX определяет максимальное значение атрибута, в качестве параметра передается имя атрибута.
- ❒ FROM — ключевое слово, указывающее источник получения данных (в данном случае их два).
  - Personal — имя таблицы — источника данных.

Результат выполнения запроса представлен в табл. 2.22.

Таблица 2.22

Expr1000
5

Запрос второй:

```
INSERT INTO Personal
VALUES(6, 'Константин', 'Петрович', 'Сидоров', 'Охранник')
```

- `INSERT INTO` — команда для добавления новой записи.
  - `Personal` — имя таблицы, в которую будут добавляться данные.
- `VALUES` — ключевое слово, после которого в скобках через запятую указываются данные, которые необходимо добавить. Важно соблюдать порядок данных, то есть они должны располагаться в том же порядке, что и атрибуты в таблице.

Третий запрос (проверим, добавились ли данные):

```
SELECT * FROM Personal WHERE PersonalID=6
```

Результат выполнения запроса представлен в табл. 2.23.

Таблица 2.23

PersonalID	Name	SecondName	Surname	Profession
6	Константин	Петрович	Сидоров	Охранник

Задача 21. Добавить нового сотрудника Леонида Андреевича Мухина, но профессию, по которой он будет работать, не указывать, так как этот вопрос пока решается.

```
INSERT INTO Personal(Name,SecondName,Surname,PersonalID)
VALUES('Леонид','Андреевич','Мухин',7)
```

В этом запросе мы явно указываем порядок добавления данных в атрибуты: `Name`, `SecondName`, `Surname`, `PersonalID`. Также перечисляем те атрибуты, в которые данные будут добавляться. В данном запросе в атрибут `Profession` ничего добавляться не будет.

Проверим, добавились ли данные:

```
SELECT * FROM Personal WHERE PersonalID=7
```

Результат выполнения запроса представлен в табл. 2.24.

Таблица 2.24

PersonalID	Name	SecondName	Surname	Profession
7	Леонид	Андреевич	Мухин	

Задача 22. Повысить в должности до старшего программиста сотрудника Василия Васильевича Васечкина с порядковым номером 2.

```
UPDATE Personal SET Profession='Старший программист'
WHERE PersonalID=2
```

- `UPDATE` — команда для изменения записей (в данном случае будем менять одну запись).
  - `Personal` — имя таблицы, записи которой будут изменяться.
- `SET` — ключевое слово, после которого указывается атрибут и после знака равно — его новое значение. Можно указывать несколько атрибутов через запятую.
  - `Profession='Старший программист'` — атрибут, который подвергнется изменению, и его новое значение.
- `WHERE` — ключевое слово, после которого указываются условия изменения записей.
  - `PersonalID=2` — само условие (в таблице Personal данному условию удовлетворяет только одна запись).

Проверим, изменились ли данные (табл. 2.25):

```
SELECT * FROM Personal WHERE PersonalID=2
```

Таблица 2.25

PersonalID	Name	SecondName	Surname	Profession
2	Вася	Васильевич	Васечкин	Программист

Задача 23. Повысить у всех сотрудников выплаченную зарплату на 100.

```
UPDATE Pay SET Summa=Summa+100
```

В этом запросе используется конструкция `Summa=Summa+100`, то есть берется значение атрибута `Summa` и прибавляется к нему 100.

Проверим, изменились ли данные (табл. 2.26):

```
SELECT * FROM Pay
```

Таблица 2.26

PayID	Data	Summa	PersonalKod	TypeOfPay
1	10.10.2005	10 100,00 р.	4	Оклад
2	13.10.2005	5100,00 р.	1	Оклад
3	15.10.2005	7100,00 р.	4	Премия
4	11.10.2005	1100,00 р.	3	Аванс
5	11.10.2005	3100,00 р.	3	Оклад
6	18.11.2005	4100,00 р.	2	Оклад
7	20.12.2005	3100,00 р.	5	Оклад

Задача 24. Удалить сотрудника Мухина.

```
DELETE FROM Personal WHERE Surname='Мухин'
```

- DELETE — команда для удаления записей (в данном случае будем удалять одну запись).
  - Personal — имя таблицы, записи которой будут изменяться.
- FROM — ключевое слово, указывающее источник данных.
  - Personal — название таблицы.
- WHERE — ключевое слово, после которого указываются условия удаления записей.
  - Surname='Мухин' — само условие (в таблице Personal данному условию удовлетворяет только одна запись).

2.4. Команды определения данных (DDL)

Предположим, что в базу данных необходимо добавить дополнительную таблицу "Информация" (Info), которая будет содержать паспортные сведения персонала.

Рассмотрим SQL-запрос для создания таблицы:

```
CREATE TABLE Info
(
 InfoID integer  NOT NULL   PRIMARY KEY,
 PersonalKod integer,
 PasportSeriya char(10)
)
```

- `CREATE TABLE` — команда, указывающая серверу, что будет создаваться таблица.
  - `Info` — имя создаваемой таблицы.
- В скобках указываются атрибуты будущей таблицы и их типы данных. Ключевое слово `NOT NULL` указывает, что в атрибуте не могут содержаться пустые значения. Ключевое слово `PRIMARY KEY` указывает, что атрибут будет ключевым.
  - `Integer` — указывает, что тип данных атрибута `PersonalKod` будет числом.
  - `Char(10)` — указывает, что тип данных атрибута `PasportSeriya` будет символьным с максимальным количеством знаков 10.

Рассмотрим основные типы данных:

- `Char(n)` — символьный тип. Максимальное количество знаков указывается в скобках. Пример: char(10).
- `Varchar(n)` — символьный тип с максимальным количеством знаков, указанных в скобках. В отличие от типа `Char` хранит фактическое количество символов. Если атрибут типа `Varchar` будет содержать два символа, то длина строки будет равняться двум, несмотря на то, что максимальное количество введенных символов может быть 10.
- `Integer` — целое число.
- `Decimal(N,M)` — число с дробной частью. `N` — количество целых знаков. `M` — количество знаков после дробной части.
- `Real` — вещественное число.
- `Date` — дата, хранится в специальном формате (как правило, mm/dd/yy, формат вывода на экран зависит от настроек СУБД).
- `Time` — время, хранится в специальном формате (как правило, отображается в формате hh:mm:ss).
- `DateTime` — дата и время.

Задача 25. Создать индекс для атрибута `PersonalKod` таблицы Info. Через данный атрибут будет осуществляться связь с таблицей Personal, поэтому желательно наличие индекса.

```
CREATE INDEX IndexForPersonalKod ON Info(PersonalKod)
```

- `CREATE INDEX` — команда создания индекса.
  - `IndexForPersonal` — имя индекса.

- ON — ключевое слово, после него указываются имя таблицы и атрибут, для которого создается индекс.
  - Info(PersonalKod) — имя таблицы и в скобках имя атрибута, для которого будет создан индекс.

Задача 26. Добавить в таблицу Info дополнительный атрибут DiplomSeriya, в котором будет храниться серия диплома сотрудника.

```
ALTER TABLE Info ADD DiplomSeriya char(20)
```

- ALTER TABLE — команда изменения структуры таблицы.
  - Info — имя таблицы, которая подвергнется изменению.
- ADD — ключевое слово, указывает на то, что в таблицу будет добавлен дополнительный атрибут.
  - DiplomSeriya — имя добавляемого атрибута.
  - Char(20) — тип данных атрибута DiplomSeriya.

Задача 27. Удалить атрибут DiplomSeriya из таблицы Info.

```
ALTER TABLE Info DROP DiplomSeriya
```

Для MS SQL Server запрос будет следующим:

```
ALTER TABLE Info DROP COLUMN DiplomSeriya
```

- ALTER TABLE — команда изменения структуры таблицы.
  - Info — имя таблицы, которая подвергнется изменению.
- DROP — ключевое слово, указывает на то, что из таблицы будет удален атрибут.
  - DiplomSeriya — имя удаляемого атрибута.

2.5. Несколько полезных запросов

Задача 28. Добавить в таблицу Info следующие записи:

- 1 —1 —1A45B
- 2 — 2 — 37GH2
- 3 —3 — SSS67DD

Запросы:

```
INSERT INTO Info VALUES(1,1,'1A45B')
INSERT INTO Info VALUES(2,2,'37GH2')
INSERT INTO Info VALUES(3,3,'SSS67DD')
```

Проверим, добавились ли данные (результат представлен в табл. 2.27):

```
SELECT * FROM Info
```

Таблица 2.27

InfoID	PersonalKod	PasportSeriya
1	1	1A45B
2	2	37GH2
3	3	SSS67DD

Задача 29. Вывести фамилию, профессию, дату получения зарплаты, сумму зарплаты и паспортные данные людей, работающих в фирме.

```
SELECT Personal.Surname, Personal.Profession,
       Pay.Data, Pay.Summa, Info.PasportSeriya
FROM Personal, Pay, Info
WHERE (Personal.PersonalID = Pay.PersonalKod)
                        AND
            (Personal.PersonalID = Info.PersonalKod)
```

Все команды должны быть уже знакомы, особого внимания заслуживает только условие `(Personal.PersonalID = Pay.PersonalKod) AND (Personal.PersonalID = Info.PersonalKod)`. В нем мы указываем, что связь между таблицами осуществляется через атрибут `PersonalID` главной таблицы и атрибут `PersonalKod` таблиц Pay и Info.

Результат выполнения запроса представлен в табл. 2.28.

Таблица 2.28

Surname	Profession	Data	Summa	PasportSeriya
Васечкин	Водитель	13.10.2005	5100,00 р.	1A45B
Кузькин	Секретарь	11.10.2005	1100,00 р.	SSS67DD
Кузькин	Секретарь	11.10.2005	3100,00 р.	SSS67DD
Васечкин	Программист	18.11.2005	4100,00 р.	37GH2

Задача 30. Вывести фамилию, профессию, дату получения зарплаты, сумму зарплаты для тех людей, у которых не введены паспортные данные.

```
SELECT Personal.Surname, Personal.Profession, Pay.Data, Pay.Summa
FROM Personal, Pay
```

```
WHERE (Personal.PersonalID = Pay.PersonalKod)
                    AND
    Personal.PersonalID NOT IN (SELECT PersonalKod FROM Info)
```

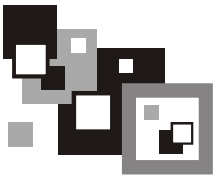
- Сначала мы указываем атрибуты для выборки, затем источники данных. Внимание следует обратить на то, что таблица Info здесь не указана, потому что смысла в этом нет, данных о людях, которых мы ищем, там не содержится.
- Условие поделено на две части, с первой проблем возникнуть не должно, так как это просто указания связи между таблицами Personal и Pay. Вторая часть представляет интерес, так как состоит из еще одного запроса.
  - `Personal.PersonalID NOT IN (SELECT PersonalKod FROM Info)` — условие указывает, что значения атрибута `PersonalID` не должны принадлежать значениям атрибута `PersonalKod` таблицы Info. Ключевое слово `NOT` является отрицанием (НЕ), а ключевое слово `IN` является оператором принадлежности.

Результат выполнения запроса представлен в табл. 2.29.

Таблица 2.29

Surname	Profession	Data	Summa
Иванов	Директор	10.10.2005	10 100,00 р.
Иванов	Директор	15.10.2005	7100,00 р.
Чуркин	Уборщик	20.12.2005	3100,00 р.

Глава 3



InterBase

3.1. Вступление

В этой главе мы разработаем клиент-серверное приложение Продажа товара. Оно будет позволять регистрировать товар на складе, оформлять продажу, а также будет предоставлять возможность простейшей аутентификации пользователей. Также рассмотрим вопросы установки и администрирования *InterBase*.

3.2. Повторение материала

Не секрет, что базы данных делятся на однопользовательские и многопользовательские. В первом варианте пользователь работает один со своей базой данных и может делать с ней все, что хочет. При многопользовательском варианте сразу возникает проблема: нужно обеспечить достоверность данных при одновременной работе нескольких пользователей с одной базой данных.

Чтобы обеспечить пользователей удобной работой в многопользовательском режиме, используют транзакции, принцип действия их вам уже знаком, но я все равно напомню.

Перед внесением изменений в базу данных:

1. Начинаем транзакцию.
2. Вносим изменения.
3. Пытаемся завершить транзакцию.
   - Если завершить удалось, то все изменения успешно сохраняются.
   - Если произошла ошибка, то отказываемся от всех изменений.
4. Если передумали вносить изменения, то пропускаем пункт 3 и сразу отменяем транзакцию.

Также в этой главе мы будем использовать команды языка SQL *(см. главу 2)*.

3.3. Постановка задачи

Представьте, что перед вами стоит задача написания программы для продажи товара со склада различным организациям. Вы подумали над этой проблемой и решили, что в вашей базе данных будут следующие таблицы: табл. 3.1, 3.2 и 3.3.

Таблица 3.1. *Таблица "Товар" (Tovar)*

Имя атрибута	Описание	Тип данных
TovarID	Поле-счетчик	Счетчик
Name	Название товара	Текст(100)
Price	Цена товара	Вещественный

Таблица 3.2. *Таблица "Фирмы" (Firm)*

Имя атрибута	Описание	Тип данных
FirmID	Поле-счетчик	Счетчик
Name	Название организации	Текст(100)

Таблица 3.3. *Таблица "Продажи" (Sale)*

Имя атрибута	Описание	Тип данных
SaleID	Поле-счетчик	Счетчик
FirmKod	Код фирмы, совершающей покупку	Число
TovarKod	Код товара, который купили	Число
Rem	Комментарии по покупке	Текст(100)

3.4. Установка InterBase

В дистрибутив Delphi 6 входит InterBase 6 — это последняя бесплатная версия данной СУБД.

С Delphi 7 поставляется InterBase 6.5, с Delphi 2005 — InterBase 7.5.

Рис. 3.1. Окно выбора установки компонентов Delphi 7

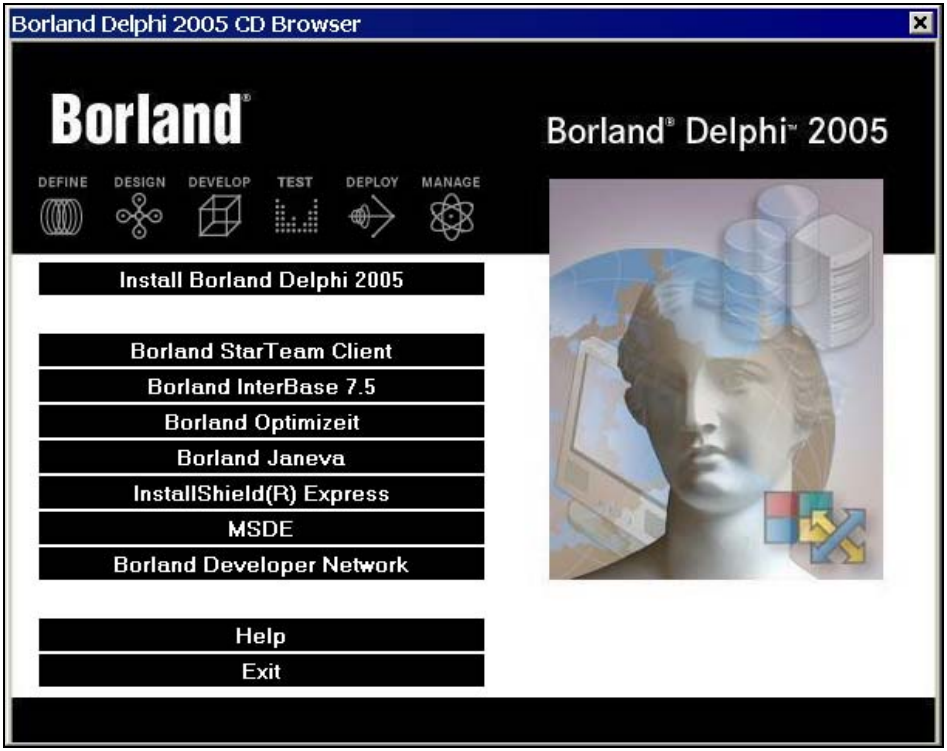


Рис. 3.2. Окно выбора установки компонентов Delphi 2005

Замечание

Мы рассмотрим установку InterBase 6.5 и InterBase 7. Установка любой другой версии InterBase практически ничем не отличается. Главное отличие, что версии выше 6 платные и ими можно пользоваться бесплатно только некоторое время, а потом потребуется регистрация.

Установку можно запустить двумя способами.

- Запустить файл установки Delphi install.exe (для версий 7 и 2005 он называется одинаково). Появится окно выбора компонентов для установки, в котором выбирается установка InterBase 6.5 Server для Delphi 7 и Borland InterBase 7.5 для Delphi 2005 (рис. 3.1 и 3.2).
- Запустить файл установки InterBase. Для Delphi 7 этот файл называется setup.exe находится в подкаталоге ib6.5 дистрибутива Delphi 7. Для Delphi 2005 файл называется ib_setup.exe и находится в подкаталоге ib7.5 дистрибутива Delphi 2005.

3.4.1. Установка InterBase 6.5

После запуска установки появится окно приветствия (рис. 3.3).

Рис. 3.3. Окно приветствия InterBase 6.5

Нажимаем на кнопку **Next**. Появляется окно **Software License Agreement**, в котором предлагается ознакомиться с условиями лицензионного соглашения программного обеспечения. Нажимаем **Yes**. После этого появится окно **Important Installation Information** с информацией о том, как именно происходит установка InterBase 6.5. Нажимаем **Next**.

Появляется окно **Select Destination Directory**, в котором предлагается выбрать путь установки, рекомендую оставить его предложенным по умолчанию. Нажимаем **Next**. Появится окно **Select Component**, в котором выбирается тип установки и устанавливаемые компоненты. Оставляем **Typical Installation** и включаем установку **Example Databases** (Примеры баз данных). Для этого щелкаем левой кнопкой мыши на названии устанавливаемого компонента и в появившемся выпадающем меню выбираем **Install the component** (рис. 3.4).

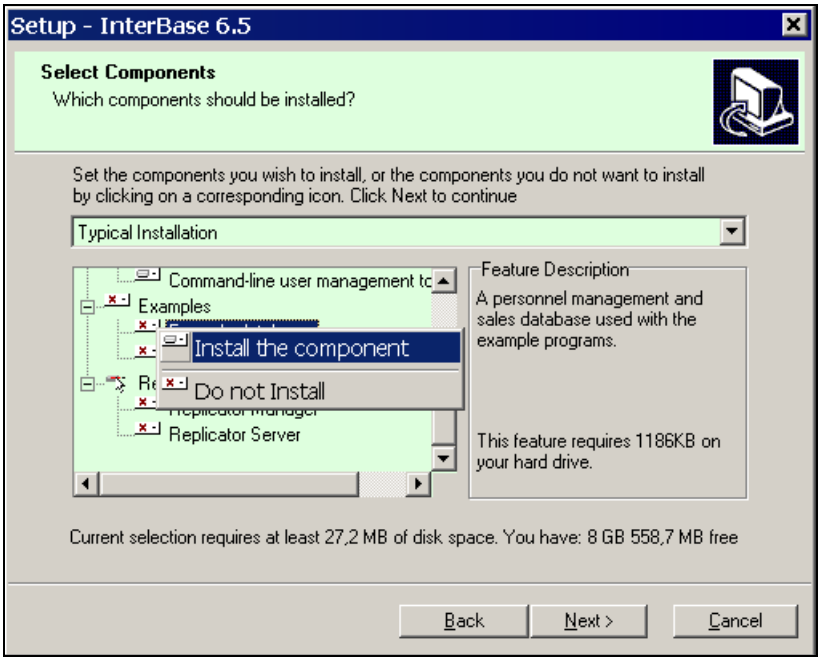


Рис. 3.4. Окно выбора компонентов для установки

Нажимаем **Next**. Появляется окно **Select Start Menu Folder**, в котором предлагается выбрать раздел в меню **Программы**, где будут созданы ярлыки приложения. Оставляем по умолчанию. Нажимаем **Next**. Появляется окно **Ready to Install**, в котором отображается информация по установке (обобщенные данные о нашем выборе в предыдущих окнах). Нажимаем **Install**. Начнется

установка, после ее завершения появится окно, в котором сообщается о том, что установка закончена (рис. 3.5).

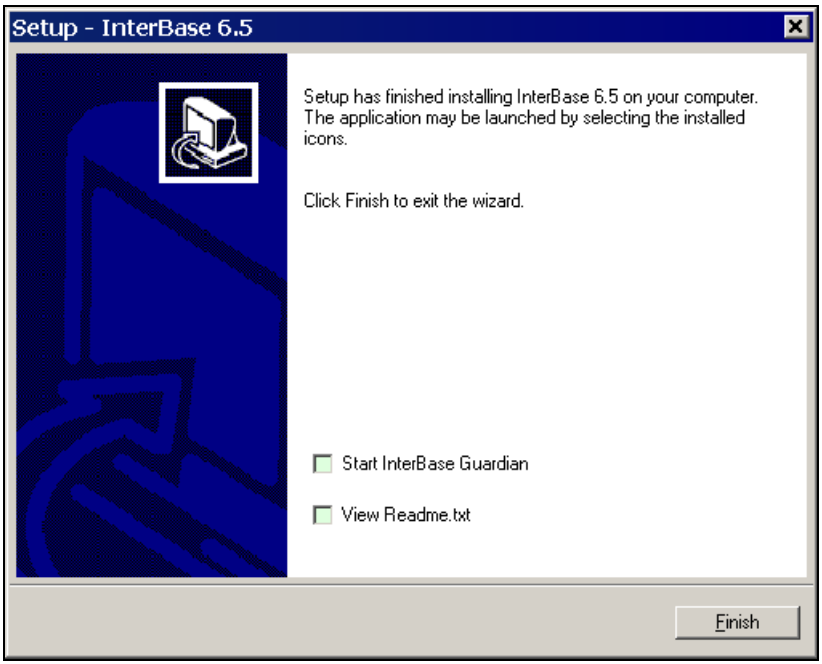


Рис. 3.5. Окно, появляющееся после завершения установки InterBase 6.5

Нажимаем **Finish**.

3.4.2. Установка InterBase 7.5

После запуска установки появится окно-заставка (рис. 3.6).

После того как мастер установки InterBase 7.5 загрузится в оперативную память компьютера, на экране отобразится окно приветствия (рис. 3.7).

Нажимаем **Next**. Появляется окно **License Agreement**, в котором предлагается ознакомиться с условиями лицензионного соглашения программного обеспечения. Устанавливаем переключатель **I accept the terms of the License Agreement**. Нажимаем **Next**. Появляется окно **MultiVersion InterBase**, в котором говорится о том, что данная версия InterBase предоставляет возможность запускать несколько экземпляров сервера на одном компьютере. Каждый экземпляр будет использовать в своей работе определенный номер порта (рис. 3.8).

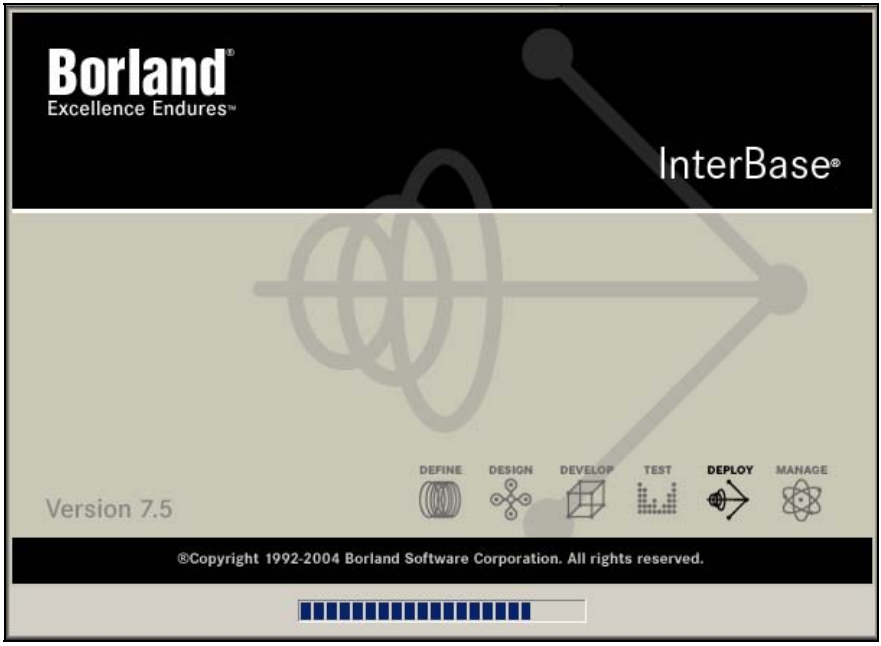


Рис. 3.6. Окно-заставка InterBase 7.5

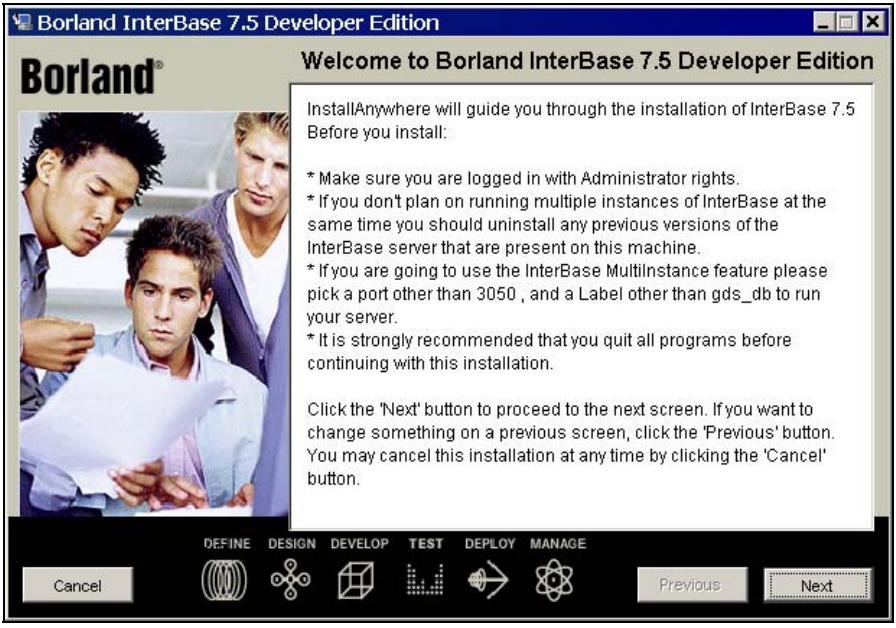


Рис. 3.7. Окно приветствия InterBase 7.5

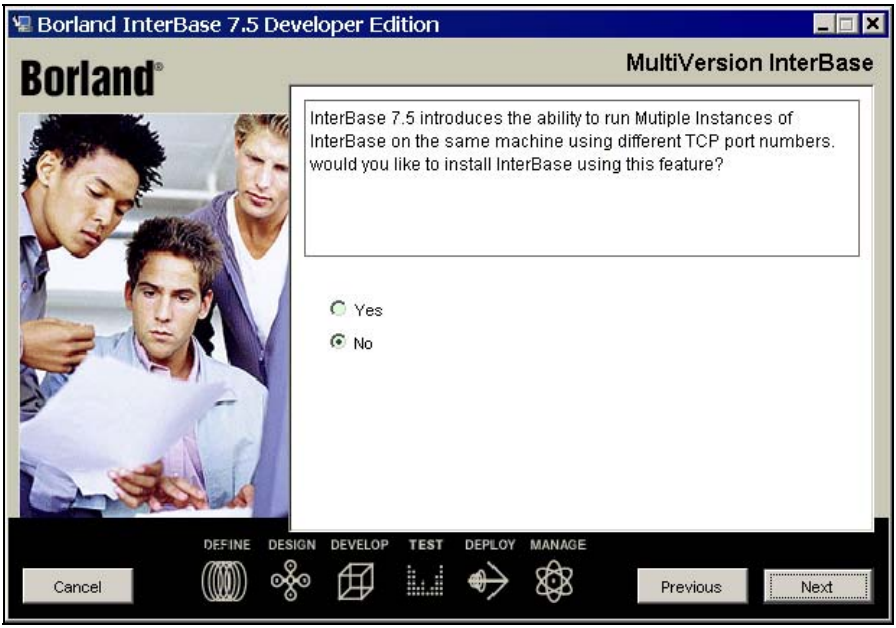


Рис. 3.8. Окно **MultiVersion InterBase**

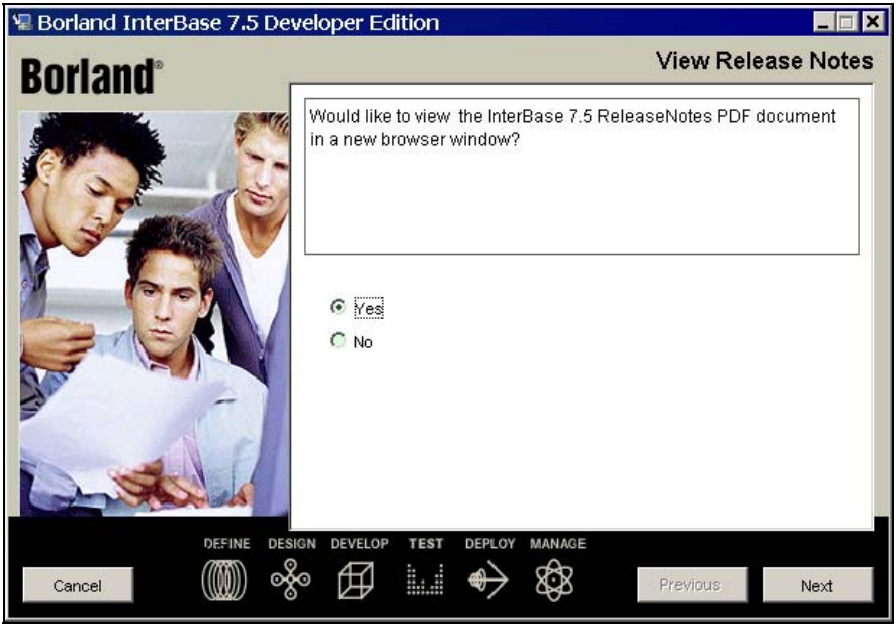


Рис. 3.9. Окно **View Release Notes**

Оставляем установленный по умолчанию переключатель **No** и нажимаем **Next**. Появляется окно **Choose Install Folder**, в котором предлагается выбрать путь установки, рекомендую оставить его по умолчанию. Нажимаем **Next**. Появляется окно **Pre-Installation Summary**, в котором отображается информация по установке (обобщенные данные о нашем выборе в предыдущих окнах). Нажимаем **Install**. Начнется установка, после ее завершения появится окно **View Release Notes**, в котором предлагается просмотреть в AcrobatReader особенности и нововведения InterBase 7.5 (рис. 3.9). Если вас интересует эта информация (она на английском языке), то оставьте установленным переключатель **Yes**, в противном случае установите переключатель **No**.

Нажимаем **Next**. Появляется последнее окно **Install Complete**. Нажимаем **Done**.

3.4.3. После установки

После установки рекомендуется перезагрузить компьютер.

Замечание

Работа с обеими версиями InterBase после их установки ничем не отличается.

Особый интерес представляет инструмент InterBase Manager (менеджер InterBase). Для его запуска необходимо выбрать **Пуск\Настройка\Панель управления\InterBase Manager**. На экране появится окно **InterBase Manager** (рис. 3.10).

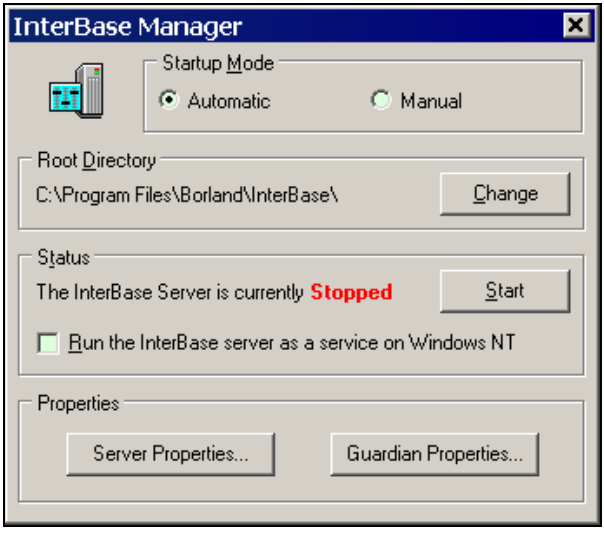


Рис. 3.10. Окно **InterBase Manager**

В этом окне можно запускать и останавливать работу сервера. Можно установить, чтобы сервер запускался автоматически при старте системы или чтобы запуск происходил в ручном режиме. Также можно установить, чтобы сервер запускался как приложение, а можно, чтобы он работал как сервис.

Остановка работы сервера осуществляется нажатием на кнопку **Stop**. Если сервер отключен, то его можно запустить, нажав на кнопку **Start** (она будет располагаться на месте кнопки **Stop**). Также можно поменять способ работы сервера. Если установлен переключатель **Run the InterBase server as a service on Windows NT**, то сервер будет работать как служба Windows, иначе — как отдельное приложение.

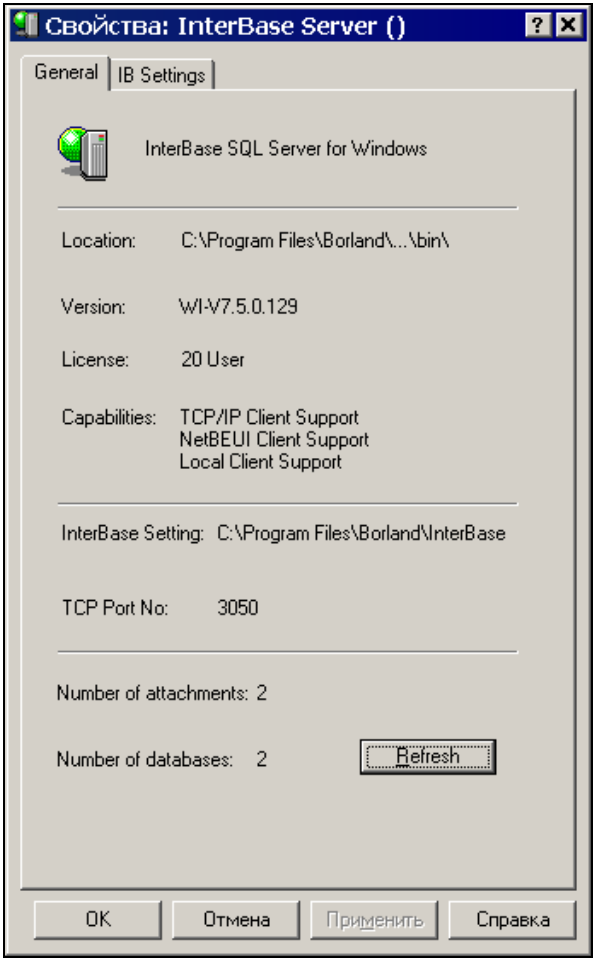


Рис. 3.11. Окно **Свойства: InterBase Server**

В группе элементов **Startup Mode** можно менять тип запуска сервера с помощью переключателей:

- **Automatically** — автоматический запуск при загрузке Windows.
- **Manually** — ручной запуск. Если установлен этот переключатель, то для того чтобы запустить сервер, необходимо будет войти в менеджер **InterBase** и нажать кнопку **Start**.

Когда InterBase запускается как отдельное приложение, то в трее (области уведомления панели задач) появляется новый ярлык. Если произвести на нем двойной щелчок левой кнопкой мыши, то появится окно **Свойства: InterBase Server** (рис. 3.11).

В этом окне на вкладке **General** можно просмотреть информацию о версии InterBase (**Version**); количество лицензий для одновременно подключенных пользователей (**License**); номер порта, на котором будет работать InterBase (**TCP Port No**). На вкладке **IB Settings** можно изменить параметры работы сервера InterBase.

Замечание

Окно **Свойства: InterBase Server** также можно вызвать нажатием кнопки **Server properties** в менеджере **InterBase**.

3.5. Создание базы данных в InterBase

Для того чтобы приступить к созданию базы данных, необходимо запустить IBConsole — это специальный инструмент для управления работой InterBase.

Замечание

Далее рассматривается работа с InterBase 7.5. Работа с младшими версиями ничем не отличается, хотя могут быть совсем незначительные отличия, на которые автор не обратил внимания.

Для Delphi 2005 выбираем **Пуск\Программы\InterBase 7.5 Developer Edition\IBConsole** (рис. 3.12). Для Delphi 7: **Пуск\Программы\InterBase\IBConsole**.

Окно после запуска IBConsole представлено на рис. 3.13. Оно состоит из главного меню, панели инструментов и двух информационных областей. Левая область предназначена для отображения различных объектов, ими могут быть сервер, база данных, таблица, хранимая процедура и т. д. Правая часть содержит информацию по выбранному объекту.

Так как мы только что установили InterBase, то доступных объектов у нас нет. В первую очередь необходимо зарегистрировать сервер баз данных и

подключиться к нему. Для этого необходимо выбрать пункт меню **Server\Register** либо щелкнуть правой кнопкой мыши на первой иконке слева в панели инструментов. На экране появится окно **Register Server and Connect** (Зарегистрировать сервер и подключиться) — рис. 3.14.

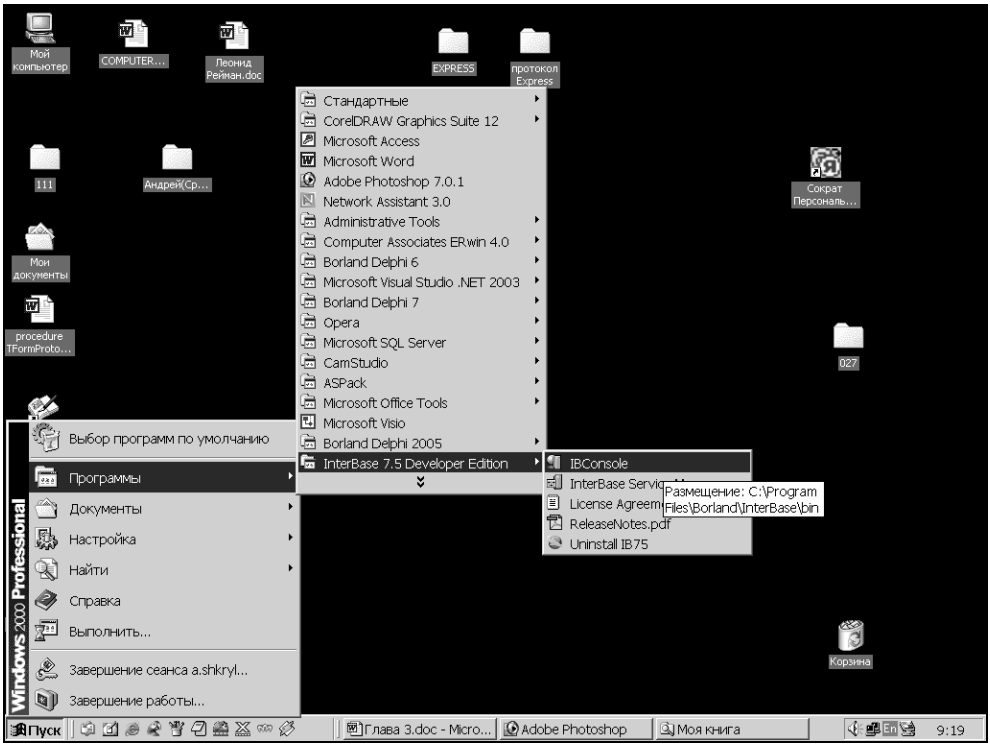


Рис. 3.12. Запуск IBConsole

Так как база будет создаваться на своем компьютере, устанавливаем переключатель **Local Server**. Если хочется поэкспериментировать и создать базу удаленно на сервере (если, конечно, он имеется), то устанавливаем переключатель **Remote Server**.

- ❒ В поле **Description** вводим описание сервера.
- ❒ В поле **Login Information** вводим логин и пароль администратора базы данных:
  - **UserName**: SYSDBA
  - **PASSWORD**: masterkey

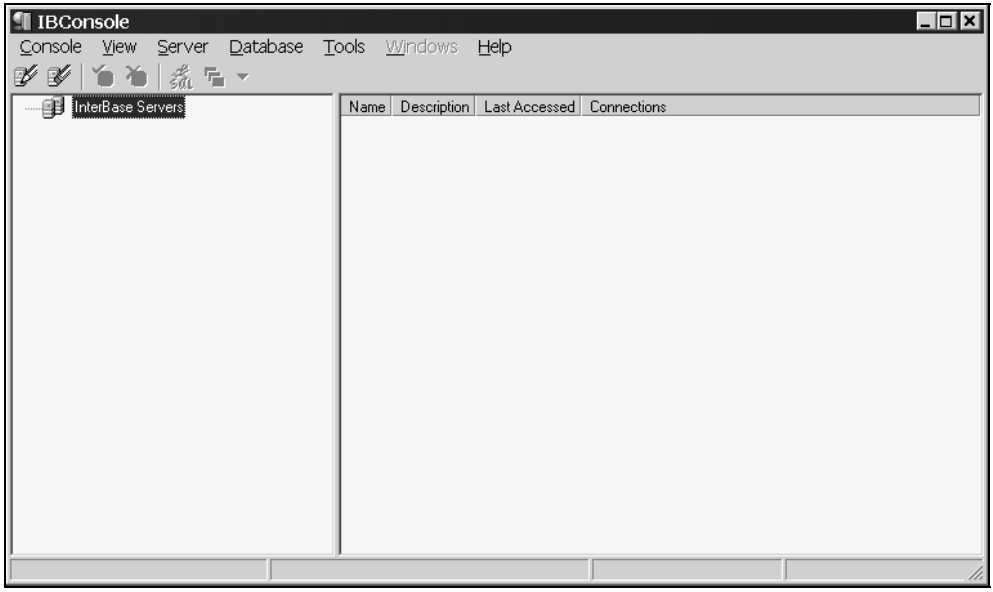


Рис. 3.13. IBConsole

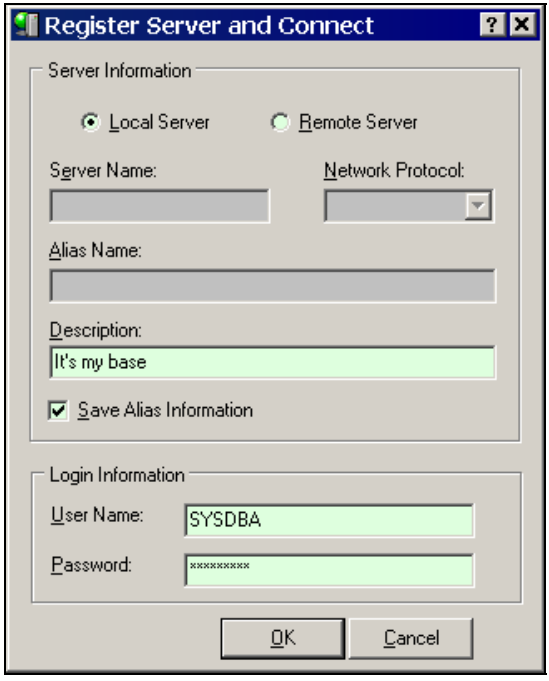


Рис. 3.14. Окно **Register Server and Connect**

Замечание

После установки InterBase логин администратора — SYSDBA, пароль — masterkey.

Нажимаем **OK** — сервер зарегистрирован. Если сервер в этот момент не запущен, то на экране появится окно для подтверждения **Confirm** (Подтвердить), в котором будет задан вопрос: "Сервер не запущен. Вы хотите запустить его сейчас?" Соглашаемся, нажимая на **Yes** (рис. 3.15).

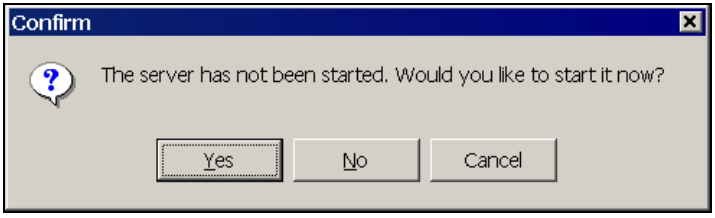


Рис. 3.15. Окно **Confirm**
для подтверждения запуска сервера

После того как сервер запущен, в верхней части его пиктограммы появится зеленая галочка. Если соединение с сервером не установлено, то отобразится красный крестик. Имя сервера будет **LocalServer**, оно присваивается автоматически (рис. 3.16).

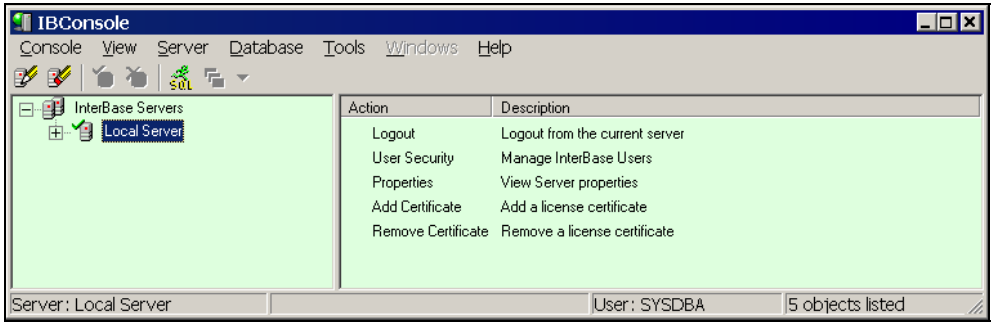


Рис. 3.16. Вид окна **IBConsole**
после регистрации и подключения к серверу

Теперь создадим рабочую папку для будущего проекта, у меня путь к ней следующий: C:\WorkIB.

Замечание

Правилом хорошего тона является создавать для каждого нового проекта отдельную папку с осмысленным названием. В дальнейшем, когда количество ваших проектов увеличится, это позволит вам быстро найти необходимый проект.

Создаем базу данных. Выбираем пункт меню **Database\Create Database**. Появится окно **Create Database**.

Указываем название файла базы данных и его месторасположение (у меня C:\WorkIB\WorkIB.gdb) в первой строке списка **Files**.

Замечание

Расширение для баз данных InterBase — GDB.

В выпадающем списке **Default Character Set** выбираем пункт **WIN1251**. Это необходимо для того, чтобы нормально отображались русские буквы.

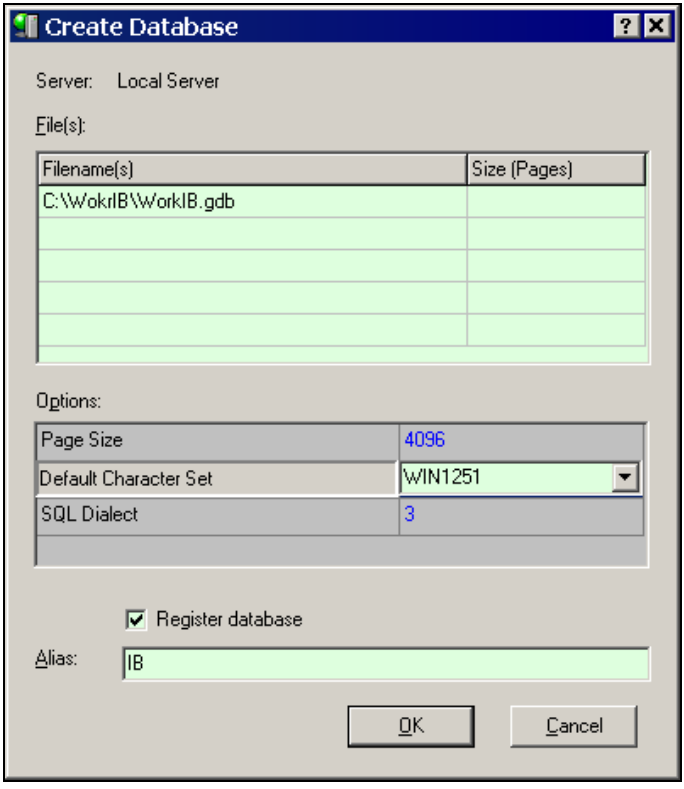


Рис. 3.17. Окно **Create Database** с заполненными параметрами для создания базы данных

В поле **Alias** вводим название будущей базы данных (у меня IB). Остальное оставляем по умолчанию (рис. 3.17).

Нажимаем **OK** — база создана, окно **IBConsole** приобретает следующий вид (рис. 3.18).

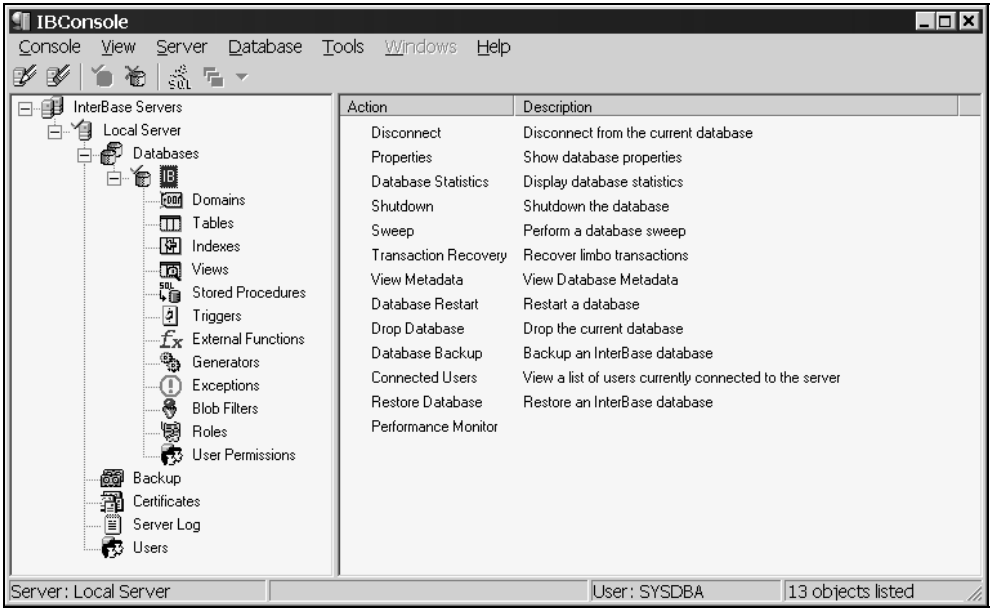


Рис. 3.18. Окно **IBConsole** после создания базы данных

Теперь переходим к созданию таблиц. Выбираем в дереве в левой половине окна пункт **Tables** (Таблицы) и нажимаем кнопку **SQL** в панели инструментов (пятая слева). Появляется окно **Interactive SQL** (Интерактивный SQL). У меня данное окно представлено уже с введенным запросом на создание таблицы "Товар" (рис. 3.19).

Рассмотрим особенности этого запроса:

```
CREATE TABLE "TOVAR"
(
  "TOVARID"    INTEGER NOT NULL,
  "NAME"       VARCHAR(100) CHARACTER SET WIN1251 NOT NULL,
  "PRICE"      INTEGER NOT NULL,

 PRIMARY KEY ("TOVARID")
 );
```

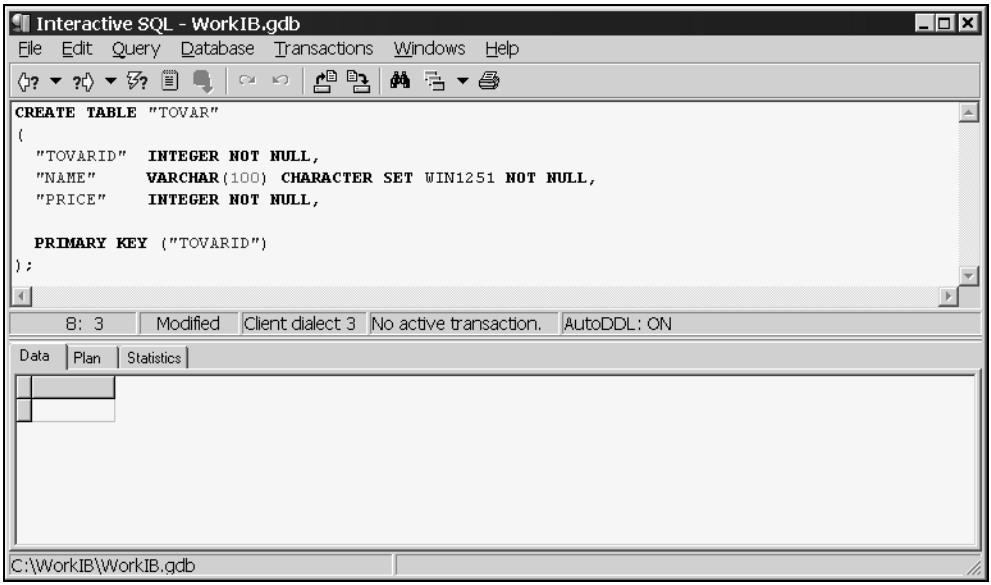


Рис. 3.19. Окно **Interactive SQL** после создания базы данных

`CREATE TABLE` — команда создания таблицы. В кавычках указывается ее название, дальше в скобках указываются параметры таблицы, потом скобка закрывается и обязательно ставится точка с запятой. В скобках у нас указано три поля.

`NOT NULL` означает, что поле не может содержать пустых значений. Если пользователь попытается сохранить значение `NULL` в атрибуте таблицы, для которой установлен этот параметр, то произойдет ошибка на уровне СУБД, а обрабатывать эту ошибку в своей программе или нет — это уже решение программиста.

Атрибут `Name` имеет текстовое значение в InterBase. Оно обозначается как `VARCHAR`, можно использовать и просто `CHAR` *(о различиях между ними можно посмотреть в главе 2)*. В скобках указывается максимальное количество возможных символов, также указывается кодировка для символов — это нужно, чтобы нормально отображались русские символы.

Замечание

В окне **Interactive SQL** можно выполнять абсолютно все типы запросов: создание таблиц, добавление в них данных, извлечение данных и т. д. Если вы забыли какую-то из команд SQL, то всегда можно воспользоваться справкой. Там можно найти много полезной информации. Справка вызывается выбором пункта меню **HELP\SQL Reference** в окне **Interactive SQL**.

Нажимаем на кнопку "Высокое напряжение" (третья слева), после чего таблица будет создана и окно с запросом очистится. Закрываем окно **Interactive SQL** и смотрим результат нашей работы. Вид окна **IBConsole** после создания таблицы представлен на рис. 3.20.

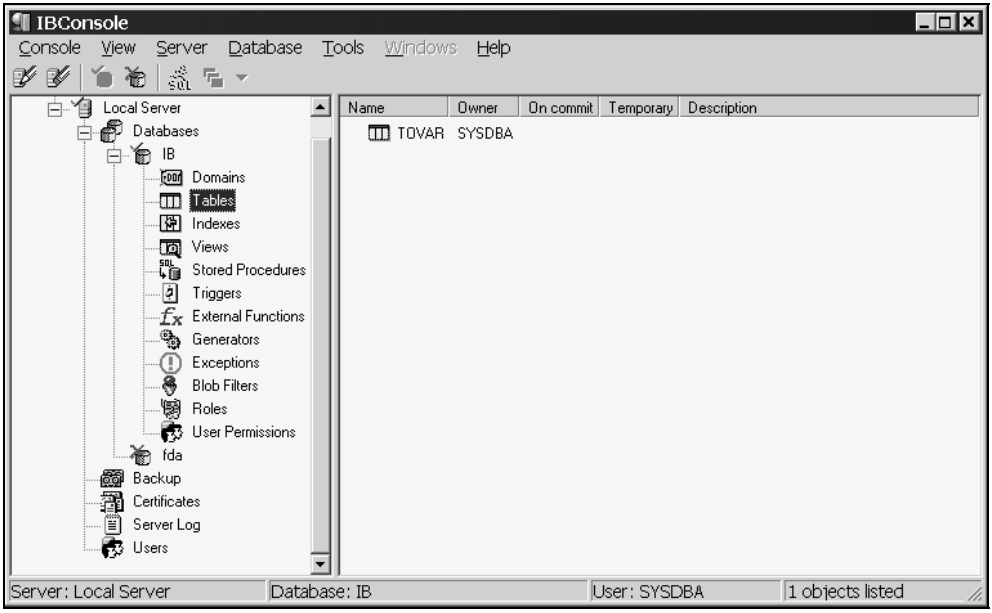


Рис. 3.20. Окно **IBConsole** после создания таблицы

В InterBase нет такого типа, как счетчик или **autoincrement** (данный тип есть в Microsoft Access), выход из этой ситуации обеспечивается созданием так называемого *генератора*.

Генератор — это хранящаяся в базе данных программа (или скрипт), выдающая при каждом обращении к ней уникальное число. Для каждого автоинкрементного атрибута в базе данных можно создать свой генератор, можно также использовать один генератор для нескольких атрибутов. Мы пойдем по первому пути, то есть для каждого атрибута-счетчика будем создавать отдельный генератор.

Вызываем **Interactive SQL**. Пишем запрос присвоения генератору начального значения:

```
CREATE GENERATOR GEN_TOVAR
```

Нажимаем выполнить. После этого запрос выполнится, и окно запросов очистится. Мы создали генератор `GEN_TOVAR`. Теперь нам надо установить

начальное значение генератора, для этого пишем запрос присвоения генератору начального значения:

```
SET GENERATOR GEN_TOVAR TO 0
```

Нажимаем выполнить. При этом запрос выполнится, и окно запросов очистится.

Замечание

В запросе присвоения начального значения само значение не может быть результатом запроса SELECT, поэтому, если таблица не пустая на момент создания генератора, то начальное значение определится после выполнения следующего запроса:

```
SELECT MAX(TOVARID)+1 FROM TOVAR
```

Теперь у нас есть генератор с заданным начальным значением. Осталось только привязать генератор к конкретному атрибуту таблицы. Пишем запрос на создание триггера:

```
SET TERM !! ;
CREATE TRIGGER "BEF_INS_TOVAR" FOR "TOVAR"
ACTIVE BEFORE INSERT
AS
BEGIN
 NEW.TOVARID=GEN_ID(GEN_TOVAR,1);
END!!
SET TERM ; !!
```

Выполняем запрос, после чего генератор будет введен в работу.

Замечание

Конструкция SET TERM задает новый разделитель операторов. Если ее не использовать, то получится, что у вас и после NEW.TOVARID=GEN_ID(GEN_TOVAR,1); стоит точка с запятой, и после END тоже стоит точка запятой, а InterBase это не нравится.

Рассмотрим подробней то, что мы написали:

- CREATE TRIGGER — данная команда указывает, что мы хотим создать триггер (или правило). Далее указываем название триггера и для какой таблицы он будет предназначен (FOR "TOVAR").
- Предложение ACTIVE BEFORE INSERT — указывает, когда триггер должен выполняться, в данном случае каждый раз перед созданием новой записи (ACTIVE BEFORE INSERT переводится с английского, как активный до вставки).

- Слово AS зарезервированное, открывает тело триггера. Тело триггера всегда (даже если триггер содержит единственный оператор, как в нашем случае) должно ограничиваться парой ключевых слов BEGIN — END. В шестой строке расположен оператор, в котором новому значению (слово NEW) атрибута TOVARID присваивается значение, полученное от встроенной функции GEN_ID. Двумя параметрами обращения к этой функции указывается имя генератора (GEN_TOVAR) и то значение, на которое должно увеличиться текущее значение генератора ("шаг" генератора), в нашем случае "шаг" равен единице.

Выбирая справа в дереве окна **IBConsole** пункт **Generators**, можно проверить наличие созданного генератора. А если щелкнуть два раза левой кнопкой мыши по имени генератора, то отобразится окно **Properties**, содержащее свойства выбранного генератора (рис. 3.21 и 3.22).

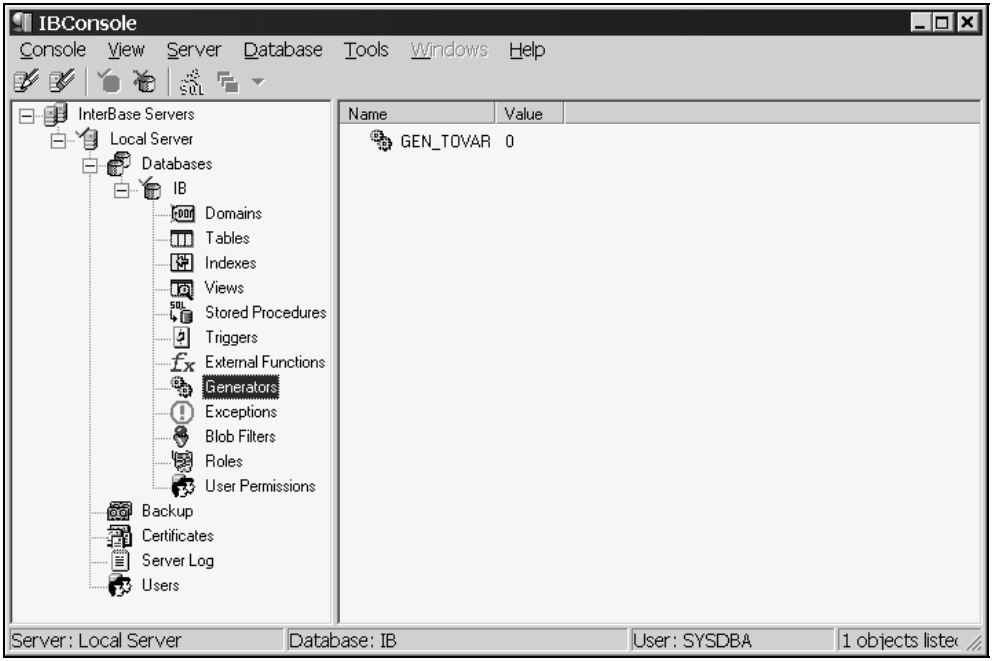


Рис. 3.21. Окно **IBConsole** с выбранным пунктом **Generators**

Созданный триггер можно посмотреть, выбрав в окне **IBConsole** пункт **Table** и щелкнув два раза левой кнопкой мыши на имени таблицы (в нашей базе есть пока только одна таблица TOVAR). Появится окно **Properties**, в данном случае оно будет отображать свойства таблицы. Необходимо выбрать вкладку **Dependencies** (рис. 3.23).

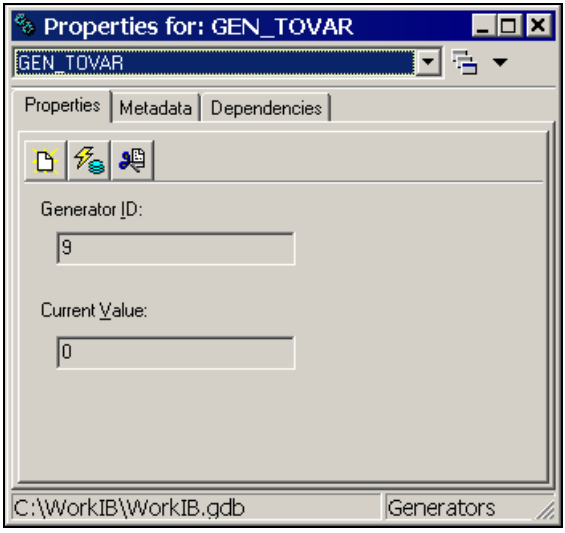


Рис. 3.22. Окно, содержащее свойства генератора

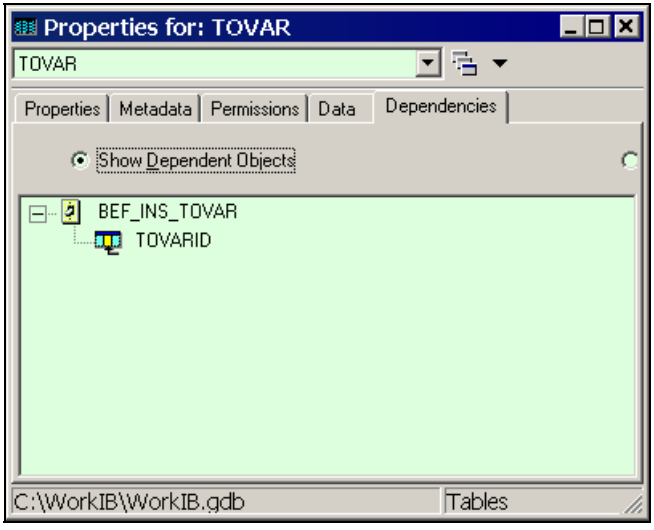


Рис. 3.23. Окно, содержащее свойства таблицы (в данном случае, созданный триггер, привязанный к событию ACTIVE BEFORE INSERT таблицы TOVAR)

Если выбрать вкладку **Metadata**, то можно увидеть текст SQL-запроса, который полностью описывает структуру таблицы `TOVAR`.

Замечание

Полезно иногда заглядывать на вкладку **Metadata**, так как с помощью нее можно легко восполнить пробелы в знаниях SQL.

Нам осталось создать еще две таблицы: "Фирмы" и "Продажи". Запускайте **Interactive SQL**.

Для создания справочника организаций пишем запрос на создание таблицы "Фирма":

```
CREATE TABLE "FIRM"
(
  "FIRMID"  INTEGER NOT NULL,
  "NAME"    VARCHAR(100) CHARACTER SET WIN1251 NOT NULL,

 PRIMARY KEY ("FIRMID")
 );
```

Выполняем этот запрос.

Создаем генератор для таблицы — пишем в **Interactive SQL**:

```
CREATE GENERATOR GEN_FIRM
```

Теперь устанавливаем начальное значение для генератора `GEN_FIRM`:

```
SET GENERATOR GEN_FIRM TO 0
```

Привязываем генератор `GEN_FIRM` к атрибуту `FIRMID` таблицы `FIRM` (создаем триггер):

```
SET TERM !! ;
CREATE TRIGGER "BEF_INS_FIRM" FOR "FIRM"
ACTIVE BEFORE INSERT
AS
BEGIN
 NEW.FIRMID=GEN_ID(GEN_FIRM,1);
END!!
SET TERM ; !!
```

Теперь точно таким же образом создадим таблицу "Продажи":

```
CREATE TABLE "SALE"
(
  "SALEID"   INTEGER NOT NULL,
  "FIRMKOD"  INTEGER NOT NULL,
```

```
  "TOVARKOD" INTEGER NOT NULL,
  "REM"      VARCHAR(100) CHARACTER SET WIN1251,

 PRIMARY KEY ("SALEID")
 );
```

Создаем генератор GEN_SALE для таблицы SALE:

```
CREATE GENERATOR GEN_SALE
```

Устанавливаем начальное значение для генератора GEN_SALE:

```
SET GENERATOR GEN_SALE TO 0
```

Привязываем генератор GEN_SALE к атрибуту SALEID таблицы SALE:

```
SET TERM !! ;
CREATE TRIGGER "BEF_INS_SALE" FOR "SALE"
ACTIVE BEFORE INSERT
AS
BEGIN
 NEW.SALEID=GEN_ID(GEN_SALE,1);
END!!
SET TERM ; !!
```

Таким образом мы создали базу данных.

Хочется отметить, что в InterBase 7.5 есть более простой способ создания таблиц, который я решил оставить на самостоятельное изучение читателя. Он заключается в том, что в левой части окна **IBConsole** выбирается компонент **Tables**, далее правой кнопкой мыши производится щелчок в правой части окна **IBConsole**. В выпадающем меню можно выбрать пункты:

- **Create** — создание новой таблицы. Появится дополнительное окно **Table Editor**, в котором задается имя таблицы, с помощью кнопки **Add Field** добавляются атрибуты, также можно создать первичные ключи и индексы;
- **Alter** — изменение существующей таблицы. Чтобы данный пункт меню был активным, необходимо производить щелчок правой кнопкой мыши на названии таблицы;
- **Drop** — удаление таблицы. Чтобы данный пункт меню был активным, необходимо производить щелчок правой кнопкой мыши на названии таблицы.

Замечание

Для того чтобы упростить процесс обучения, можно подключить учебные варианты базы данных. Если при инсталляции InterBase вы оставили путь, предло-

женный по умолчанию, то они будут находиться в каталоге C:\Program Files\Borland\InterBase\examples\database. Там есть две базы employee.gdb и intlemp.gdb, а также их резервные копии, имеющие расширение gbk. Чтобы подключить базы, необходимо выбрать пункт меню **Database\Register**.

3.6. Резервное копирование и восстановление базы данных

Очень важно регулярно выполнять резервное копирование базы данных. Не важно, работаете ли вы в крупной или небольшой организации, разрабатываете базу данных у себя на компьютере для диплома или курсового проекта. Никто не застрахован от случайностей и непредвиденных обстоятельств. Имея резервную копию, вы всегда можете восстановить базу данных в кратчайшие сроки, при этом до минимума сократив потраченное время и нервы. В организациях резервное копирование производится обычно каждый день. Если вы программируете у себя на компьютере, то наилучшим решением будет производить резервное копирование каждый раз, перед тем как вы соберетесь производить более или менее серьезные изменения в базе данных. Если что-то пойдет не так, то вы всегда сможете вернуть все назад.

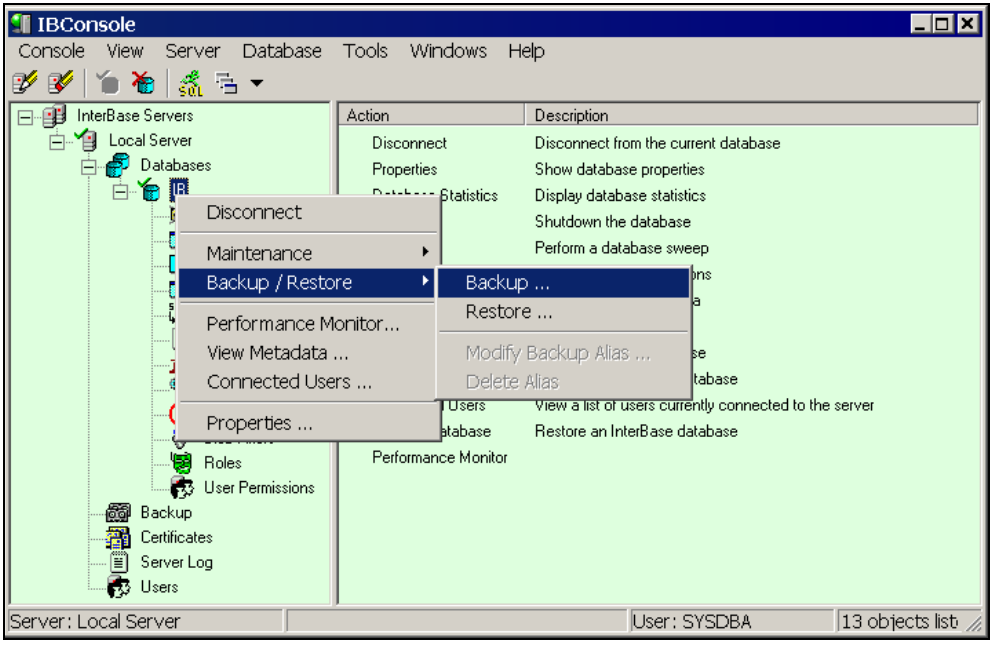


Рис. 3.24. Запуск резервного копирования

Для резервного копирования необходимо выбрать имя базы данных, для которой будет производиться операция (в нашем случае IB), щелкнуть на нем правой кнопкой мыши и в выпадающем меню выбрать пункт **Backup/Restore**, затем пункт **Backup** (рис. 3.24). Аналогичного эффекта можно добиться, выбрав пункт главного меню **Database\Maitenance\Backup/Restore\Backup**.

На экране появится окно **Database Backup**. В поле **Alias** группы элементов **Database** будет указана выбранная база данных — IB. В группе элементов **Backup File(s)** необходимо ввести IB в поле **Alias** (имя базы в резервной копии). В первой строчке списка **Filename(s)** указываем C:\WorkIB\backup.dat — это путь и имя файла для архивной копии (рис. 3.25).

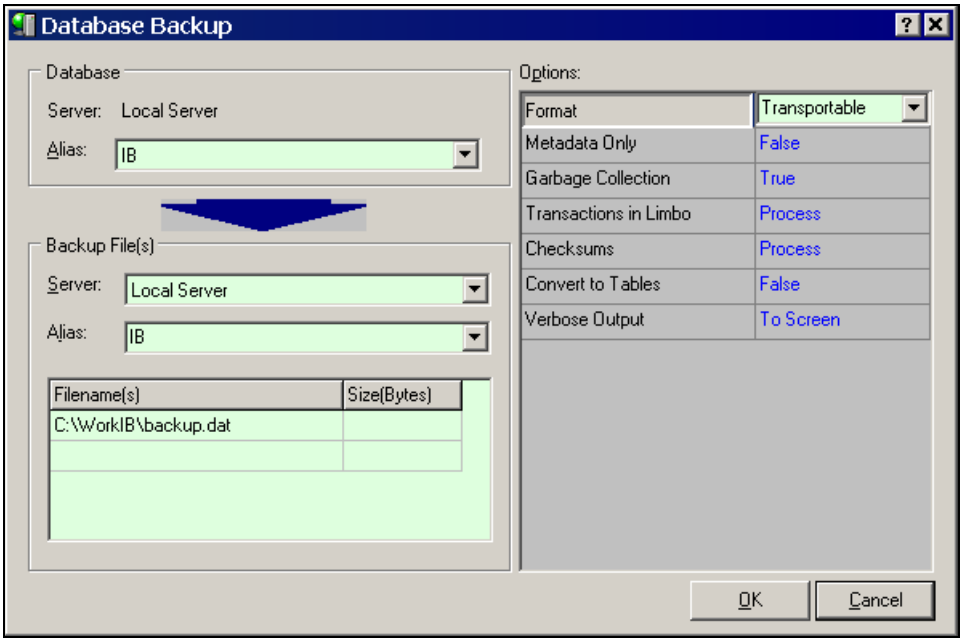


Рис. 3.25. Окно **DataBase Backup** с установленными параметрами

Замечание

Расширения для файла, содержащего архивную копию, может быть любым. Я использовал DAT.

При нажатии на кнопку **OK** начнется процесс резервного копирования. После его завершения появится окно **Information** (Информация), в котором сообщается, что резервное копирование завершено (рис. 3.26).

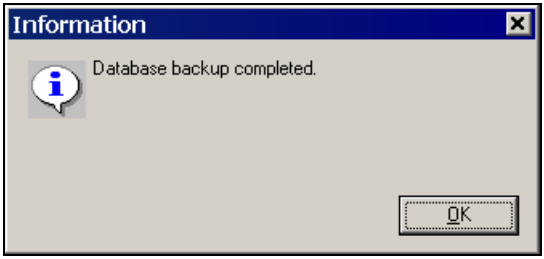


Рис. 3.26. Сообщение о том, что резервное копирование завершено

Нажимаем **OK**. Закрываем окно **Database Backup**, в котором отображается информация по резервному копированию (рис. 3.27).

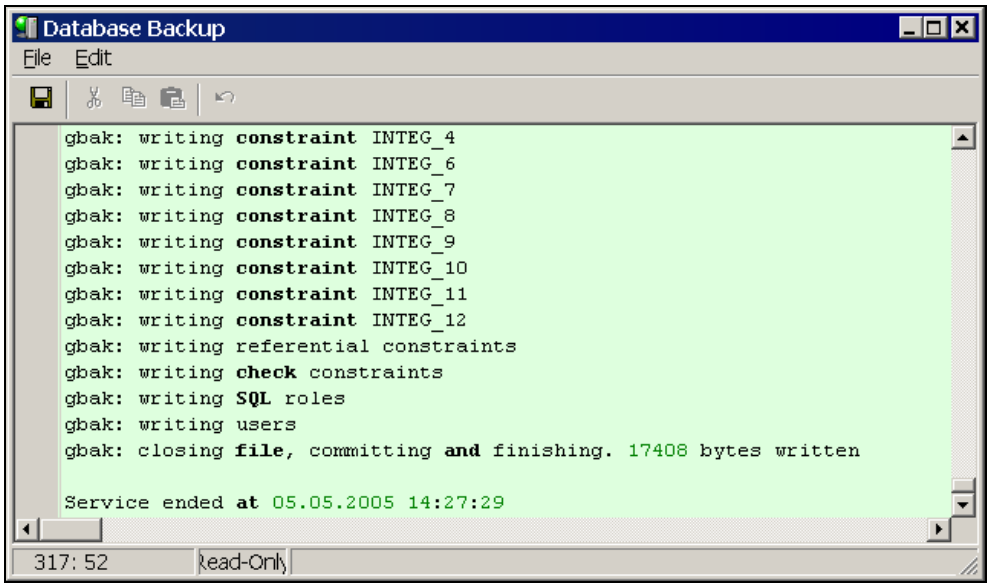


Рис. 3.27. Окно с информацией по резервному копированию

Для восстановления резервной копии необходимо сначала отключить соединение с данной базой. Для этого щелкаем на имени базы (в нашем случае имя IB) правой кнопкой мыши и выбираем из выпадающего меню пункт **Disconnect**. На экране появится окно **Confirm**, в котором будет спрашиваться, уверены ли мы в том, что хотим отключить соединение с выбранной базой данных (рис. 3.28). Нажимаем **Yes**.

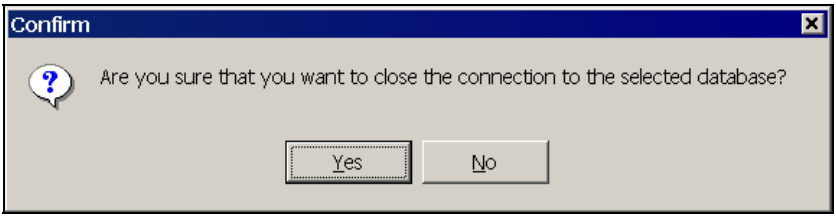


Рис. 3.28. Окно для подтверждения отключения базы данных

Далее выбираем пункт главного меню **Database\Maitenance\Backup/Restore\Restore**. На экране появится окно **Database Restore**. В поле **Alias** группы элементов **Backup File(s)** будет указана выбранная база данных — IB. В этой же группе элементов в списке **Filename(s)** будет указано месторасположение резервной копии C:\WorkIB\backup.dat.

В группе элементов **Database** будут указаны:

- тип сервера — Local Server;
- имя, под которым восстановится база, — IB;
- путь, куда будет происходить восстановление, — C:\WorkIB\WorkIB.gdb.

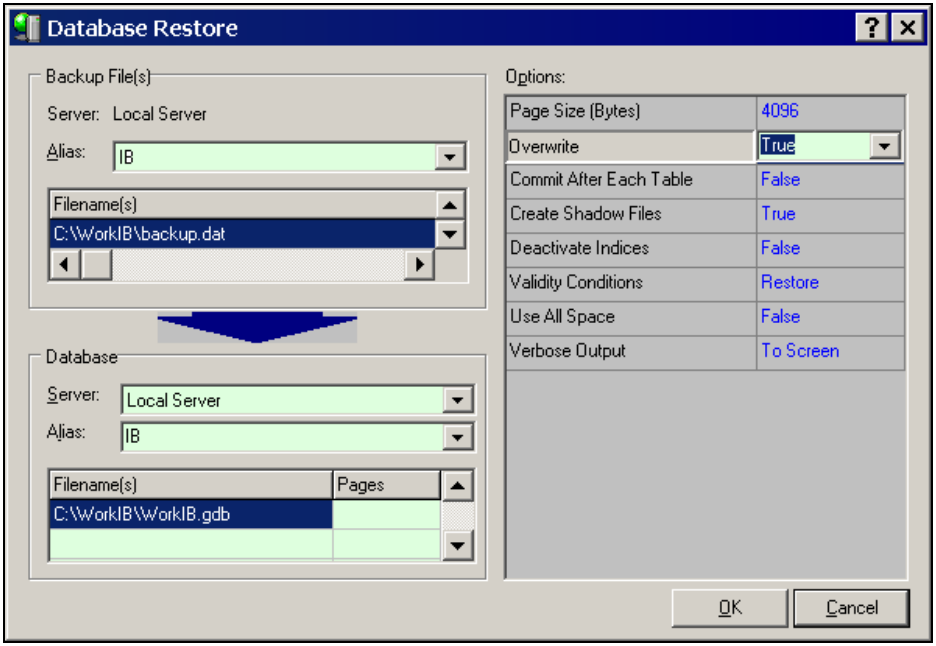


Рис. 3.29. Окно **Database Restore** с установленными параметрами для восстановления резервной копии

Как видите, InterBase самостоятельно заполнил все необходимые параметры. Нам необходимо только установить в поле **Overwrite** группы элементов **Options** значение `True`, которое будет указывать на то, что база данных будет переписываться из резервной копии (рис. 3.29).

При нажатии **OK** начнется процесс восстановления базы данных из резервной копии. После его завершения появится окно **Database Restore**, в котором можно будет увидеть запись **Service ended**, означающую, что операция прошла успешно (рис. 3.30). Теперь можно закрыть окно **Database Restore**.

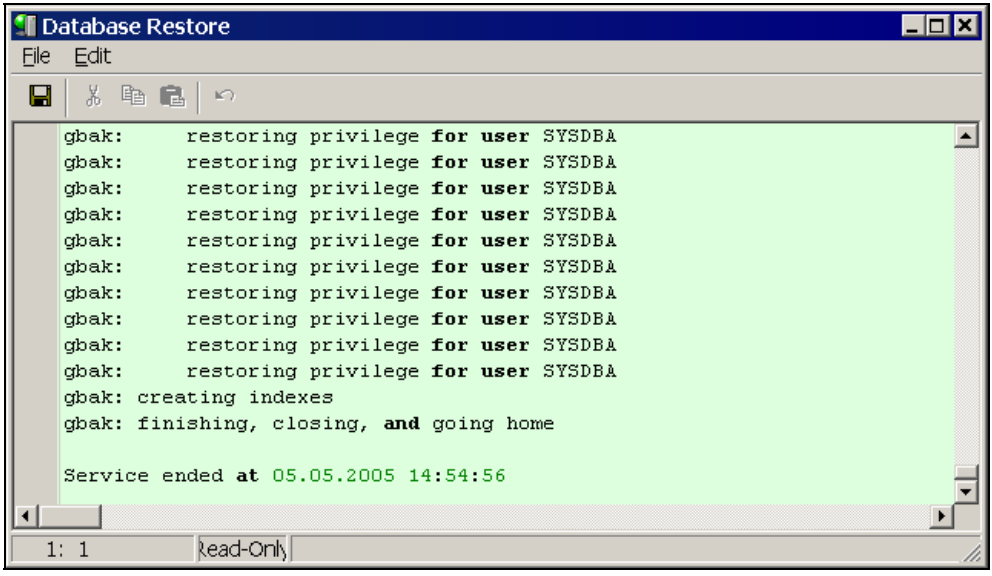


Рис. 3.30. Окно **Database Restore** после завершения операции восстановления базы

3.7. Разработка приложения "клиент-сервер" в Delphi

В данном разделе мы будем использовать следующие компоненты, описанные в табл. 3.4, 3.5, 3.6 и 3.7.

Таблица 3.4. *Основные компоненты вкладки* ***InterBase***

Название	Основные свойства	Комментарии
IBDatabase		Отвечает за связь с базой данных
	DatabaseName	Имя базы данных и путь к ней
	Params	Параметры подключения к базе: имя пользователя и пароль
	LoginPromt	Логическое поле. Отвечает, будет ли отображаться окно ввода имени пользователя и пароля каждый раз, когда происходит соединение с базой данных
	Connected	Логическое свойство. Если содержит True, то связь с базой данных установлена, в противном случае — нет
IBTransaction		Отвечает за работу всех транзакций для связанной с данным компонентом базой данных
	DefaultDatabase	Содержит имя компонента IBDatabase, транзакциями которого будет осуществляться управление
IBQuery		Позволяет отправлять запросы к базе данных и получать результат их выполнения
	DatabaseName	Содержит имя компонента IBDatabase. Соответственно все запросы будут отправляться базе данных, прописанной в компоненте IBDatabase
	SQL	SQL-запрос для работы с базой
	Active	Логическое свойство. Когда установлено в True, SQL-запрос выполняется и результат передается в клиентское приложение (туда, откуда он был послан)

Таблица 3.5. *Основные компоненты вкладки* ***DataAccess***

Название	Основные свойства	Комментарии
DataSource		Служит посредником между компонентами визуализации данных (таких как DBGrid и DBEdit) и источниками данных (таких как IBQuery). Без данного компонента увидеть данные на экране не удастся

Таблица 3.5 *(окончание)*

Название	Основные свойства	Комментарии
	DataSet	Содержит имя компонента источника данных (например, IBQuery1)

Таблица 3.6. *Основные компоненты вкладки* ***DataControl***

Название	Основные свойства	Комментарии
DBGrid		Компонент, отображающий содержимое таблицы или результата запроса в виде двухмерной таблицы, разделенной сеткой
	DataSource	Имя компонента-посредника DataSource.
DBNavigator		Набор кнопок для работы с данными, содержит кнопки перехода между записями таблицы, а также кнопки для подтверждения либо отказа от изменений
	DataSource	Имя компонента-посредника типа DataSource. Указывается для того, чтобы привязать кнопки к определенному набору данных

Таблица 3.7. *Основные компоненты вкладки* ***Standart***

Название	Основные свойства	Комментарии
MainMenu		Данный компонент организует главное меню для формы, на которой он расположен
	Items	В свойстве хранятся пункты и подпункты меню
Button		Обычная кнопка
	Caption	Определяет текст кнопки
	Font	Определяет параметры шрифта текста кнопки

Замечание

Если быть максимально точным, то, когда речь заходит о компонентах, правильно писать их с буквой `T` в начале, например, `TLabel`, `TEdit`. А `Label` и `Edit` — это экземпляры компонентов (объекты). Но я специально отказался от данной записи. Во-первых, разница между понятиями не столь существенна, во-вторых, многие авторы используют упрощенный вариант записи (без буквы `T`).

Запускаем Delphi. Создаем новое приложение через пункт меню **File\New\Application** (для Delphi 2005 выбираем **File\New\VCL Forms Application Delphi for Win32**). Свойство `Name` формы меняем на `MainForm`. Нажимаем кнопку **Сохранить все** (можно воспользоваться пунктом меню **File\Save All**). Выбираем рабочую папку (у меня это C:\WorkIB). Модуль `Unit1` сохраняем под именем `UMain`, а модуль приложения под именем `WorkIB`.

Выбираем пункт меню **Project\Options**, заходим на вкладку **Application**. В поле **Title** группы элементов **Application Settings** вводим "Учет продаж". Это название будет отображаться в **Панели задач** при запуске приложения. Нажимаем **OK**.

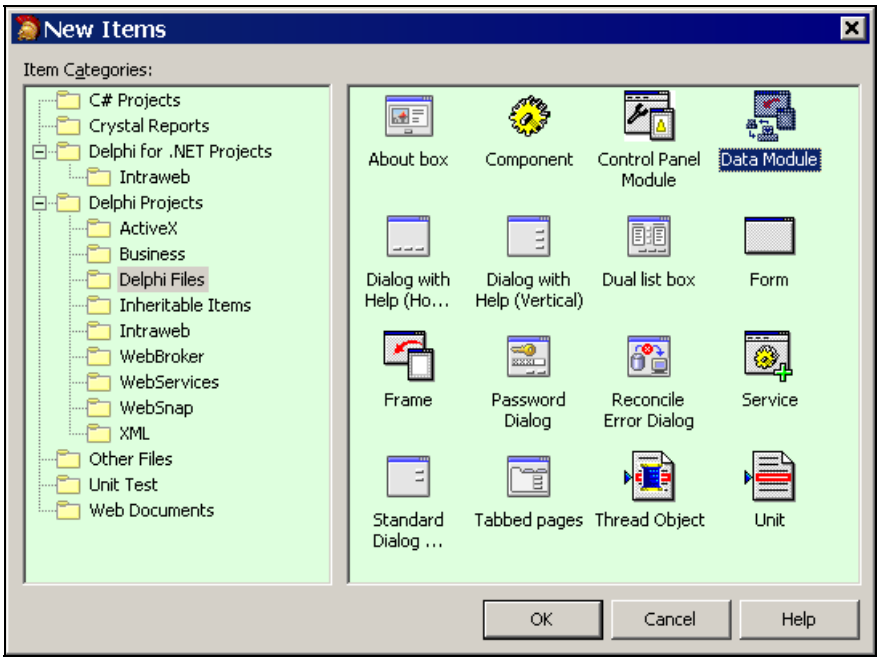


Рис. 3.31. Окно **New Items** для создания DataModule в Delphi 2005

Создаем новый модуль с данными: `DataModule`. Для этого выбираем пункт меню **File\New\DataModule** для Delphi 7 (для Delphi 2005 необходимо вы-

брать пункт меню **File\New\Other**, в появившемся окне **New Items** нужно выбрать **Delphi Files** в группе элементов **Items Categories** и затем щелкнуть левой кнопкой мыши на пиктограмме **DataModule**, после этого нажать кнопку **OK**, рис. 3.31).

В свойство `Name` пишем `DM`. Выбираем пункт меню **Files\Save All** (сохранить все) и называем модуль `UDM` (рис. 3.32).

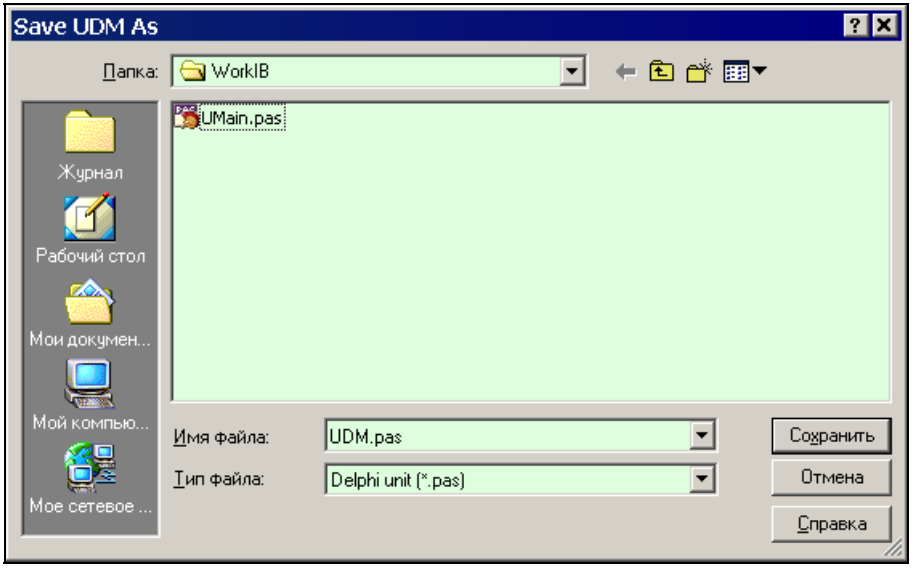


Рис. 3.32. Окно сохранения модуля в Delphi 2005

Помещаем на `DataModule` компоненты `IBDatabase` и `IBTransaction` с вкладки InterBase (рис. 3.33 и 3.34). В свойство `DefaultDatabase` компонента `IBTransaction` устанавливаем значение `IBDatabase1`.

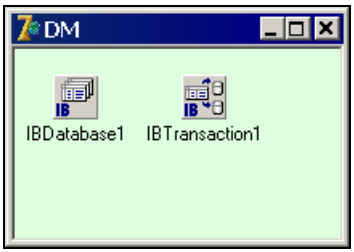


Рис. 3.33. Окно **DataModule** в Delphi 7

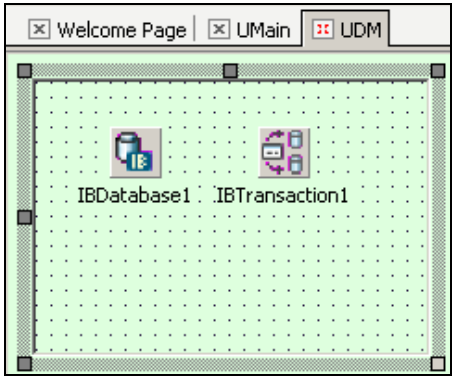


Рис. 3.34. Окно **DataModule** в Delphi 2005

Для компонента `IBDatabase1` устанавливаем следующие свойства:

- `DatabaseName` — указываем файл нашей базы, у меня C:\WorkIB\WORKIB.GDB;
- В свойстве `Params` пишем:

  ```
  USER_NAME=SYSDBA
  PASSWORD=masterkey
  ```

Таким образом, мы указываем, что хотим работать от имени администратора. Свойство `LoginPrompt` ставим в `False`, чтобы каждый раз при запуске программы не появлялось окошко ввода имени пользователя и его пароля. Свойство `Connect` устанавливаем в `True`. После этого связь с базой данных будет установлена.

Помещаем на форму `MainForm` компонент `MainMenu`, расположенный на вкладке **Standart**. Производим на нем двойной щелчок левой кнопкой мыши и создаем пункты меню как на рис. 3.35.

Создаем новую форму (Delphi 7: **File\New\Form**; Delphi 2005: **File\New\Form — Delphi for Win32**), называем ее `FormTovar`. В свойство `Caption` пишем "Справочник — товар". Сохраняем модуль под именем `UTovar`.

Помещаем на форму `FormTovar` компоненты `DBGrid`, `DBNavigator`, 2 компонента `Button`, `IBQuery`, `DataSource` (рис. 3.36).

Свойство `Enabled` кнопки "Выбрать" устанавливаем в `False`, чтобы данная кнопка была доступна только в определенные моменты времени, которые мы определим сами.

Для компонента `DBNavigator` свойство `VisibleButtons` делаем, как показано на рис. 3.37 и 3.38.

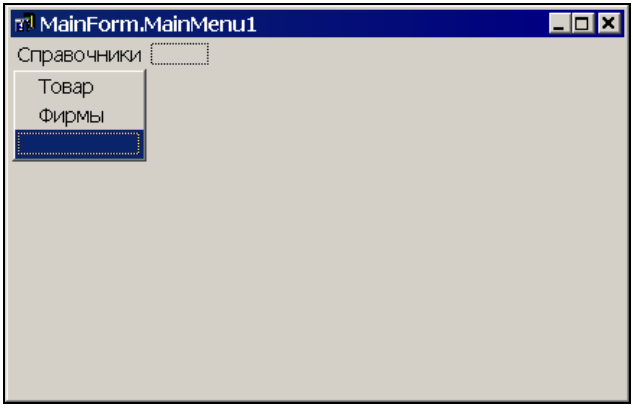


Рис. 3.35. Структура меню
для создаваемого приложения

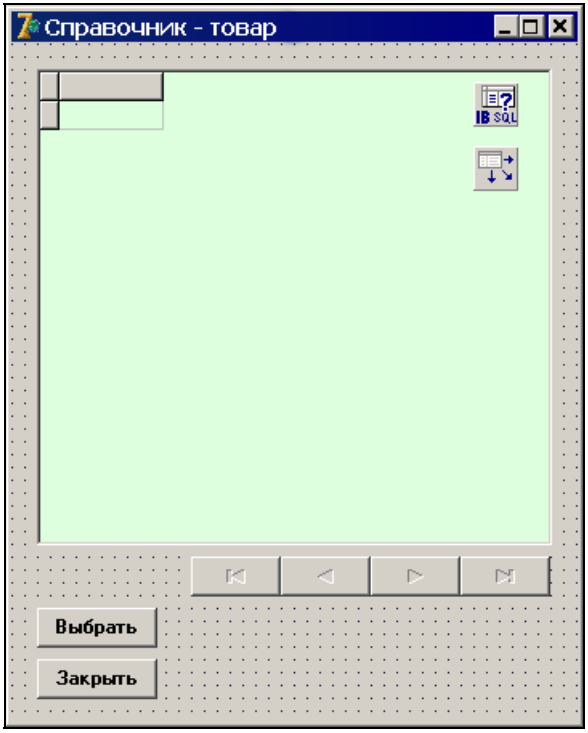


Рис. 3.36. Вид окна **Справочник - товар**
с размещенными компонентами

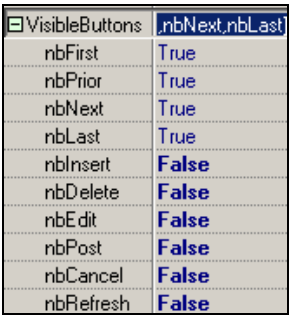


Рис. 3.37. Настройка свойства `VisibleButtons` компонента `DBNavigator` для Delphi 7

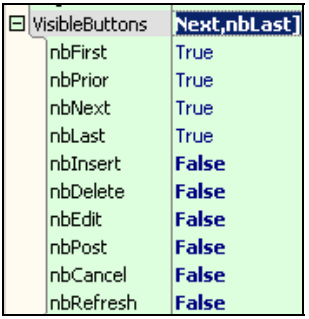


Рис. 3.38. Настройка свойства `VisibleButtons` компонента `DBNavigator` для Delphi 2005

В свойстве `DataSet` компонента `DataSource` устанавливаем значение `IBQuery1`. В свойстве `DataSource` компонентов `DBGrid` и `DBNavigator` ставим `DataSource1`.

Выбираем пункт меню **File\USE Unit** (рис. 3.39).

Выбираем `UDM` и нажимаем **OK**. Это нужно, чтобы мы смогли подключить компонент `IBQuery1` к компоненту `IBDatabase`, так как `IBDatabase` расположен в совершенно другом модуле — `UDM`, а сейчас мы работаем с модулем `UTovar`. В свойстве `Database` компонента `IBQuery1` устанавливаем `DM.IBDatabase1`. Теперь заходим в свойство `SQL` этого же компонента, появится окно **Command Text Editor**. В этом окне пишем запрос, как показано на рис. 3.40.

Данный запрос означает выбрать (команда `SELECT`) все данные из таблицы "Товар" и отсортировать их (команда `ORDER`) по названию товара (атрибут `Name`).

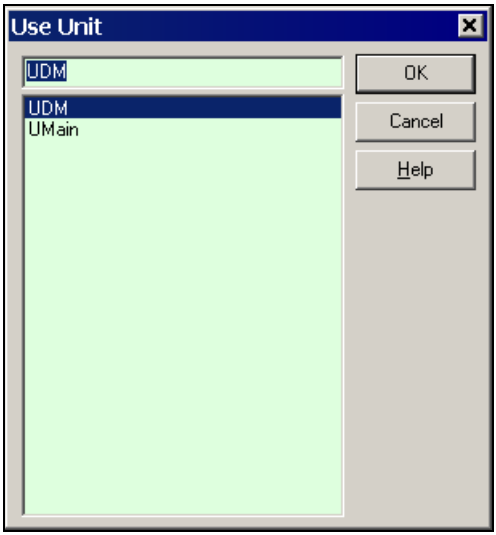


Рис. 3.39. Окно **Use Unit**,
которое предназначено для подключения модулей

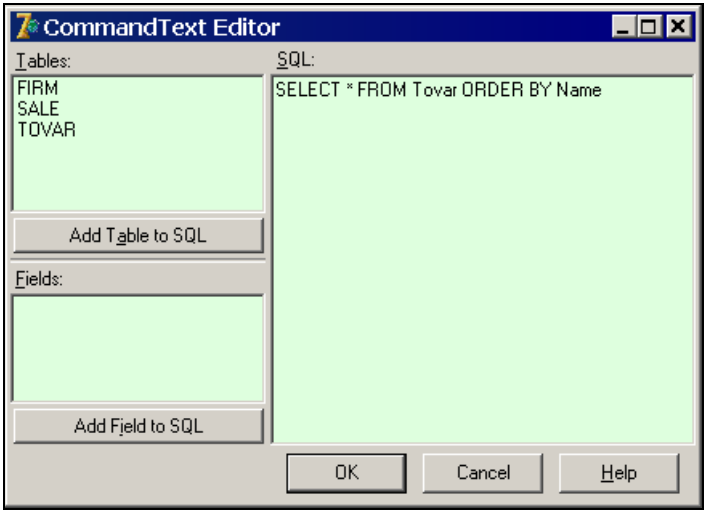


Рис. 3.40. Окно **CommandText Editor**
предназначено для ввода SQL-запроса

Щелкаем два раза левой кнопкой мыши по компоненту `IBQuery1`. Щелкаем правой кнопкой мыши в появившемся окне и выбираем **Add all fields** (рис. 3.41).

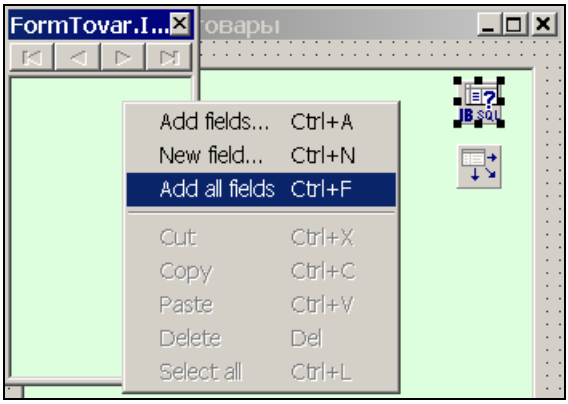


Рис. 3.41. Окно для выбора всех доступных атрибутов

После этого действия данное окно примет вид, как показано на рис. 3.42.

Рис. 3.42. Окно выбора полей

Выбираем `TOVARID` и в инспекторе объектов свойство `Visible` ставим в `False`. Теперь этот атрибут не будет отображаться на экране в сетке `DBGrid` во время работы с программой.

Для атрибута `NAME` в свойство `DisplayLabel` пишем "Название товара". В свойство `DisplayWidth` ставим 30 — это свойство отвечает за ширину атрибута в `DBGrid`. Выбираем атрибут `PRICE`, в свойство `DisplayLabel` пишем "Цена".

Щелкаем два раза на кнопке `Button` с текстом "Закрыть" и пишем код для закрытия формы "Справочник — товары":

```
FormTovar.Close;
```

Выбираем пункт меню **Project\Option** и вкладку **Forms**. И с помощью стрелок переносим `FormTovar` в правую половину окна. Таким образом, мы говорим Delphi, что во время работы программы мы будет сами следить за созданием этой формы и ее удалением из памяти компьютера (рис. 3.43).

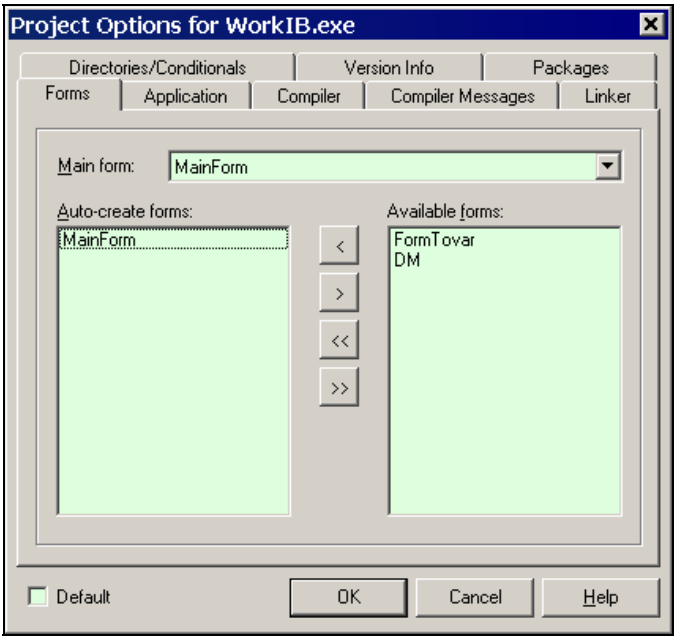


Рис. 3.43. Окно для настройки способа создания форм во время запуска приложения

Нажимаем **OK**.

Делаем активной форму `MainForm` и для пункта меню **Справочники\Товар** пишем код, предварительно подключив модуль `UTovar` (это делается с помощью пункта меню **File\Use Unit**):

```
FormTovar:=TFormTovar.Create(self); {Создаем форму}
FormTovar.IBQuery1.Open; {Открываем набор данных}
FormTovar.ShowModal; {Отображаем форму на экране}
```

Переходим на форму `FormTovar` и в событии `OnClose` пишем код для закрытия набора данных перед закрытием формы:

```
FormTovar.IBQuery1.Close; {Закрыть набор данных, если закрывают форму}
```

Помещаем на форму `FormTovar` еще один компонент `IBQuery` (он будет называться `IBQuery2`), в свойство `DATABASENAME` пишем `DM.IBDATABASE1`.

В свойстве SQL компонента IBQuery2 пишем запрос для добавления данных в таблицу "Товар":

```
INSERT INTO TOVAR (NAME, PRICE)
VALUES (:NAME,:PRICE);
```

Данный запрос означает добавить (команда INSERT INTO) новую запись в таблицу TOVAR, а именно в атрибуты Name и Price, которые содержатся в параметрах :Name и :Price.

Замечание

Параметры легко можно отличить от остальных элементов по двоеточию, стоящему перед их именем.

Щелкаем по свойству Params компонента IBQuery2, появляется окно **Editing** (рис. 3.44).

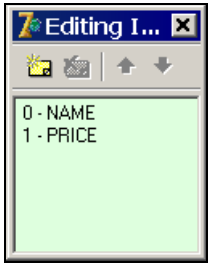


Рис. 3.44. Окно **Editing** компонента IBQuery2

Выделяем запись "0 – Name" и в инспекторе объектов ставим в свойстве DataType значение ftString. В свойстве Value указываем "AAA". DataType — тип данных, который будет содержать параметр, в данном случае он текстовый. Value — значение по умолчанию, если параметру не присвоено никакого значения (предположим, произошел какой-то сбой в системе), то ему будет присвоено "AAA".

Выделяем запись "1 - Price" и в инспекторе объектов ставим в свойство DataType значение ftInteger — значение параметра целое число, а в свойстве Value ставим значение по умолчанию 0.

Располагаем на форме еще одну кнопку Button, в свойстве Caption пишем "+". В событие OnClick пишем код для добавления новой записи в таблицу "Товар":

```
procedure TFormTovar.Button3Click(Sender: TObject);
var
{Объявляем переменные}
```

```
 s:string; {для названия товара}
 c:integer; {для цены товара}
begin
 {Используем стандартную функцию InputBox,
 которая выводит на экран окно с полем для
 ввода информации}
 s:=InputBox('Введите данные','Введите название товара','');
 c:=StrToInt(InputBox('Введите данные','Введите цену товара',''));

 {Передаем в параметр Name введенное название товара}
 IBQuery2.Params.ParamByName('Name').Value:=s;
 {Передаем в параметр Price введенную цену товара}
 IBQuery2.Params.ParamByName('Price').Value:=c;

 {Пытаемся выполнить запрос на вставку записи,
 используем защитную конструкцию Try - except}
 try
{Выполняем запрос, методом ExecSQL,
 т.к. запрос не будет возвращать данных}
  IBQuery2.ExecSQL;

 {Если не удалось, то сообщить пользователю}
 except
   {Выводим окно с текстом с помощью функции ShowMessage}
   ShowMessage('Ошибка при добавлении данных!,'+#13+
               'Попробуйте еще раз!');
   {откатываем изменение}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры с помощью exit}
   exit;
 end;
 {Если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {Обновляем набор данных}
 IBQuery1.Close;
 IBQuery1.Open;

end;
```

Помещаем на форму еще один компонент IBQuery (его имя будет IBQuery3), свойство DATABASENAME устанавливаем в DM.IBDATABASE1.

В свойстве SQL IBQuery3 пишем код для удаления записи из таблицы "Товар":

```
DELETE FROM Tovar
WHERE TovarID=:Par1
```

Данный запрос означает: удалить (команда DELETE) из таблицы TOVAR товар с кодом (запись, у которой атрибут TovarID), равным значению параметра.

Щелкаем по свойству Params компонента IBQuery3, появляется окно **Editing** (рис. 3.45).

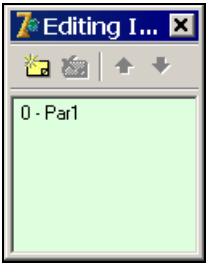


Рис. 3.45. Окно **Editing** компонента IBQuery3

Выделяем запись "0 -Par1" и в инспекторе объектов ставим в свойстве DataType значение ftInteger, в свойстве Value пишем значение по умолчанию 0.

Помещаем на форму еще одну кнопку Button, в свойство Caption пишем "—". В событие OnClick пишем код удаления записи из таблицы "Товар":

```
procedure TFormTovar.Button4Click(Sender: TObject);
begin
 {Спрашиваем пользователя, правда ли он хочет удалить запись,
 используется стандартная функция MessageDlg для вывода
 окна с кнопками Yes и No}
 if MessageDlg('Вы уверены, что хотите удалить запись?',
                mtConfirmation,[mbYes,mbNo],0)=mrYes then
 {если да, то}
 begin
  {запоминаем код выбранной записи и помещаем это
  значение в параметр Par1}
  IBQuery3.Params.ParamByName('Par1').Value:=
    IBQuery1TOVARID.Value;
```

```
  {Пытаемся удалить запись}
  try
     {Выполняем запрос методом ExecSQL,
     т.к. запрос не будет возвращать данных}
     IBQuery3.ExecSQL;

 {если не получилось, то сообщаем об этом пользователю}
 except
   {Выводим окно с текстом с помощью функции ShowMessage}
   ShowMessage('Удаление не прошло!'+#13+
               'Запись заблокирована либо уже удалена!');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры с помощью exit}
   exit;
 end;

 {Если все нормально, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {Обновляем набор данных}
 IBQuery1.Close;
 IBQuery1.Open;
 end;

end;
```

На рис. 3.46 представлен окончательный вариант формы "Справочник — товар".

Одну форму мы полностью закончили. Дальше будет легче.

Создаем новую форму, называем ее `FormFirm`. В свойство `Caption` пишем "Справочник — фирмы". Сохраняем под именем `UFirm`.

Помещаем компоненты `DBGrid`, `DBNavigator`, 2 компонента `Button`, `IBQuery`, `DataSource` (рис. 3.47).

Свойство `Enabled` кнопки "Выбрать" установите в `False`. Для компонента `DBNavigator` свойство `VisibleButtons` сделайте таким же, как в прошлый раз. В свойство `DataSet` компонента `DataSource` устанавливаем `IBQuery1`.

В свойство DataSource компонентов DBGrid и DBNavigator ставим DataSource1.

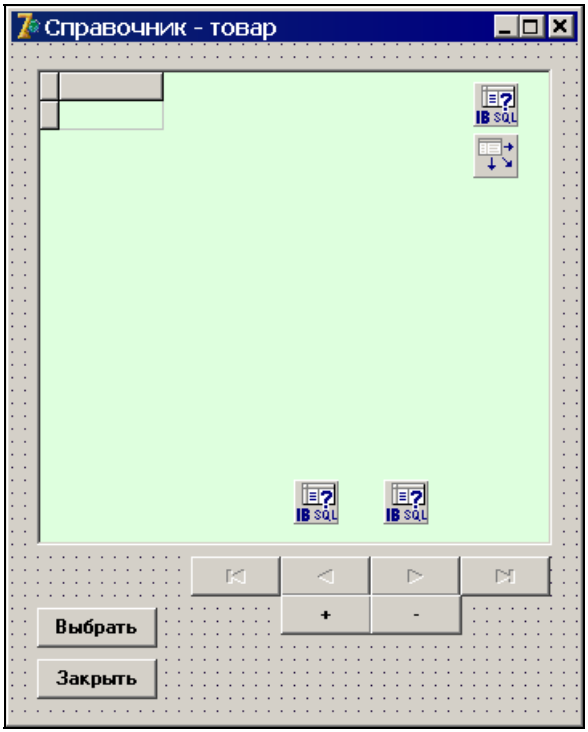


Рис. 3.46. Окончательный вариант формы "Справочник — товар"

Выбираем пункт меню **File\USE Unit**. В появившемся окне **Use Unite** выбираем UDM и нажимаем **OK**. Это нужно, чтобы мы смогли подключить компонент IBQuery1 к компоненту IBDatabase, так как они находятся на разных формах. В свойстве Database компонента IBQuery1 ставим DM.IBDatabase1.

Теперь заходим в свойство SQL этого же компонента. И пишем в окне **Command Text Editor** запрос для выборки всех записей из таблицы Firm:

```
SELECT * FROM Firm ORDER BY Name
```

Данный запрос означает выбрать (команда SELECT) все данные из таблицы "Фирмы" и отсортировать их (команда ORDER) по названию фирмы (атрибут Name).

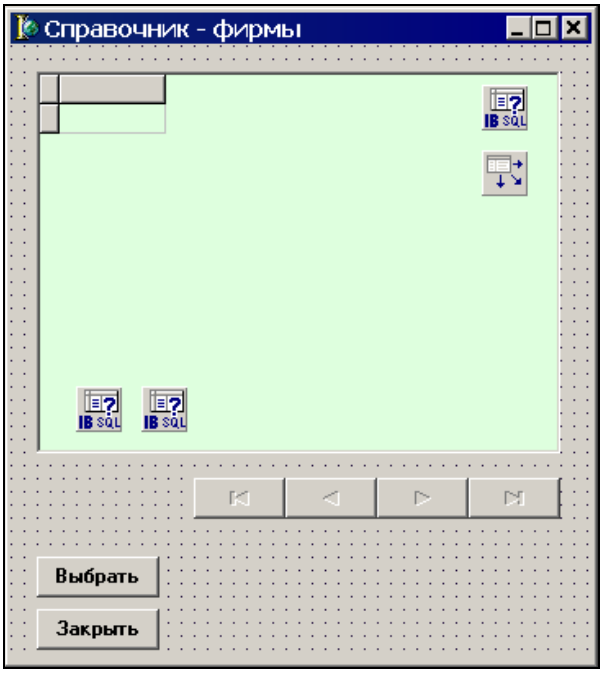


Рис. 3.47. Форма "Справочник — фирмы"

Щелкаем два раза левой кнопкой мыши по компоненту `IBQuery1`. Щелкаем правой кнопкой мыши в появившемся окне и выбираем **Add all fields** (рис. 3.48). После этого приступим к настройке отображения атрибутов.

Рис. 3.48. Окно **Add all fields** компонента `IBQuery1`

Выбираем FIRMID и в инспекторе объектов в свойство Visible ставим False. Теперь этот атрибут не будет отображаться на экране во время работы программы.

Для атрибута Name в свойство DisplayLabel пишем "Название фирмы". В свойство DisplayWidth ставим 30 — это свойство отвечает за ширину поля в DBGrid.

Щелкаем два раза на кнопке Button с текстом "Закрыть" и пишем код для закрытия формы "Справочник — фирмы":

```
FormFirm.Close;
```

Выбираем пункт **Project\Option** главного меню. И с помощью стрелок переносим FormFirm в правую половину окна; таким образом мы указываем Delphi, что во время работы программы мы будем сами следить за созданием этой формы и ее удалением. После этого нажимаем **OK**.

Делаем активной форму MainForm и для пункта меню **Справочники\Фирмы** пишем код для создания формы "Справочник — фирмы" (предварительно нам надо будет подключить модуль UFirmTovar):

```
FormFirm:=TFormFirm.Create(self); {Создаем форму}
FormFirm.IBQuery1.Open; {Открываем набор данных}
FormFirm.ShowModal; {Отображаем форму на экране}
```

Переключаемся на форму FormFirm и в событии OnClose пишем:

```
{Закрыть набор данных, если закрывают справочник}
FormFirm.IBQuery1.Close;
```

Помещаем на форму компонент IBQuery (он будет называться IBQuery2), в свойство DATABASENAME пишем DM.IBDATABASE1. В свойство SQL пишем запрос для добавления данных в таблицу "Фирма":

```
INSERT INTO Firm (Name)
VALUES (:Name);
```

Данный запрос означает добавить (команда INSERT INTO) новую запись в таблицу FIRM (а именно в атрибут Name), добавляемые данные содержатся в параметре :Name.

Щелкаем по свойству Params компонента IBQuery2, в появившемся окне **Editing** выделяем запись Name и в инспекторе объектов ставим в свойство DataType значение ftString, в свойство Value ставим значение по умолчанию "БББ".

Располагаем на форме еще одну кнопку Button, в свойство Caption пишем "+". В событие OnClick кнопки пишем код для добавления новой записи в таблицу "Фирмы":

```
procedure TFormFirm.Button3Click(Sender: TObject);
var
 {Объявляем переменные}
 s:string; {для названия фирмы}
begin
 {Запрашиваем у пользователя данные}
 s:=InputBox('Введите данные','Введите название организации','');

 {Передаем в параметр Name введенное название фирмы}
 IBQuery2.Params.ParamByName('Name').Value:=s;

 {пытаемся выполнить запрос на вставку записи,
 используем защитную конструкцию}
 try
  IBQuery2.ExecSQL;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Добавление не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 IBQuery1.Close;
 IBQuery1.Open;
end;
```

Помещаем на форму еще один компонент IBQuery (его имя будет IBQuery3), в свойство DATABASENAME устанавливаем DM.IBDATABASE1.

В свойстве `SQL` компонента `IBQuery3` пишем запрос для удаления данных из таблицы "Фирмы":

```
DELETE FROM Firm
WHERE FirmID=:Par1
```

Данный запрос означает удалить (команда `DELETE`) из таблицы `Firm` фирму с кодом (запись, у которой атрибут `FirmID`), равным значению параметра.

Щелкаем по свойству `Params` компонента `IBQuery3`, появляется окно **Editing**. Выделяем запись "`0 -Par1`" и в инспекторе объектов ставим в свойстве `DataType` значение `ftInteger`, в свойстве `Value` пишем значение по умолчанию 0.

Помещаем на форму еще одну кнопку `Button`, в свойство `Caption` пишем "—". В событие `OnClick` пишем код для удаления записи из таблицы "Фирмы":

```
procedure TFormFirm.Button4Click(Sender: TObject);
begin
 {Спрашиваем пользователя, правда ли он хочет удалить запись,
 используется стандартная функция MessageDlg для вывода
 окна с кнопками Yes и No}
 if MessageDlg('Вы уверены, что хотите удалить запись?',
                mtConfirmation,[mbYes,mbNo],0)=mrYes then
 {если да, то}
 begin
  {запоминаем код выбранной записи и помещаем это
  значение в параметр Par1}
  IBQuery3.Params.ParamByName('Par1').Value:=
     IBQuery1FIRMID.Value;

  {Пытаемся удалить запись}
  try
  {Выполняем запрос методом ExecSQL}
  IBQuery3.ExecSQL;

  {если не получилось, то сообщаем об этом пользователю}
  except
   {Выводим окно с текстом с помощью процедуры ShowMessage}
   ShowMessage('Удаление не прошло!'+#13+
               'Запись заблокирована, либо уже удалена!');
   {откатывем транзакцию}
```

```
   DM.IBTransaction1.RollbackRetaining;
  {выходим из процедуры с помощью exit}
   exit;
  end;

  {Если все нормально, то подтверждаем транзакцию}
  DM.IBTransaction1.CommitRetaining;

 {Обновляем набор данных}
 IBQuery1.Close;
 IBQuery1.Open;
 end;

 end;
```

На рис. 3.49 представлен окончательный вариант формы "Справочник — фирмы".

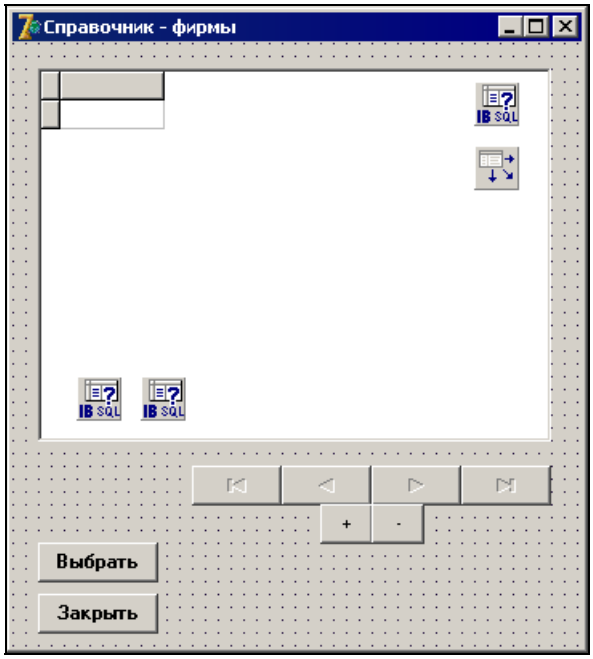


Рис. 3.49. Окончательный вариант формы "Справочник — фирмы"

Разработка последнего справочника завершена. Теперь перейдем к самому интересному. Будем кодировать процесс продажи товара.

Переходим на форму `MainForm`, располагаем на ней компоненты `DBGrid`, `IBQuery`, `DataSource` и три компонента `Button` (рис. 3.50).

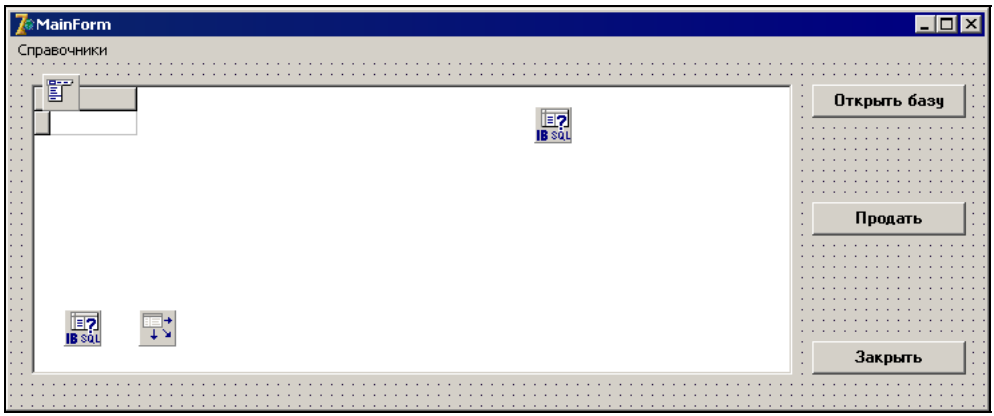


Рис. 3.50. Форма `MainForm`

У кнопок `Button` в свойстве `Caption` пишем: "Открыть базу", "Продать", "Закрыть", как показано на рис. 3.50. Выбираем пункт меню **File\USE Unit**. Выбираем `UDM` и нажимаем **OK**.

В свойстве `Database` компонента `IBQuery1` устанавливаем `DM.IBDatabase1`. Теперь заходим в свойство `SQL` этого же компонента. Появится окно **Command Text Editor**, в котором необходимо написать запрос для выборки записей из таблиц "Продажи", "Товар", "Фирмы" по условию:

```
SELECT * FROM Sale,Tovar,Firm
WHERE
      Sale.FIRMKOD=FIRM.FIRMID
                  AND
      Sale.TOVARKOD=TOVAR.TOVARID
```

Данный запрос означает выбрать (команда `SELECT`) все данные по продажам (таблицы `SALE`, `TOVAR`, `FIRM`). Причем нужно связать таблицы `Sale` и `Firm` по коду организации (`Sale.FIRMKOD=FIRM.FIRMID`), а таблицы `Sale` и `Tovar` — связать по коду товара (`Sale.TOVARKOD=TOVAR.TOVARID`). Таким образом, с помощью этого запроса мы связываем все три таблицы.

Щелкаем два раза по компоненту `IBQuery1` и настраиваем атрибуты для отображения в приложении.

В свойство DataSet компонента DataSource устанавливаем значение IBQuery1. А в свойство DataSource компонента DBGrid устанавливаем DataSource1.

Щелкаем два раза по кнопке Button с текстом "Открыть базу" и здесь напишем простую процедуру для аутентификации пользователя при попытке открыть таблицу "Продажи":

```
procedure TMainForm.Button1Click(Sender: TObject);
var
 s:string; {Хранит введенный пароль}
begin
 {Используем стандартную функцию InputBox,
 которая выводит на экран окно с полем для
 ввода информации; таким образом мы
 будем запрашивать пароль для открытия базы}
 s:=InputBox('Аутентификация','Введите пароль:','');
 {если пароль правильный (паролем является слово Parol), то}
 if s='Parol' then
 begin
   {открываем базу продаж}
   IBQuery1.Open;
   {кнопку "Открыть базу" делаем недоступной}
   Button1.Enabled:=false;
 end
 else
   {если пароль неправильный, то выводим
   пользователю информационное сообщение}
   ShowMessage('Неправильный пароль!');

end;
```

Теперь случайный человек не сможет получить доступ к базе продаж.

В событие OnClose формы MainForm пишем код для закрытия базы продаж при выходе из программы:

```
procedure TMainForm.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 {Если выходим из программы и база продаж открыта,
 то закрыть ее}
```

```
 if IBQuery1.Active then
   IBQuery1.Close;
end;
```

Располагаем на форме `MainForm` компонент `IBQuery`, задаем ему свойство `DatabaseName`, в свойстве `SQL` пишем запрос для добавления данных в таблицу "Продажи":

```
INSERT INTO SALE (FIRMKOD, TOVARKOD, REM)
VALUES (:FIRMKOD,:TOVARKOD,:REM);
```

Задаем свойство `Params`. Первые два параметра типа `ftInteger` со значением по умолчанию 0. Третий параметр `ftString`, значение по умолчанию — "BBB".

Для кнопки `Button` с текстом "Продать" пишем код для создания формы `FormSale`:

```
{Создание и отображение формы FormSale}
FormSale:=TFormSale.Create(self);
FormSale.ShowModal;
```

Создаем новую форму, называем ее `FormSale`, сохраняем под именем `USale`. Располагаем на ней компоненты, как показано на рис. 3.51 (компонент с тремя точками — это обычный `Button`).

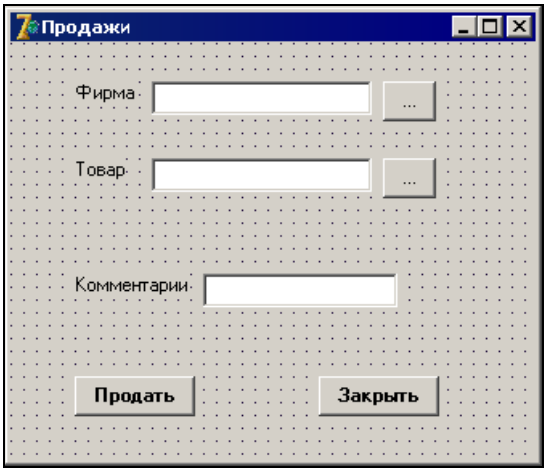


Рис. 3.51. Форма "Продажи"

Выбираем пункт главного меню **Project\Option**, отменяем автоматическое создание формы. Для события `OnClick` кнопки `Button` с тремя точками,

расположенной напротив слова "Фирма", пишем код для создания формы FormFirm:

```
FormFirm:=TFormFirm.Create(self);
FormFirm.IBQuery1.Open;
{Делаем доступной кнопку "Выбрать"}
FormFirm.Button1.Enabled:=True;
FormFirm.ShowModal;
```

Выбираем форму FormFirm и для кнопки Button с текстом "Выбрать" пишем код:

```
procedure TFormFirm.Button1Click(Sender: TObject);
begin
 {Запоминаем название организации и его код, код
 пишем в свойство Tag}
 FormSale.Edit1.Text:=IBQuery1NAME.Value;
 FormSale.Edit1.Tag:=IBQuery1FIRMID.Value;
 FormFirm.Close;
end;
```

Выбираем форму FirmSale. Для кнопки с тремя точками, расположенной напротив слова "Товар", пишем код для создания формы FormTovar:

```
FormTovar:=TFormTovar.Create(self);
FormTovar.IBQuery1.Open;
{Делаем доступной кнопку "Выбрать"}
FormTovar.Button1.Enabled:=True;
FormTovar.ShowModal;
```

Выбираем форму FormTovar и для кнопки "Выбрать" пишем код:

```
procedure TFormTovar.Button2Click(Sender: TObject);
begin
 {Запоминаем название товара и его код, код
 пишем в свойство Tag}
 FormSale.Edit2.Text:=IBQuery1NAME.Value;
 FormSale.Edit2.Tag:=IBQuery1TOVARID.Value;
 FormTovar.Close;
end;
```

Переходим на форму FormSale, для кнопки "Закрыть" пишем код:

```
FormSale.Close;
```

Для кнопки "Продать" пишем код для продажи товара:

```
procedure TFormSale.Button3Click(Sender: TObject);
begin
 {Передаем в параметры данные по продаже}
 MainForm.IBQuery2.Params.ParamByName('FIRMKOD').Value:=Edit1.Tag;
 MainForm.IBQuery2.Params.ParamByName('TOVARKOD').Value:=Edit2.Tag;
 MainForm.IBQuery2.Params.ParamByName('REM').Value:=Edit3.Text;

 {пытаемся выполнить запрос}
 try
  MainForm.IBQuery2.ExecSQL;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Продать товар не получилось!'+#13+
               'Повторите попытку!');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем таблицу продаж, если она открыта}
 if MainForm.IBQuery1.Active then
 begin
   MainForm.IBQuery1.Close;
   MainForm.IBQuery1.Open;
 end;

 FormSale.Close;

end;
```

Пишем обработчик для кнопки "Закрыть", расположенной на форме `MainForm`:

```
MainForm.Close;
```

Теперь щелкаем два раза на компоненте `IBQuery1`, расположенном на главной форме, добавляем все атрибуты и оставляем видимыми только те, что

нужны нам: Rem, Name, Name1, Price. Можно менять их очередность, перетаскивая мышкой. Задаем им русский текст в свойстве DisplayLabel.

Все, проект готов. Осталось рассмотреть лишь несколько мелочей.

Перейдите на UDM (DataModule), щелкните правой кнопкой мыши по компоненту IBDatabase1 и выберите пункт **Database Editor**, появится окно **Database Component Editor** (рис. 3.52).

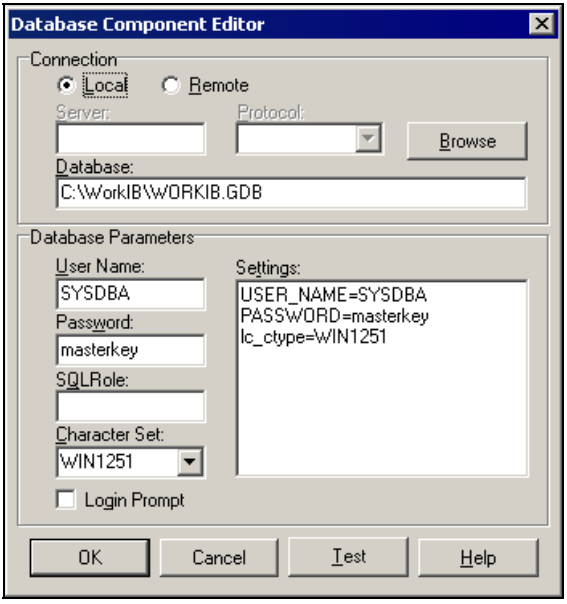


Рис. 3.52. Окно **Database Component Editor**

В выпадающем списке **Character Set** обязательно выберите WIN1251, без этого не будут нормально отображаться русские символы.

Для того чтобы подключиться с другого компьютера к нашей базе данных, надо установить на этот компьютер клиента InterBase (при установке InterBase выбирается соответствующий пункт). И написать в программе процедуру OpenBD.

Для этого сделайте активной форму MainForm, нажмите <F12> и перед разделом VAR объявите процедуру OpenBD:

```
{Процедура открытия базы}
procedure OpenBD;
var
  MainForm: TMainForm;
```

Для открытия базы с использованием пути из файла пишем код:

```
procedure OpenBD;
var
 FP:TextFile;
 StringPath:string;
 NameOfFile:string;
begin
  {получаем путь к файлу path.txt, там у нас будет
  храниться адрес БД}
  NameOfFile:=ExtractFileDir(Application.ExeName)+'\path.txt';

  {Если файл есть, то связываемся с ним, иначе
  сообщаем об ошибке и выходим из программы}
  if FileExists(NameOfFile) then
  begin
    Assign(Fp,NameOfFile);
    try
     begin
      Reset(FP);
      Read(FP,StringPath);
     end;
    finally
     CloseFile(FP);
    end;
  end
    else
     begin
      ShowMessage('Файл пути к БД не найден'+
                  #13+
                  'Создайте в папке программы файл path.txt'+
                  #13+
                  'и запишите туда путь к БД'+
                  #13+
                  'Пример: C:\WorkIB\WorkIB.GDB);
      Application.Terminate;
     end;

    {задаем базу данных для компонента IBDatabase}
    DM.IBDatabase1.DatabaseName:=StringPath;
```

```
    {Открываем базу данных и компонент для работы с
    транзакциями БД}
    with DM do
    begin
     IBDatabase1.Connected:=True;
     IBTransaction1.Active:=True;
    end;
end;
```

Теперь для модуля данных `DM` в событии `OnCreate` вызываем процедуру `OpenBD` (только необходимо будет подключить модуль `UMain`):

```
procedure TDM.DataModuleCreate(Sender: TObject);
begin
 OpenBD;
end;
```

В нашей рабочей папке (у меня это C:\WorkIB) создадим текстовый файл path.txt, можно с помощью Блокнота. В нем прописываем путь к базе, например, для случая, когда база лежит локально:

```
C:\WorkIB\WorkIB.gdb
```

Если необходимо подключаться к удаленной базе, то в файле path.txt необходимо сделать запись вида:

`Имя_сервера:имя_файла_БД` (имя сервера может быть символьным, либо можно ввести его IP-адрес).

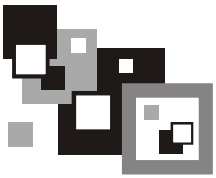
Еще один важный момент. Когда мы выполняем транзакции, мы никогда не пишем `IBTransaction1.StartTransaction`, мы либо подтверждаем их, либо откатываем. Это не ошибка, просто транзакции у нас стартуют автоматически, за это и отвечает компонент `IBTransaction`.

На компакт-диске в каталоге ГЛАВА 3\SOURCE можно увидеть пример данной программы.

В каталоге ГЛАВА 3\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 3\VIDEO можно посмотреть видеоролики, посвященные созданию разрабатываемого приложения.

Глава 4



Microsoft SQL Server 2000

4.1. Вступление

В этой главе рассмотрим написание программы Менеджер видеофильмов.

Представим, что у нас с друзьями по дому проведена локальная сеть, и у каждого есть больше десятка фильмов, которые, ну конечно, хранятся в формате AVI. Так вот, создадим программу, которая будет хранить все данные по этим фильмам. После этого нам останется просто запустить программу — она в свою очередь подключается к SQL Server — и вот, пожалуйста, список всех фильмов, их описаний и рейтинг. Только один будет недостаток — всю информацию придется забить сначала ручками. Ну, я думаю, он несущественный.

4.2. Установка

Замечание

Microsoft SQL Server 2000 Enterprise Edition предназначен только для серверов, на персональный компьютер данная версия не устанавливается. На домашний компьютер нужно устанавливать специальную версию Microsoft SQL Server Personal Edition.

В данной главе рассматривается установка версии Personal Edition. Установка Enterprise Edition практически ничем не отличается.

Запускаем файл AUTORUN.EXE, который находится в корне на диске с дистрибутивом Microsoft SQL Server. Появится окно приветствия (рис. 4.1).

Выбираем пункт **SQL Server 2000 Components**. Перед нами откроется еще одно окно (рис. 4.2).

Выбираем **Install Database Server**. Появляется окно приветствия **Welcome** мастера установки. Нажимаем **Next**. Появляется окно **Computer Name** для выбора типа сервера — локальный или удаленный (рис. 4.3).

Рис. 4.1. Окно приветствия Microsoft SQL Server

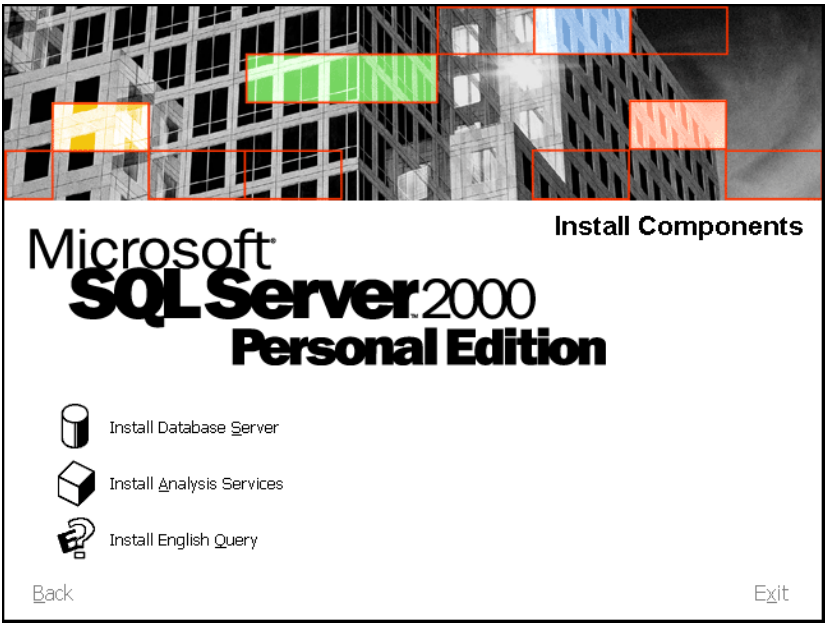


Рис. 4.2. Окно **Install Components**

Рис. 4.3. Окно **Computer Name** для выбора типа сервера

В этом окне доступны два переключателя:

- **Local Computer** — установка MS SQL Server на локальный компьютер (рекомендую выбрать данный пункт, если вы, конечно, не обладатель сервера). Поле с именем сервера (на рис. 4.3 — имя IT05) заполняется автоматически инсталлятором и является таким же, как и имя компьютера.
- **Remote Computer** — установка СУБД на удаленный сервер.

Нажимаем **Next**, появляется окно **Installation Selection** (рис. 4.4).

В нем необходимо выбрать параметры установки:

- **Create a new instance of SQL Server, or install Client Tools** — установить экземпляр сервера или его клиентскую часть;
- **Upgrade, remove, or add components to an existing instance of SQL Server** — изменение, удаление или добавление компонентов установленного экземпляра SQL Server;
- **Advanced options** — дополнительные возможности.

Выбираем первый пункт. Нажимаем **Next**. Появляется окно **User Information** (рис. 4.5), где представлены ваши персональные данные.

Рис. 4.4. Окно **Installation Selection**

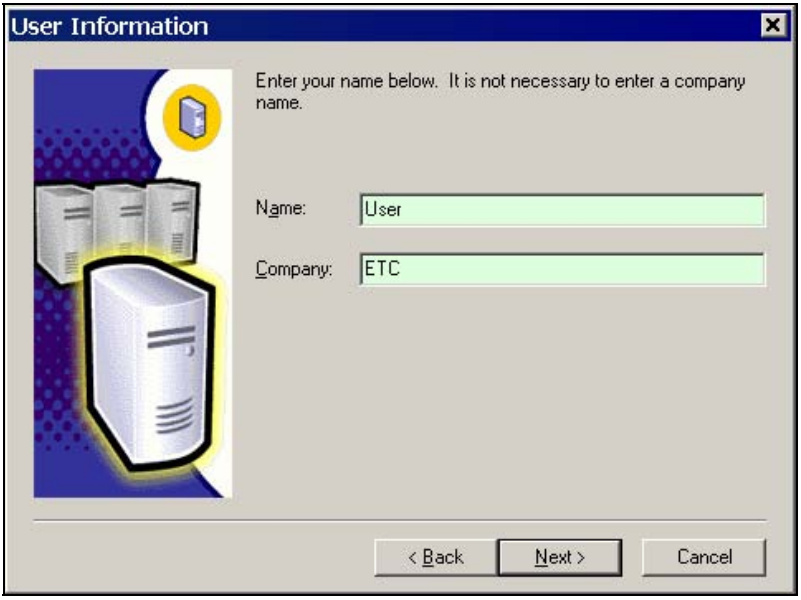


Рис. 4.5. Окно **User Information**

Нажимаем **Next**. Появляется окно **Software License Agreement**, которое содержит лицензионное соглашение. Нажимаем **Yes**. Вводим серийный номер и нажимаем **Next**. Появляется окно **Installation Definition** для выбора того, что именно будет устанавливаться (рис. 4.6).

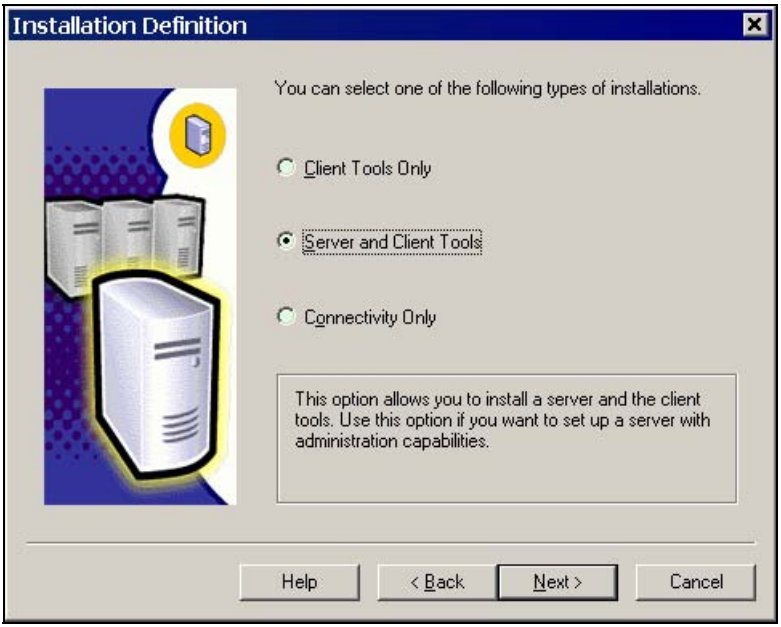


Рис. 4.6. Окно **Installation Definition** для выбора типа установки

Доступные переключатели:

- **Client Tools Only** — установка клиентской части (позволяет подключаться к удаленным серверам и работать с базами данных);
- **Server and Client Tools** — установка серверной и клиентской части (это как раз то, что нам нужно);
- **Connectivity Only** — установка только драйвера для соединения с базой.

Выбираем второй пункт и нажимаем **Next**.

Следующее окно — **Instance Name** — позволяет задать имя экземпляра SQL-сервера. Если сбросить флаг **Default**, то можно будет ввести свое имя. Если оставить флаг установленным, то имя будет такое же, как имя компьютера. Если вы введете свое имя, то имя экземпляра будет выглядеть как **Имя компьютера/Введенное имя**. Нажимаем **Next**. Появляется окно **Setup Type**, здесь выбирается тип установки (рис. 4.7).

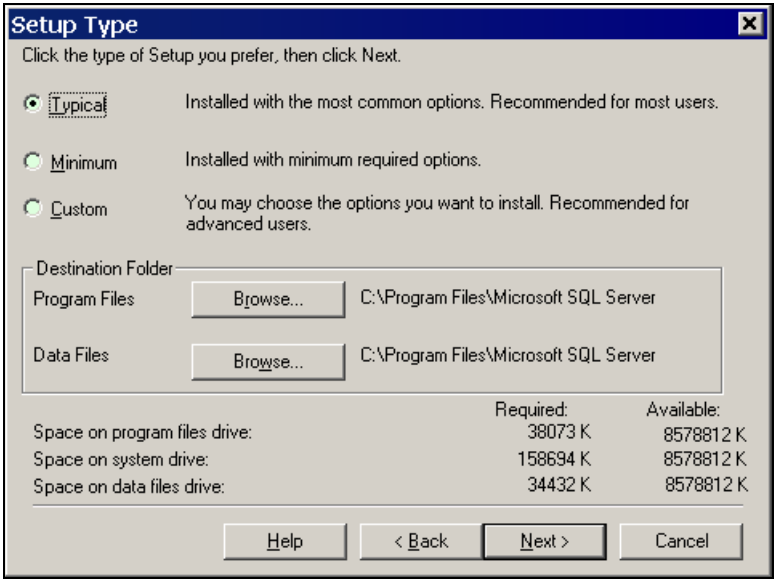


Рис. 4.7. Окно для выбора типа установки

Доступны следующие переключатели:

- **Typical** — стандартная установка;
- **Minimum** — минимальная;
- **Custom** — выборочная.

Оставляем **Typical** и нажимаем **Next**. Появляется окно настройки параметров запуска сервисов (рис. 4.8).

Доступны следующие переключатели:

- **Use the some account for each service. Auto start SQL Server Service** — сервисы будут стартовать с одинаковыми параметрами и автоматически при загрузке Windows;
- **Customize the settings for each service** — определить для каждого сервиса свои настройки, при выборе группа элементов **Services** становится доступной и можно настраивать параметры запуска отдельно для каждого сервиса.

Группа элементов **Service Settings** предназначена для настройки параметров:

- Переключатель **Use the Local System account** — использовать локальную учетную запись;
- Переключатель **Use a Domain User account** — использовать учетную запись домена, дополнительно необходимо будет ввести логин и пароль.

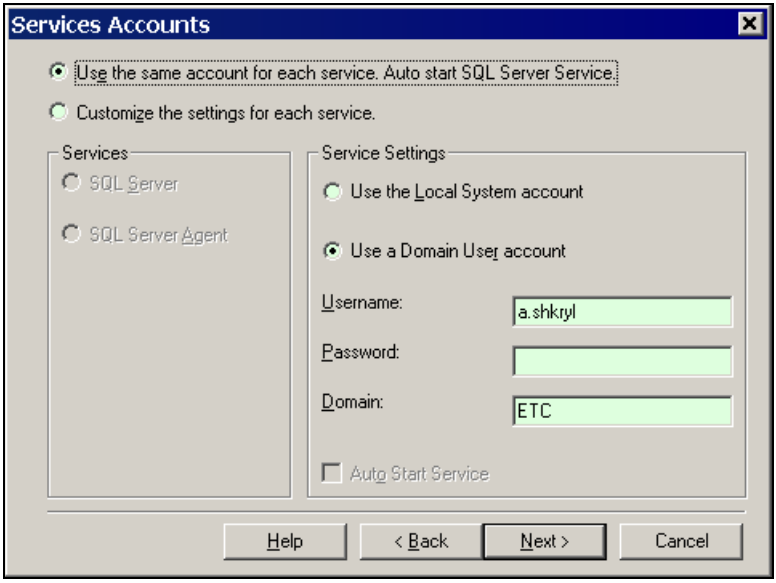


Рис. 4.8. Окно настройки параметров запуска сервисов

Если был выбран переключатель **Customize the settings for each service** группы элементов **Services**, тогда станет доступен флаг **Auto Start Service**, позволяющий запускать сервисы автоматически после загрузки Windows.

Выбираем переключатели **Use the some account for each service** и **Use the Local System account**. Нажимаем **Next**. Появляется окно **Authentication Mode**, где выбирается режим аутентификации для экземпляра сервера (рис. 4.9).

Доступны следующие переключатели:

- **Windows Authentication Mode** — использовать авторизацию Windows (когда будет происходить соединение с SQL Server, то в качестве логина и пароля будут передаваться те, с какими произошел вход в Windows);
- **Mixed Mode (Windows Authentication and SQL Server Authentication)** — смешанная авторизация (вход может быть осуществлен как с помощью учетной записи Windows, так и с помощью независимой учетной записи, созданной в SQL Server).

Выбираем второй вариант, при этом нам надо задать пароль для учетной записи администратора — sa.

Замечание

Учетная запись sa обладает правами суперпользователя и позволяет человеку, зашедшему под ней, выполнять абсолютно любые операции с экземпляром СУБД.

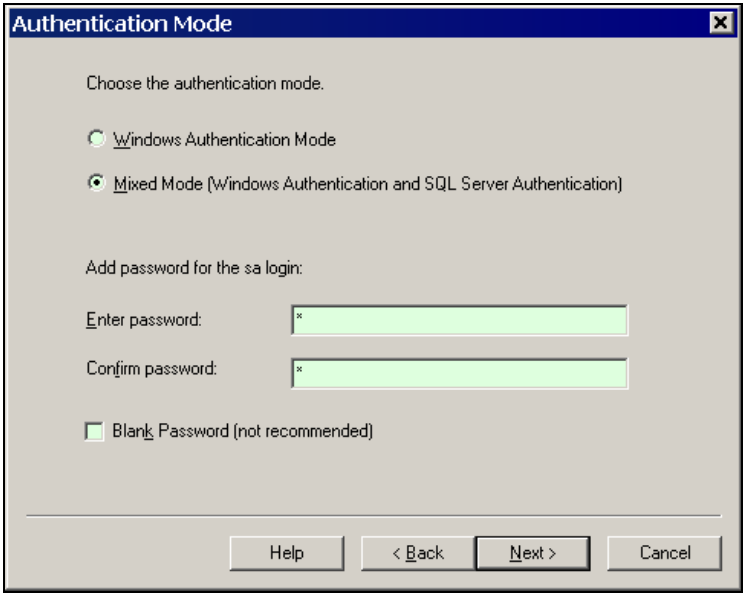


Рис. 4.9. Окно **Authentication Mode**

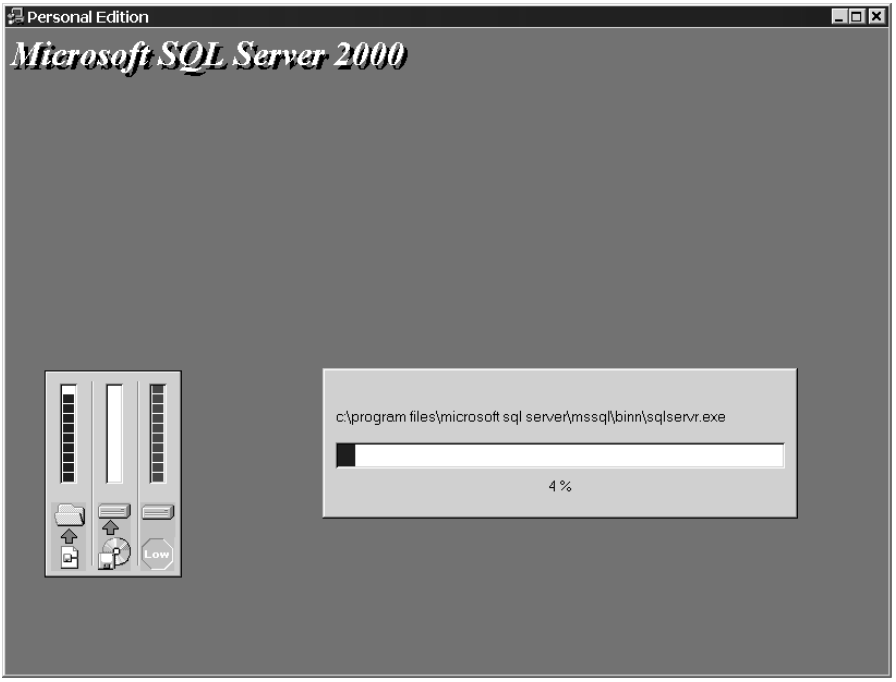


Рис. 4.10. Процесс установки MS SQL Server

Нажимаем **Next**, появляется окно **Start Copying Files**, предупреждающее о том, что сейчас начнется установка. Нажимаем **Next**. После этого начнется процесс установки.

После того как установка закончится, необходимо будет нажать кнопку **Finish**.

4.3. Постановка задачи

Сразу разработаем структуру нашей будущей базы данных, она будет состоять из таблиц, представленных в табл. 4.1—4.4.

Таблица 4.1. *Таблица "Информация по фильмам" (Main)*

Имя атрибута	Описание	Тип данных
FilmID	Поле-счетчик	Счетчик
NameOfFilm	Название фильма	Текст(100)
GanrKod	Код жанра фильма, подставляется из таблицы "Жанры"	Число
FilmAbout	Краткое содержание фильма	Текст(3000)
PathToFilm	Путь к фильму	Текст(1000)

Таблица 4.2. *Таблица "Рейтинг" (Reiting)*

Название	Описание	Тип данных
StatusID	Поле-счетчик	Счетчик
FilmKod	Код фильма, к которому относится данная оценка	Число
UserKod	Код пользователя из таблицы "Безопасность"	Число
Ball	Оценка, поставленная пользователем фильму	Число

Таблица 4.3. *Таблица "Безопасность" (SecurityTable)*

Название	Описание	Тип данных
UserID	Поле-счетчик	Счетчик
UserLogin	Имя входа пользователя	Текст(50)
UserPassword	Пароль пользователя	Текст(50)

Таблица 4.4. *Таблица "Жанры" (Ganri)*

Название	Описание	Тип данных
GanrID	Поле-счетчик	Счетчик
NameOfGanr	Название жанра	Текст(50)

Теперь необходимо разобраться в данной структуре. При входе в программу пользователю надо пройти аутентификацию: ввести имя и пароль. Которые будут храниться в таблице "Безопасность", поэтому наша программа будет состоять из двух модулей:

- модуль для работы с базой данных;
- модуль аутентификации, где мы можем добавлять, удалять и изменять данные по пользователям.

При входе в программу пользователь получает доступ к таблице "Информация по фильмам". В этой таблице не хранятся названия жанров фильмов, а только их коды из таблицы "Жанры". Между таблицами "Информация по фильмам" и "Жанры" существует связь многие-к-одному, то есть один жанр может быть присвоен нескольким фильмам.

Также у каждого фильма есть рейтинг, который формируют пользователи. Оценки пользователей, а также информация о том, какой пользователь и какому фильму поставил оценку, хранится в таблице "Рейтинг". Таким образом, посредством таблицы "Рейтинг" осуществляется связь между таблицами "Информация по фильмам" и "Безопасность" (одному фильму оценки могут поставить множество пользователей, соответственно, у множества фильмов может быть оценка, поставленная одним и тем же пользователем).

Замечание

Описание структуры базы и взаимодействия между таблицами очень важно понимать, так как это необходимо, чтобы правильно реализовать базу для конкретной СУБД и написать под нее приложение. Если же понимание всех процессов, которые будут проходить в базе данных, не полное, то повышается вероятность многократного переделывания разрабатываемой задачи.

4.4. Создание базы данных в MS SQL Server

Запускаем MS SQL Server и нажимаем на плюсе, расположенном слева от названия вашего сервера (у меня IT05), таким образом мы подключимся к выбранному серверу. После этого количество доступных элементов сразу увеличится (рис. 4.11).

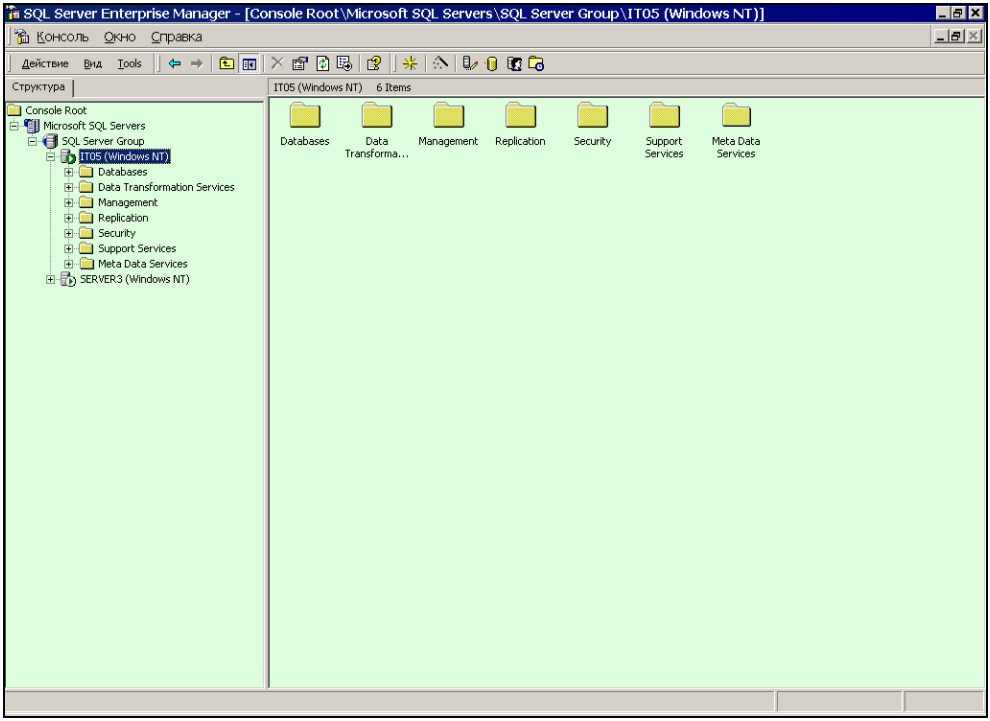


Рис. 4.11. Подключение к серверу

Щелкаем правой кнопкой мыши на пункте **Database** и выбираем в выпадающем меню **New Database** (рис. 4.12). Тем самым мы приступим к созданию новой базы данных.

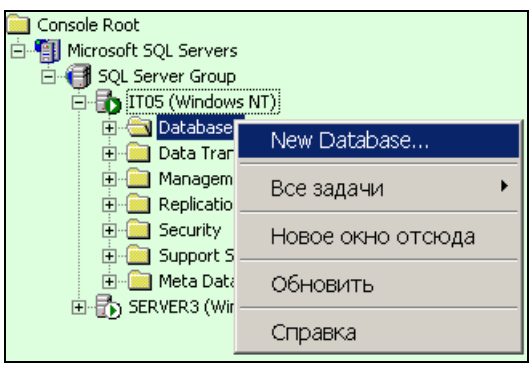


Рис. 4.12. Создание новой базы данных

Появиться окно **Database Properties**. В поле **Name** вводим **FilmManager** — это название будущей базы. Остальные параметры оставляем по умолчанию (рис. 4.13).

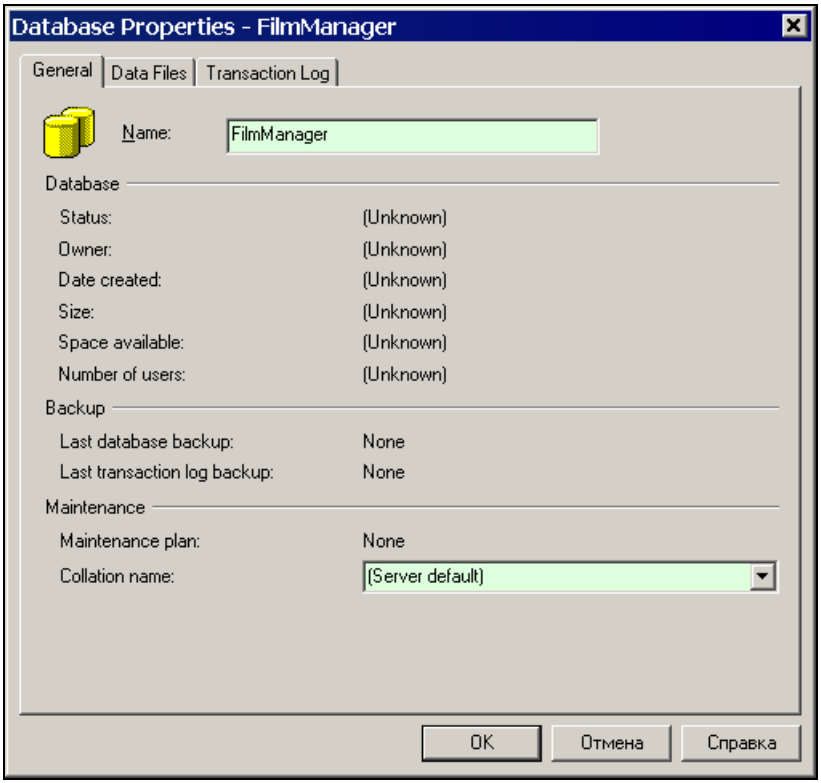


Рис. 4.13. Окно **Database Properties**

Нажимаем **OK**. После этого база будет создана. Теперь нажимаем на плюс, расположенный рядом с пиктограммой **Database**, далее на плюс рядом с **FilmManager**. Выбираем **Tables**, тем самым получая доступ ко всем таблицам выбранной базы данных (рис. 4.14).

Несмотря на то, что база была только что создана, в ней присутствуют уже несколько таблиц, которые являются системными. В них содержится информация, касающаяся схемы базы данных. Например, системная таблица `sysobject` содержит имена системных объектов: таблиц, хранимых процедур и т. д.; системная таблица `syscolumns` содержит размеры и типы столбцов внутри таблиц.

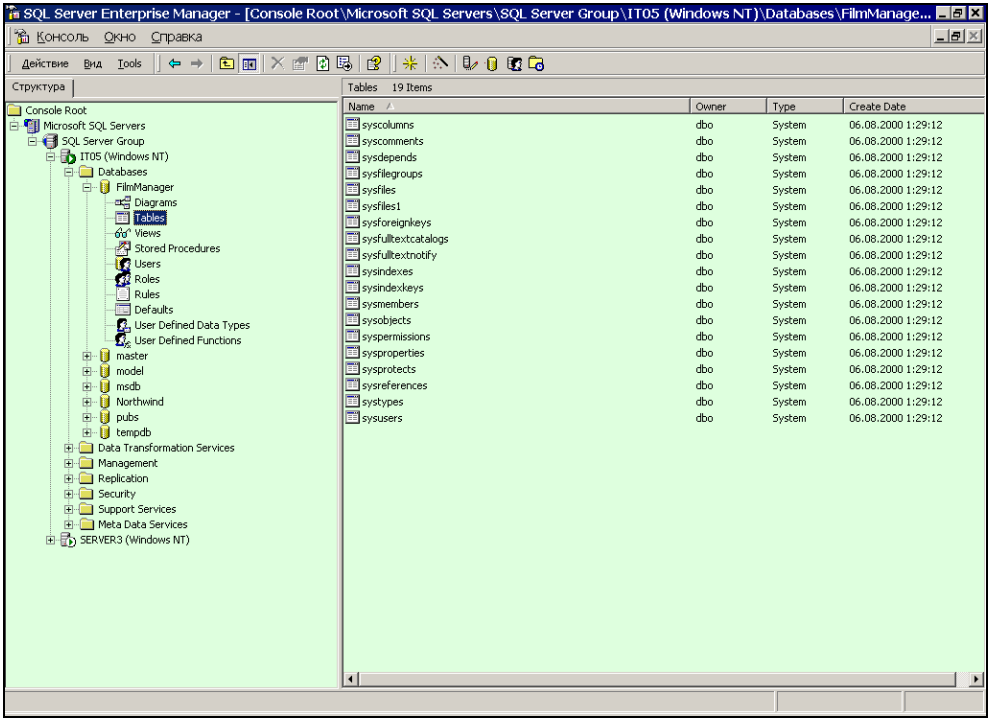


Рис. 4.14. Просмотр таблиц выбранной базы данных

Замечание

Если, открыв папку **Tables**, вы не видите системных таблиц, то нужно щелкнуть правой кнопкой мыши на названии вашего сервера (у меня IT05) и выбрать команду **Edit SQL Server Registration properties**. Появится диалоговое окно свойств регистрации **SQL Server**. Убедитесь, что флаг **Show system databases and system object** установлен.

Идем далее. Нажимаем правой кнопкой мыши на **Tables** и в выпадающем меню выбираем **New Table**. Появляется окно, в котором мы будем создавать свою первую таблицу.

Окно состоит из четырех частей:

- **Column Name** — название атрибута таблицы;
- **Data Type** — тип данных, можно выбирать из выпадающего списка;
- **Length** — длина атрибута (для некоторых типов атрибутов ставится автоматически);

- **Allow Nulls** — может ли атрибут содержать значение `Null` (многие советуют избегать ставить это свойство в `False`, хотя я в этом ничего зазорного не вижу).

Необходимо ввести название атрибутов и установить их свойства в соответствии с рис. 4.15.

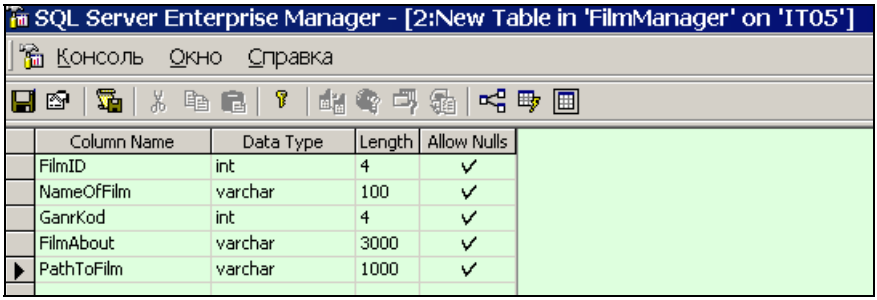


Рис. 4.15. Таблица `Main` (содержит данные по фильмам)

Сейчас необходимо сделать атрибут `FilmID` ключевым. Делается это нажатием на кнопку с изображением ключа, при этом необходимо, чтобы курсор был именно на том атрибуте, который мы хотим сделать ключевым. Теперь надо сделать из этого атрибута счетчик, чтобы значения автоматически увеличивались на единицу при создании новой записи. Для этого два раза щелкаем на свойстве `Identity`, чтобы его значение изменилось на `True` (по умолчанию оно установлено в `False`) (рис. 4.16).

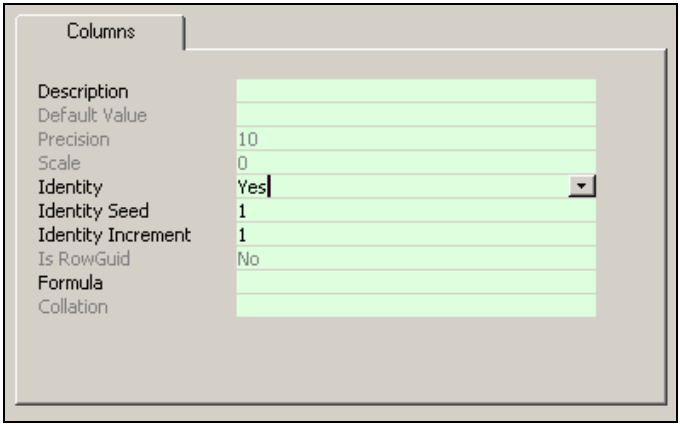


Рис. 4.16. Создание атрибута счетчика

Нажимаем на кнопку сохранения (кнопка с изображением дискеты). Появится окно **Choose Name** для ввода имени таблицы. Вводим название таблицы — Main и нажимаем **OK** (рис. 4.17). Первая таблица создана.

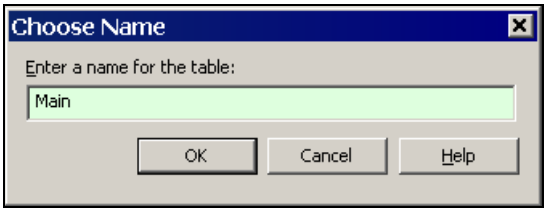


Рис. 4.17. Окно **Choose Name** для ввода имени таблицы

Закрываем окно дизайнера таблицы. Создаем новую таблицу. Вводим название атрибутов и устанавливаем их свойства в соответствии с рис. 4.18, атрибут `StatusID` делаем счетчиком и ключевым. Сохраняем таблицу под именем Reiting.

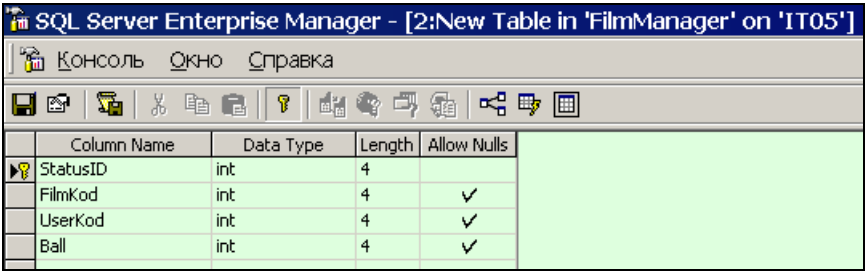


Рис. 4.18. Таблица Reiting (содержит рейтинг фильмов)

Создаем таблицу "Безопасность" в соответствии с рис. 4.19, атрибут `UserID` делаем счетчиком и ключевым. Сохраняем таблицу под именем SecurityTable.

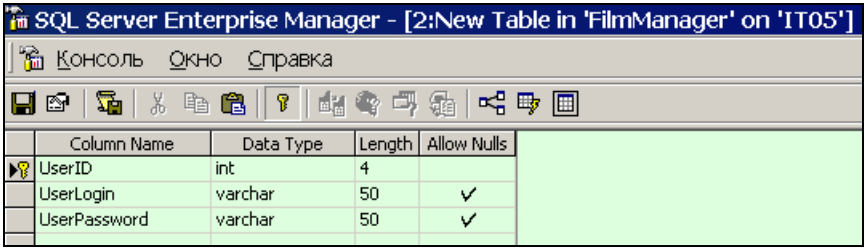


Рис. 4.19. Таблица Security (содержит имена и пароли пользователей)

Создаем таблицу "Жанры" в соответствии с рис. 4.20, атрибут `GanrID` делаем счетчиком и ключевым. Сохраняем таблицу под именем Ganri.

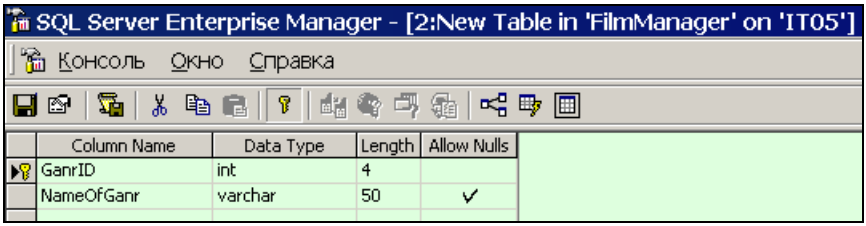


Рис. 4.20. Таблица Ganri (содержит жанры фильмов)

Замечание

Если необходимо внести изменения в структуру таблицы, после того, как она создана, необходимо щелкнуть правой кнопкой мыши на имени таблицы и выбрать в выпадающем меню пункт **Design Table**.

4.5. Целостность данных

Под целостностью данных подразумевается установка правил для элементов базы данных, в том числе и для атрибутов связанных таблиц.

Например, если в таблице "Рейтинг" будет выставлена оценка неким пользователем под кодом 999, а в таблице "Безопасность" не будет пользователя с таким кодом, то это будет указывать на то, что в базе отсутствуют правила поддержания целостности данных и, как следствие этого, информация противоречивая и, следовательно, ценность ее не высока.

Мы будем работать с целостностью на уровне ссылок (**referential integrity**) — этот уровень устанавливает правило (или правила), в соответствии с которым вносимое в атрибут значение должно существовать в связанной таблице. Этот тип целостности реализуется посредством использования внешних ключей и называется ссылочной целостностью.

Щелкните правой кнопкой мыши по таблице Main и выберите из выпадающего меню команду **Design Table** (рис. 4.21).

В открывшемся окне необходимо нажать на кнопке **Manage Relationships** (третья кнопка справа в панели инструментов дизайнера таблицы). Появится окно **Properties** (рис. 4.22).

В данном окне мы определим ссылочную целостность для таблиц Main и Ganri, данные таблицы связаны отношением многие-к-одному, то есть од-

ной записи в таблице Ganri может соответствовать несколько записей из таблицы Main (один жанр может соответствовать нескольким фильмам).

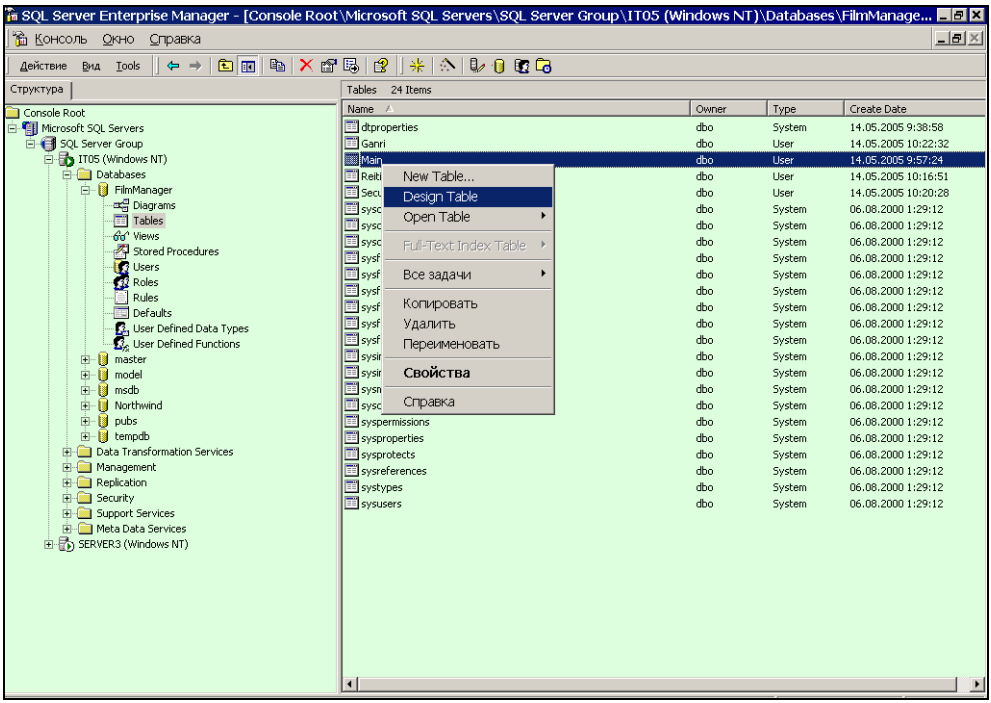


Рис. 4.21. Выбор пункта **Design Table** выпадающего меню

Нажимаем кнопку **New**. MS SQL Server автоматически установит имя для создаваемой ссылочной целостности FK_Main_Ganri. Нам нужно привести окно **Properties** к виду, как на рис. 4.23.

То есть в роли главной таблицы (**Primary key table**) у нас выступает таблица Ganri, а в роли второстепенной (**Foreign key table**) таблица Main. Так же мы указываем атрибуты этих таблиц, по которым осуществляется связь.

Теперь поговорим о флагах, которые расположены в нижней части окна.

- **Check existing data on creation** — проверять существование данных при создании отношения. Установка этого флага позволяет создавать отношение между таблицами на основе существования данных в таблице с внешним ключом. Поскольку мы имеем дело с только что созданными таблицами, не заполненными пока никакой информацией, сброс/установка данного флага никоим образом не повлияет на создание отношения между таблицами. Тем не менее оставим его в установленном положении.

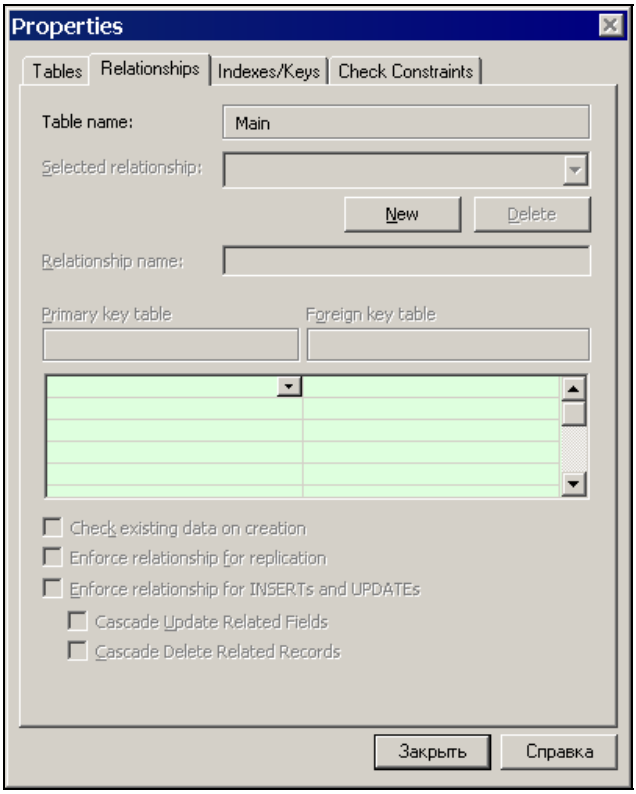


Рис. 4.22. Окно для настройки ссылочной целостности

- **Enforce relationship for replication** — сохранять отношение при репликации, гарантирует сохранение целостности данных на уровне ссылок при репликации (переносе) таблицы в другую базу данных.
- **Enforce relationship for INSERTs and UPDATEs** — сохранять отношение между таблицами при вставке и обновлении информации.
  - **Cascade Update Related Fields** — каскадное обновление связанных полей. Указывает SQL Server на необходимость обновлять значение внешнего ключа каждый раз, когда изменяется соответствующее ему значение первичного ключа. То есть если вдруг случайно изменить код пользователя в таблице SecurityTable, то, соответственно, в таблице Reiting код этого пользователя для тех записей, где он фигурирует, автоматически изменится.
  - **Cascade Delete Related Records** — каскадное удаление связанных полей. Указывает SQL Server на необходимость удаления всех записей в таблице с внешним ключом при удалении соответствующей записи из

таблицы с первичным ключом. Установка этого флага гарантирует, что если мы удалим пользователя из таблицы SecurityTable, то удалятся и все оценки, которые он ставил в таблице Reiting.

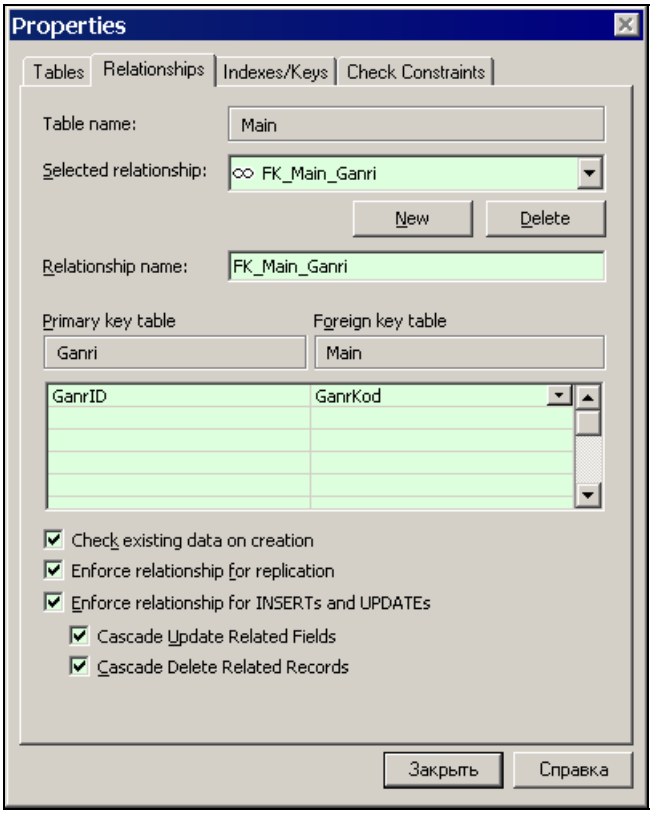


Рис. 4.23. Окно **Properties**
с настроенными параметрами ссылочной целостности

Нажимаем кнопку **Закрыть** в окне **Properties** и нажимаем кнопку сохранения (кнопка с дискетой). Появится окно **Save**, в котором SQL Server спросит: "Хотите сохранить изменения для таблиц Ganri, Main", нажимаем **Yes**. После этого произойдет сохранение изменений.

Закрываем режим дизайнера таблицы. Теперь повторим аналогичные действия для оставшихся таблиц. Производим щелчок правой кнопкой мыши по таблице Reiting и выбираем из выпадающего меню **Design Table** (рис. 4.25).

В открывшемся окне щелкните на кнопке **Manage Relationships**. Появится окно **Properties** для создания ссылочной целостности. Нажимаем на кнопку **New** и приводим окно к виду как на рис. 4.26.

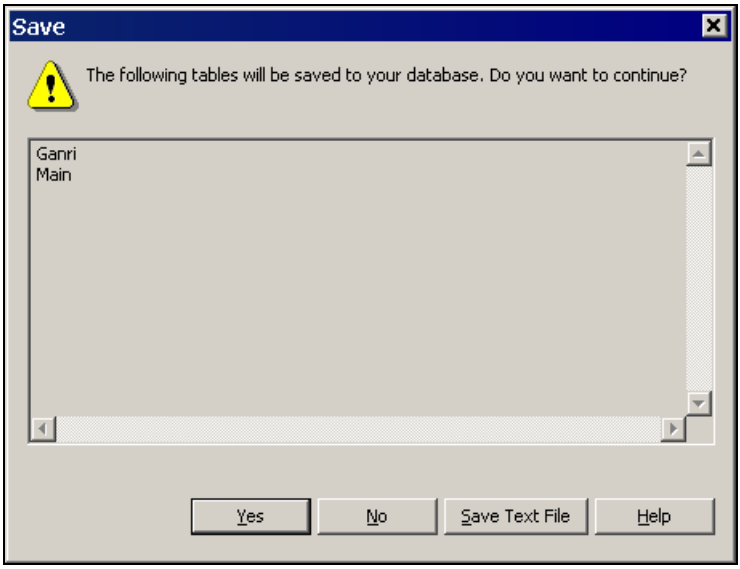


Рис. 4.24. Окно сохранения изменений

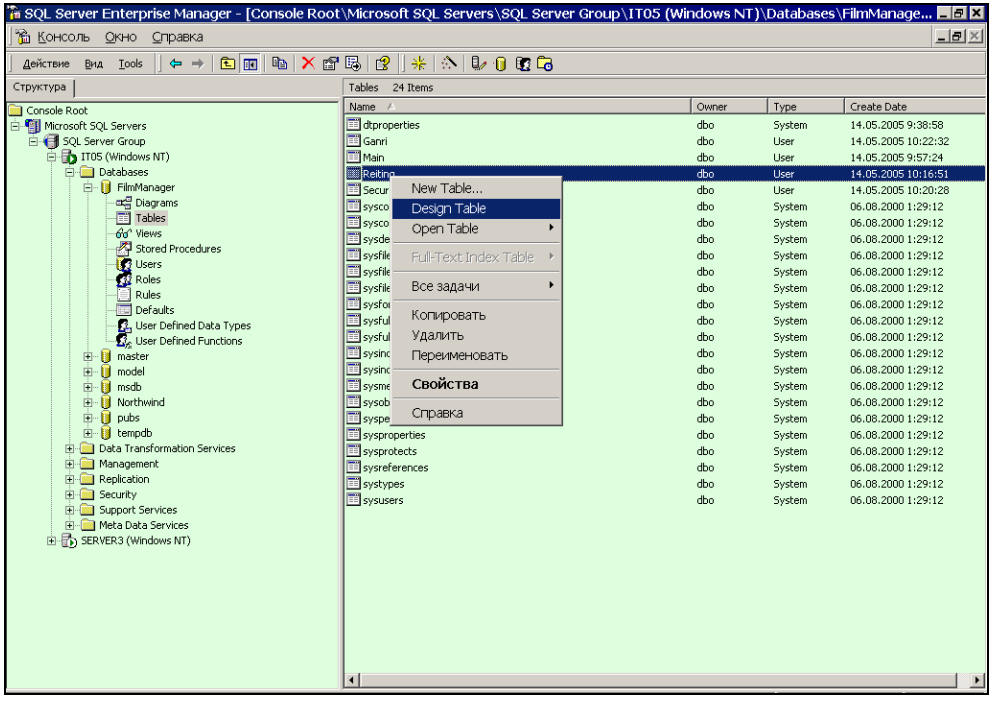


Рис. 4.25. Выбор пункта **Design Table** выпадающего меню

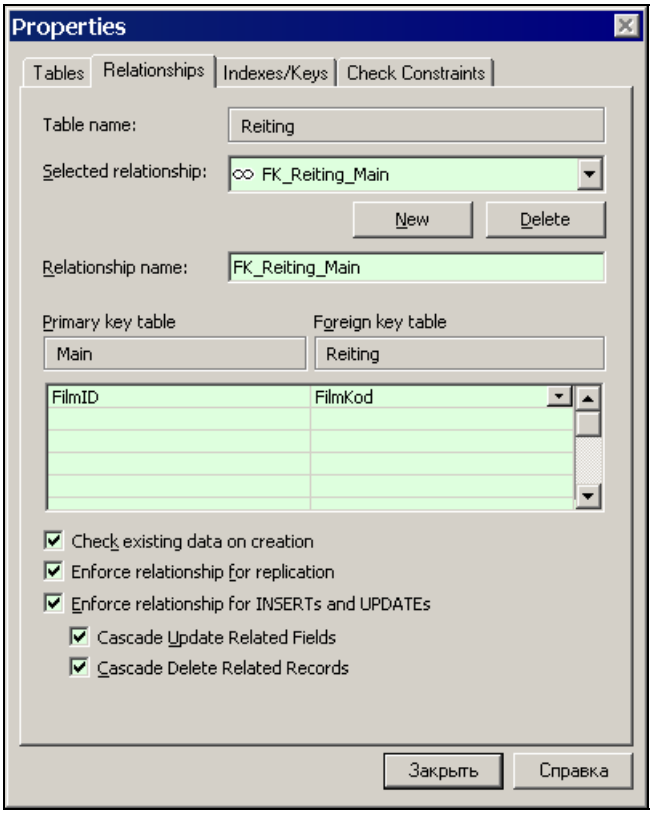


Рис. 4.26. Окно **Properties** с настроенными параметрами ссылочной целостности

Таким образом, мы указываем, что записи в таблице Reiting могут существовать только для существующих фильмов в таблице Main.

Нажимаем еще раз кнопку **New** в этом же окне. Это нужно, потому что для одной таблицы у нас будет два правила, так как она связана сразу с двумя таблицами: SecurityTable и Main. После нажатия на кнопке **New** вам нужно заполнить окно **Properties** как на рис. 4.27.

Нажимаем кнопку **Закрыть**, далее нажимаем кнопку сохранения (пиктограмма с дискетой). Появляется окно **Save**, где говорится уже о трех затронутых таблицах (рис. 4.28).

Нажимаем **Yes** и закрываем окно дизайнера таблицы. Самое сложное позади, теперь будет все проще и быстрее.

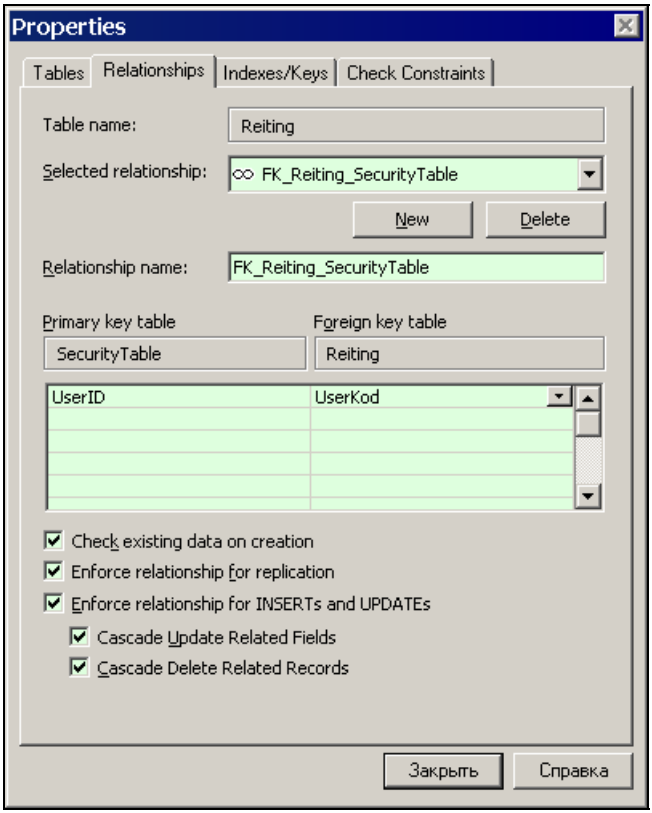


Рис. 4.27. Окно **Properties** с настроенными параметрами ссылочной целостности

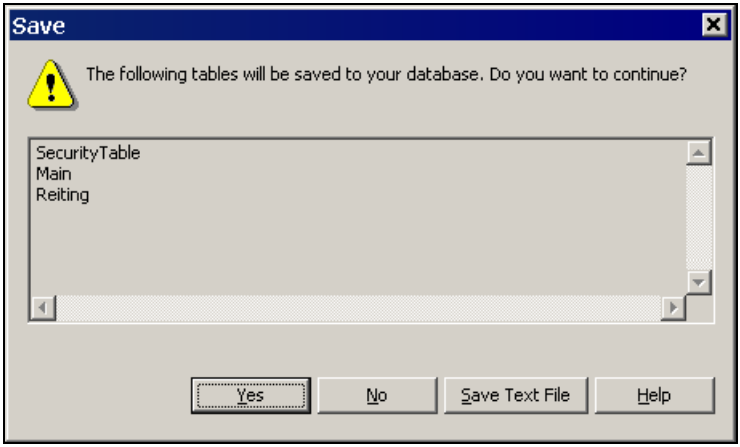


Рис. 4.28. Окно сохранения изменений

4.6. Визуализация структуры базы данных

Щелкаем правой кнопкой мыши на пиктограмме с текстом **Diagrams** (левая часть окна) и в выпадающем меню выбираем пункт **New Database Diagram** (рис. 4.29).

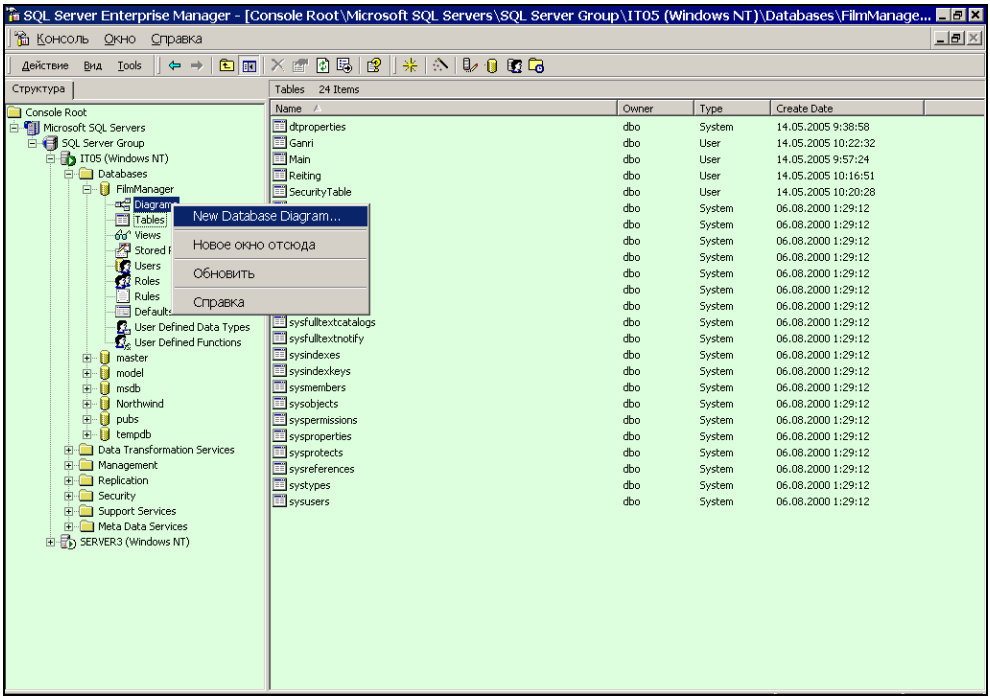


Рис. 4.29. Выбор пункта **New Database Diagram** выпадающего меню

После этого появится мастер создания диаграмм (**Create Database Diagram Wizard**) (рис. 4.30).

В нем говорится, что с помощью этого замечательного мастера вы сможете создать диаграммы, или как это называется в одной книжке по SQL Server — "воплощение диаграммы отношений" (романтично, не правда ли?). Нажимаем **Далее**. Появится следующее окно **Select Tables to be Added** (рис. 4.31).

Здесь все очень просто. Доступны два списка:

- **Available tables** — это доступные таблицы в вашей базе данных, тут их много, потому что автоматически сюда попадают и системные таблицы;

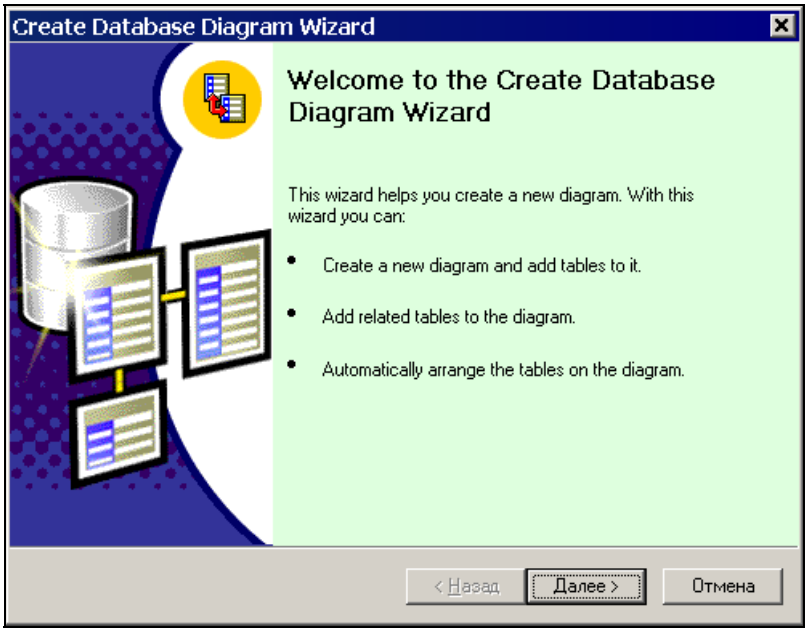


Рис. 4.30. Окно **Create Database Diagram Wizard**

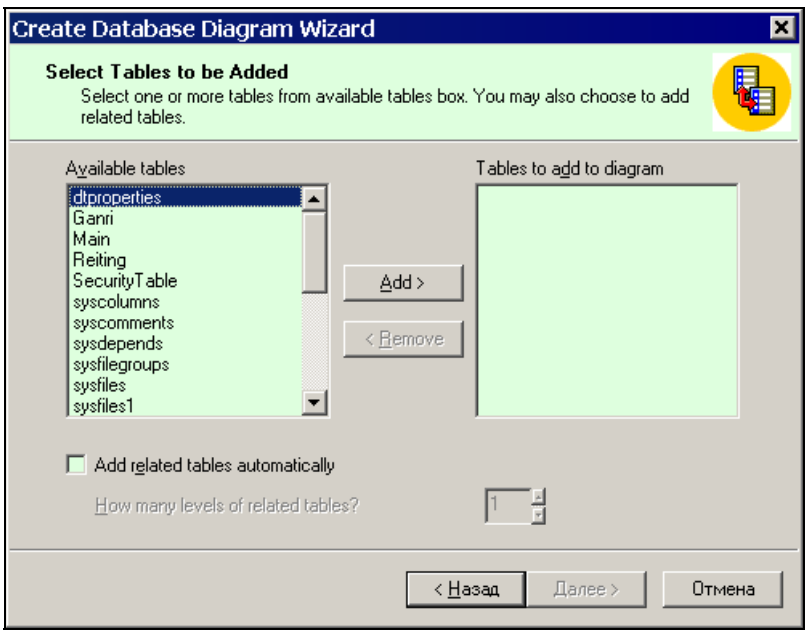


Рис. 4.31. Шаг **Select Tables to be Added** мастера создания диаграмм

- **Tables to add to diagram** — это те таблицы, которые будут отображаться на диаграмме.

Между списками расположены две кнопки: **Add** и **Remove**. Выбираем в списке **Available tables** таблицы, нажимаем **Add** и добавляем их в список **Tables to add to diagram**. Если мы ошиблись, то выбираем добавленную таблицу в списке **Tables to add to diagram** и нажимаем **Remove**, таким образом удаляя таблицу из диаграммы.

Но это простой способ и не совсем удобный. Представьте, что у вас 100 таблиц и, если вы будете добавлять их в диаграмму по одной, то этот процесс может занять большое количество времени.

Устанавливаем флаг **Add related tables automatically**. Таким образом, при добавлении одной таблицы вместе с ней будут добавляться и все связанные с ней.

Выбираем таблицу Main, нажимаем кнопку **Add**. В результате в списке **Tables to add to diagram** появятся три таблицы: Ganri, Main, Reiting. Добавляем теперь таблицу SecurityTable (рис. 4.32).

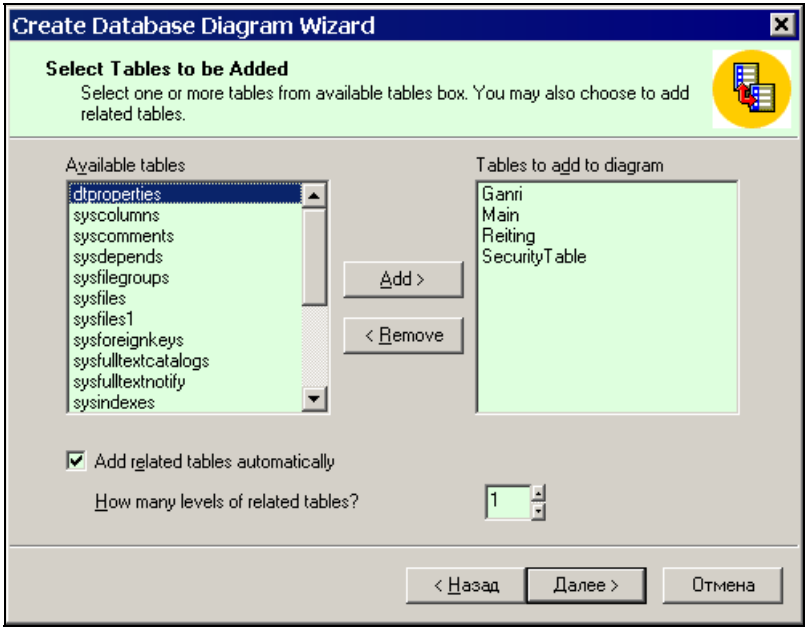


Рис. 4.32. Выбор таблицы на шаге **Select Tables to be Added** мастера создания диаграмм

Нажимаем кнопку **Далее**. Появится следующее окно (рис. 4.33).

Рис. 4.33. В этом окне отображаются выбранные таблицы

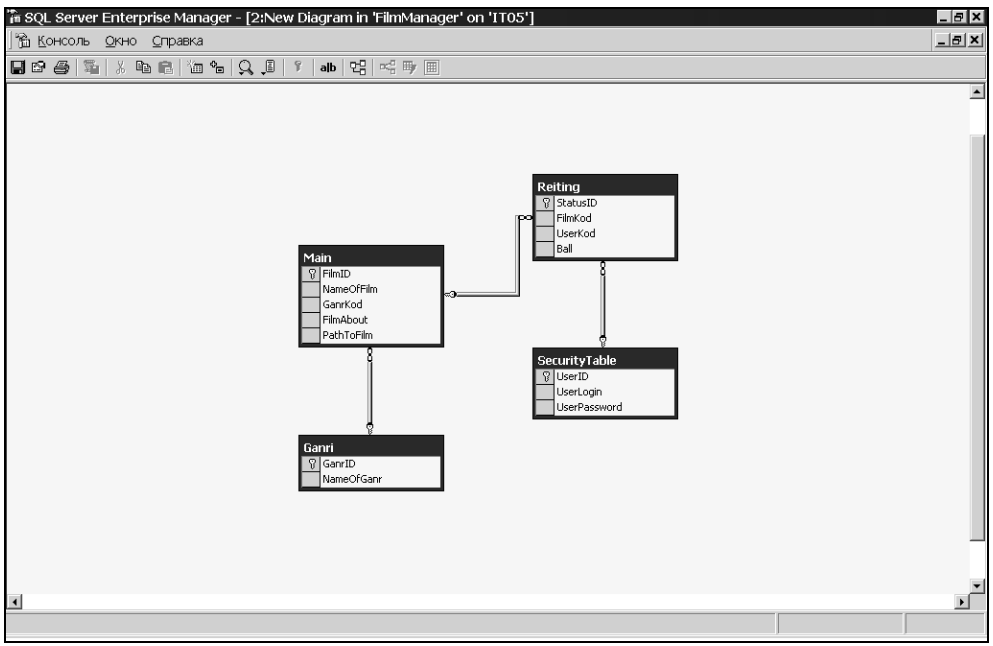


Рис. 4.34. Диаграмма базы данных

Здесь отображаются выбранные таблицы. Мы проверяем и нажимаем **Готово**. Немного подождав, можно увидеть картинку наподобие той, которая изображена на рис. 4.34.

Таблицы можно перемещать, менять их размеры. Щелкнув правой кнопкой на заголовке изображения таблицы, можно изменить режим отображения. Также диаграмму можно распечатать, так как очень удобно иметь такую диаграмму под рукой, не надо заглядывать по многу раз в SQL Server, не надо вспоминать названия атрибутов. Также с помощью меню, вызываемого правой кнопкой мыши, можно добавлять новые таблицы, удалять ненужные и т. д.

Перед закрытием окна с диаграммой ее надо сохранить.

В результате нехитрых манипуляций с мышкой и клавиатурой у меня получился следующий вариант диаграммы, показанный на рис. 4.35.

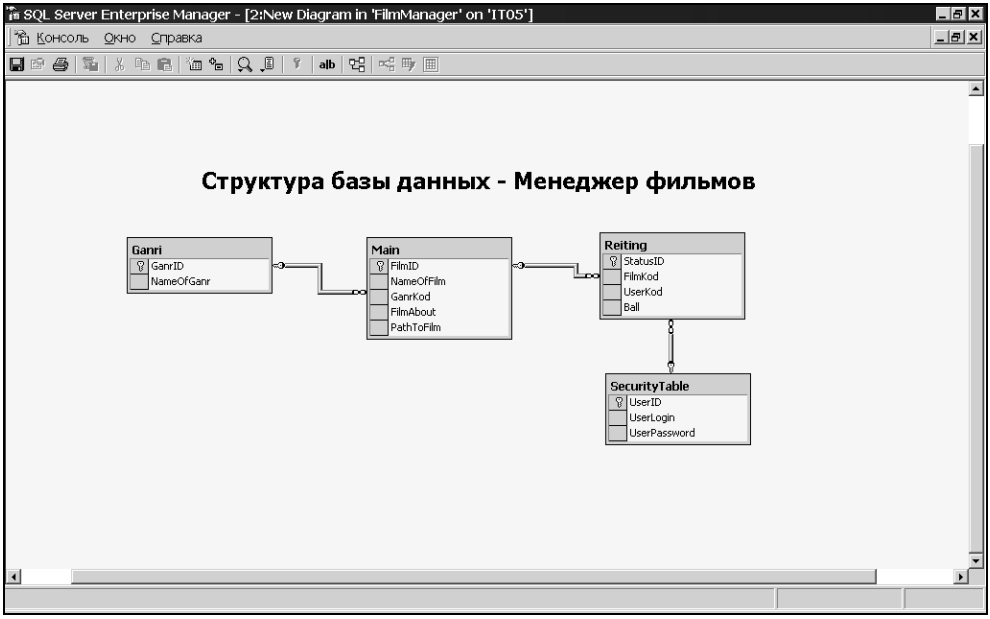


Рис. 4.35. Моя диаграмма базы данных

4.7. Работа с хранимыми процедурами

Теперь поговорим о хранимых процедурах. Это объект SQL Server, представленный набором откомпилированных операторов SQL. Мы создадим 3 хранимые процедуры для таблицы Main: на добавление, редактирование и изменение данных.

Замечание

Может возникнуть вопрос: зачем это нужно, если все это можно реализовать на клиентских машинах с помощью тех же самых операторов SQL: `INSERT`, `UPDATE`, `DELETE`. Вся фишка в том, что когда процедура хранится на сервере, она выполняется быстрее, так как SQL Server ее оптимизирует и клиентскому приложению нужно только послать запрос на выполнение определенной процедуры, ему не надо посылать сам код, из которых состоит процедура. Также такую базу данных удобней сопровождать, так как если потребуются изменения в процедурах, не потребуется доставать исходные коды приложения и компилировать его заново после внесенных изменений. Достаточно будет подключиться к серверу и оперативно внести исправления, при этом в клиентском приложении изменений не потребуется.

Щелкаем правой кнопкой мыши на пиктограмме с текстом **Stored Procedure** и в выпадающем меню выбираем пункт **New Stored Procedure** (рис. 4.36).

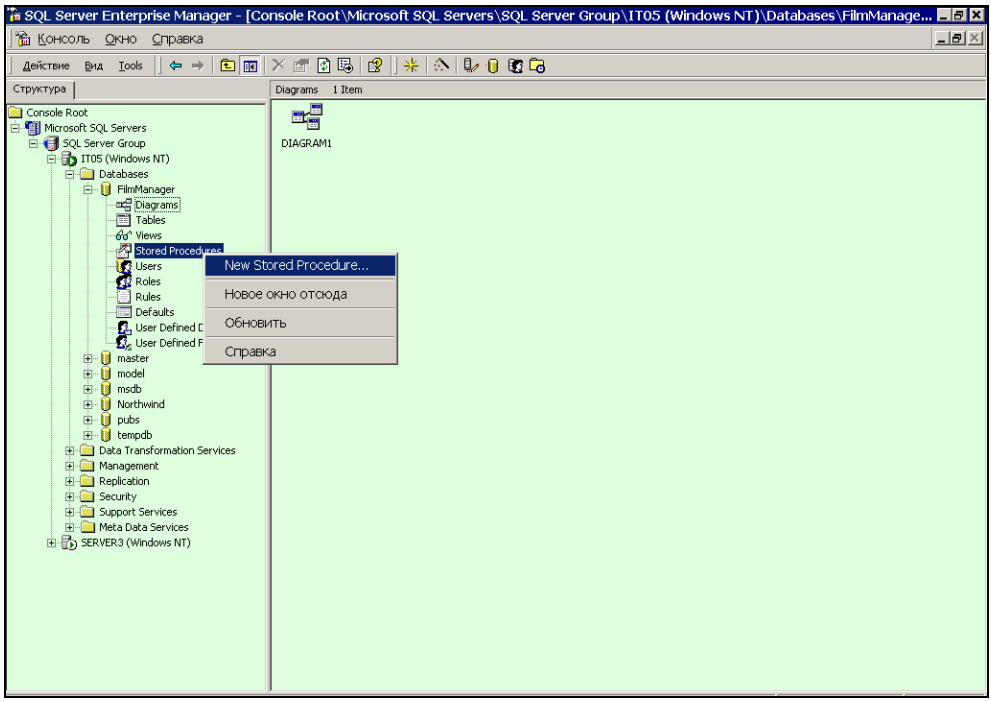


Рис. 4.36. Выбор пункта **New Stored Procedure** выпадающего меню

На экране появится окно **Stored Procedure Properties** (рис. 4.37).

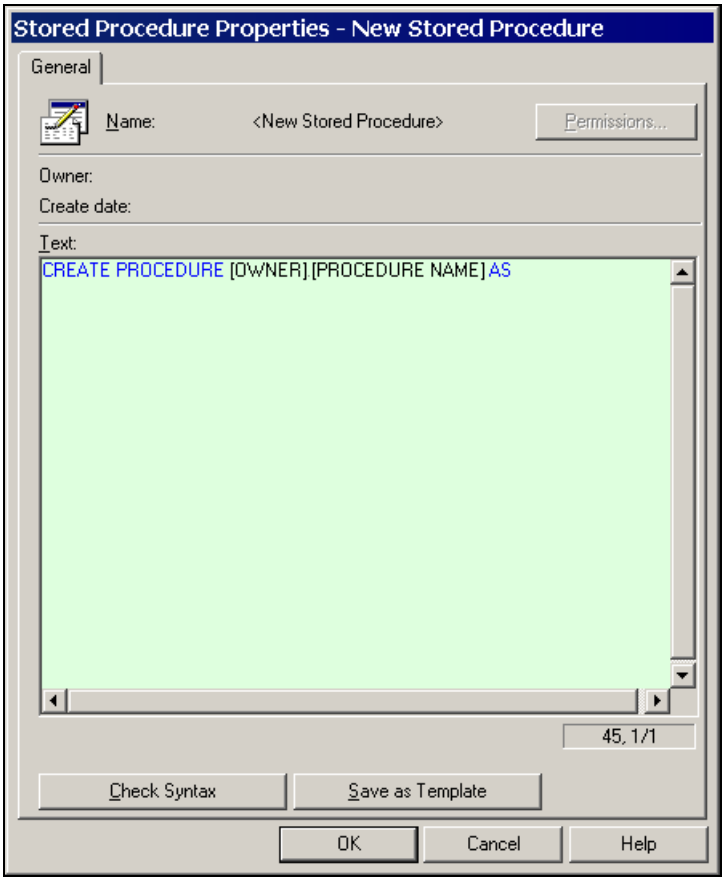


Рис. 4.37. Окно **Stored Procedure Properties**

В нем необходимо внести следующий текст процедуры добавления данных в таблицу Main (рис. 4.38):

```
CREATE PROCEDURE MainInsert
  @NameOfFilm  varchar(100),
  @GanrKod     int,
  @FilmAbout   varchar(3000),
  @PathToFilm  varchar(1000)
AS

INSERT INTO Main(NameOfFilm, GanrKod, FilmAbout, PathToFilm)

VALUES (@NameOfFilm, @GanrKod, @FilmAbout, @PathToFilm)
GO
```

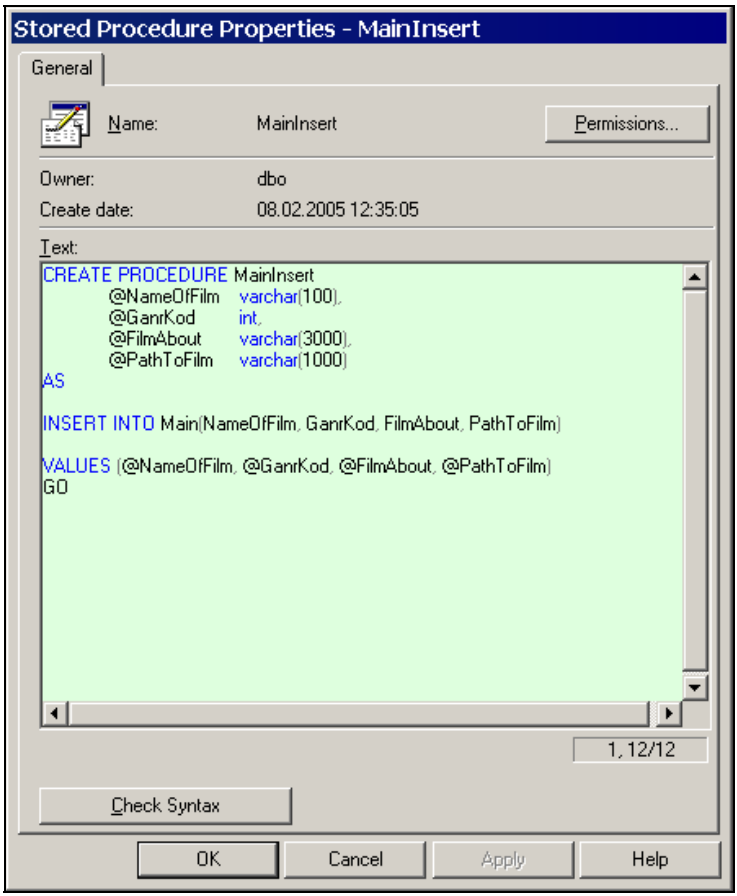


Рис. 4.38. Создание процедуры для добавления данных в таблицу Main

Разберем код процедуры:

- `CREATE PROCEDURE` — создать процедуру;
- `MainInsert` — название процедуры, по которому мы будем вызывать ее из клиентского приложения;
- Названия и типы переменных:

```
@NameOfFilm  varchar(100),
@GanrKod     int,
@FilmAbout   varchar(3000),
@PathToFilm  varchar(1000)
```

Переменные выделяются знаком "@" перед их именем, после имени ставится ее тип. Тип переменных точно соответствует типу атрибутов, которым мы будем передавать эти данные. То есть переменную `@NameOfFilm` мы передадим атрибуту `NameOfFilm` таблицы Main и т. д.;

- `AS` — зарезервированное слово, после которого идет тело процедуры;
- `INSERT INTO Main(NameOfFilm, GanrKod, FilmAbout, PathToFilm)` — добавить информацию в таблицу Main (`INSERT INTO Main`) в атрибуты, перечисленные в скобках. Информация берется из переменных, указанных после ключевого слова `VALUES`;
- `VALUES (@NameOfFilm, @GanrKod, @FilmAbout, @PathToFilm)` — переменные, в которых содержатся передаваемые значения, которые будут устанавливаться клиентским приложением, причем порядок переменных (`@NameOfFilm, @GanrKod, @FilmAbout, @PathToFilm`) и атрибутов (`NameOfFilm, GanrKod, FilmAbout, PathToFilm`), которым передаются их значения, должен совпадать;
- `GO` — оператор, сигнализирующий об окончании блока команд SQL.

Теперь нажимаем кнопку **Check Syntax**, после чего запустится проверка правильности составления процедуры. Если все написано правильно, то высветится окно как на рис. 4.39.

Рис. 4.39. Результат проверки правильности синтаксиса

Замечание

Если, например, вместо `CREATE PROCEDURE` написать `CRE PROCEDURE`, то SQL Server высветит сообщение об ошибке в тексте процедуры. Такую процедуру невозможно будет сохранить, пока ошибка не будет исправлена (рис. 4.40).

Если SQL Server высветил окошко с надписью **Syntax check successful!**, то нажимаем кнопку **OK**, иначе сверяем с текстом (см. рис. 4.38). Процедура создана. Остается создать еще две процедуры на изменение данных `UPDATE` и на удаление данных `DELETE` из таблицы Main.

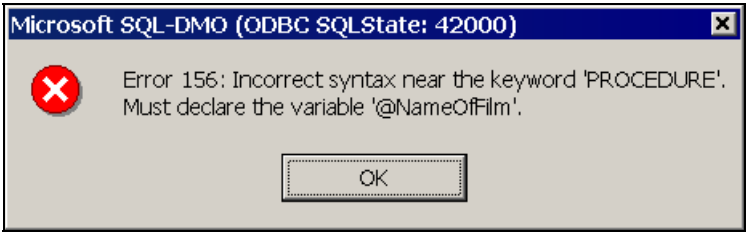


Рис. 4.40. Сообщение об ошибке в тексте процедуры

Создаем новую процедуру для изменения данных в таблице Main. Ниже представлен ее текст:

```
CREATE PROCEDURE MainEdit
  @FilmID      int,
  @NameOfFilm  varchar(100),
  @GanrKod     int,
  @FilmAbout   varchar(3000),
  @PathToFilm  varchar(1000)
AS

UPDATE Main SET
  NameOfFilm=@NameOfFilm,
  GanrKod=@GanrKod,
  FilmAbout=@FilmAbout,
  PathToFilm=@PathToFilm

WHERE
  FilmID=@FilmID
GO
```

Вначале задается имя процедуры (`MainEdit`), потом задаются переменные. Дальше указывается, что будет производиться изменение данных (команда `UPDATE`) и после зарезервированного слова `SET` указывается, значения каких переменных каким атрибутам будут присваиваться.

В этом запросе используется зарезервированное слово `WHERE`, которое указывает, какие записи будут редактироваться. В данном случае записи, у которых идентификатор фильма совпадает с переданным в переменную `@FilmID`, так как атрибут `FilmID` является ключевым; таким образом, меняться будет только одна запись.

Нажимаем **OK**. Если все введено было правильно, то процедура сохранится.

Создаем новую процедуру для удаления данных из таблицы Main. Ниже представлен ее текст:

```
CREATE PROCEDURE MainDelete
  @FilmID   int
AS

Delete from Main

WHERE
  FilmID=@FilmID
GO
```

Эта процедура осуществляет удаление записи из таблицы Main, идентификатор которой равен переданному в переменную `@FilmID` значению.

Нажимаем **OK**. На этом работа с процедурами закончена.

4.8. Резервное копирование и восстановление базы данных

Резервное копирование входит в обязанности администратора базы данных. Без эффективной стратегии резервного копирования можно попасть в ситуацию, когда база данных выходит из строя, а достаточно новой резервной копии для ее восстановления не существует. Эффективная стратегия резервного копирования позволяет восстановить базу данных при многих видах сбоев как в программном обеспечении, так и в работе оборудования.

Замечание

Резервное копирование предоставляет еще одну полезную возможность. Можно использовать резервную копию базы данных для быстрого переноса базы данных на другой сервер, просто скопировав туда файл, содержащий резервную копию, и восстановив базу данных.

Для того чтобы выполнить резервное копирование базы, нужно щелкнуть правой кнопкой мыши по названию базы и в выпадающем меню выбрать пункт **Все задачи** (All Tasks)**Backup Database** (рис. 4.41).

Также приступить к резервному копированию можно, выбрав пункт **Tools\Backup Database** главного меню (рис. 4.42).

Появится окно **SQL Server Backup** (рис. 4.43).

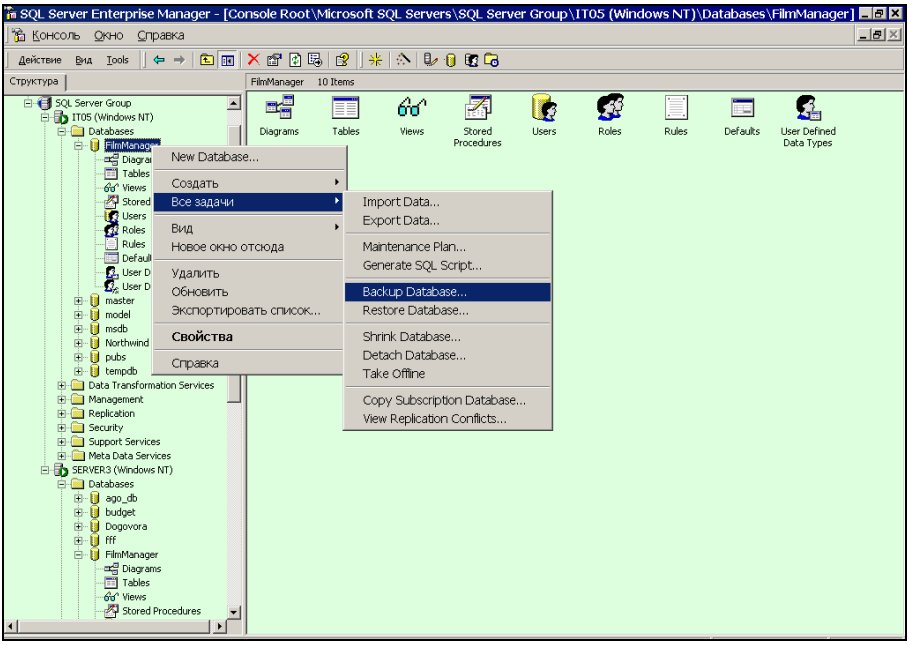


Рис. 4.41. Выбор пункта **Backup Database** выпадающего меню

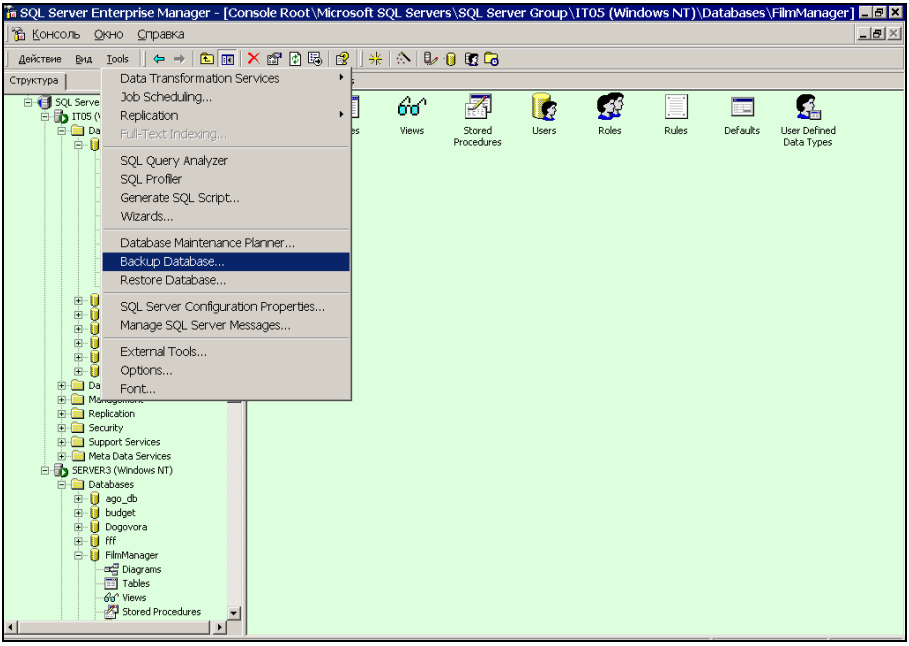


Рис. 4.42. Выбор пункта **Tools\Backup Database** главного меню

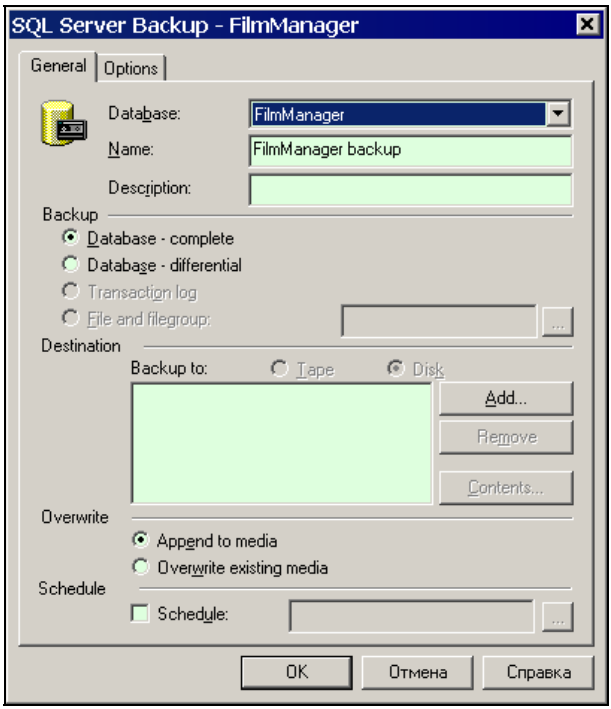


Рис. 4.43. Окно **SQL Server Backup**

В этом окне нажимаем кнопку **Add**. Появится окно **Select Backup Destination** (Выбор месторасположения резервной копии) (рис. 4.44).

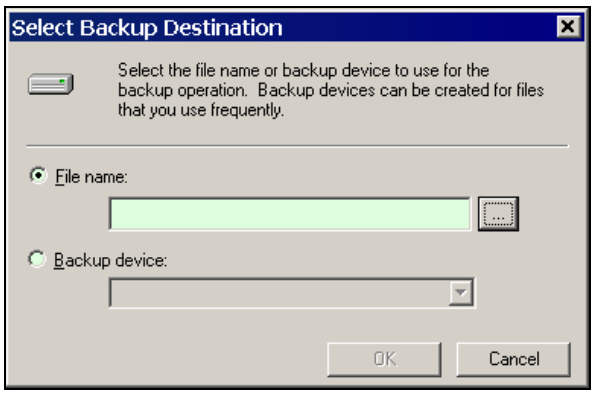


Рис. 4.44. Окно **Select Backup Destination**

Нажимаем на кнопку <...>, появляется следующее окно, в котором мы выбираем каталог для хранения архивной копии. В поле **File name** вводим имя архивного файла, оно может быть любым, главное, чтобы было понятно вам (у меня архивная копия называется **backup_FillmManager_copy1.dat**) — рис. 4.45.

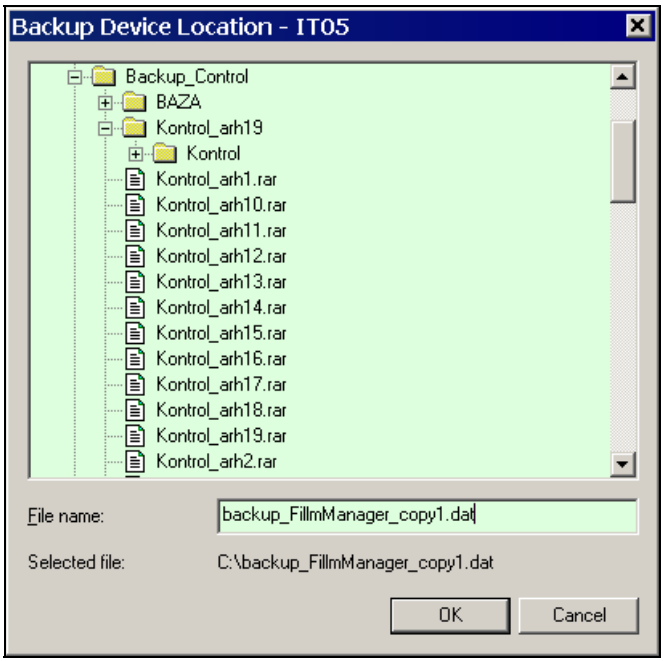


Рис. 4.45. Выбор архивного файла

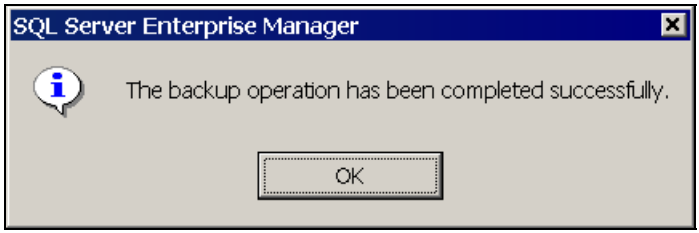


Рис. 4.46. Окно завершения процесса

Далее нажимаем **OK**, после этого необходимо еще несколько раз нажать **OK**, пока не начнется процесс создания копии. После того как архивная копия

создастся, появится информационное окно, в котором SQL Server проинформирует, что процесс завершен (рис. 4.46).

Для того чтобы выполнить восстановление резервной копии базы, нужно щелкнуть правой кнопкой мыши по названию базы и выбрать в выпадающем меню пункт **Все задачи** (All Tasks)**Restore Database** или выбрать в главном меню пункт **Tools\Restore Database**. Появится окно **Restore database** (рис. 4.47).

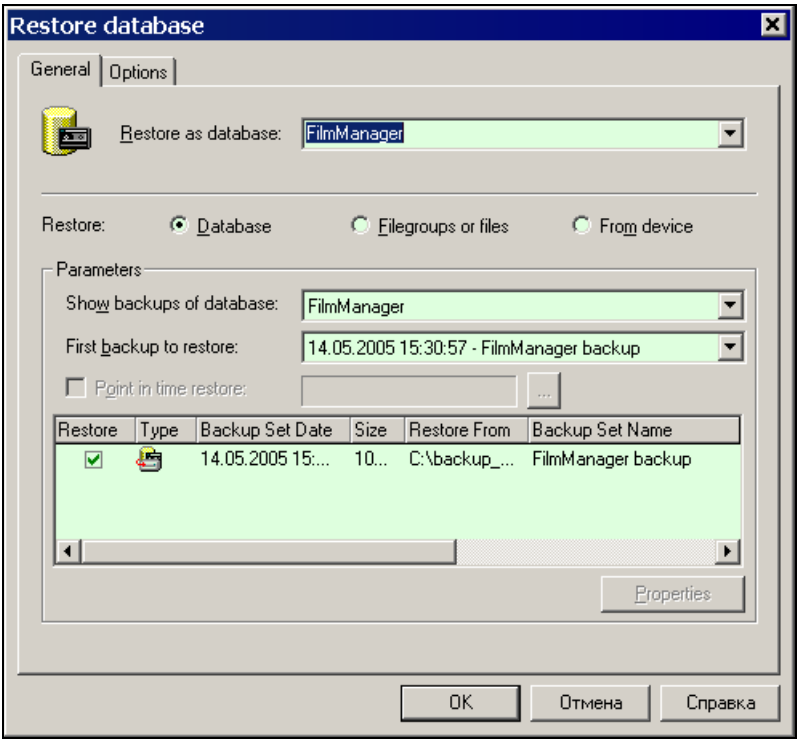


Рис. 4.47. Окно **Restore database**

В группе элементов **Restore** устанавливаем переключатель **From device**. Окно изменит свой вид, как показано на рис. 4.48.

Нажимаем кнопку **Select Devices**, появится окно **Choose Restore Devices**, в нем необходимо нажать кнопку **Add**, расположенную напротив переключателя **File**. Появится окно **Choose Restore Destination**. В нем необходимо указать файл архивной копии, который будет восстанавливаться. После этого несколько раз нажимаем **OK**, пока не начнется процесс восстановления.

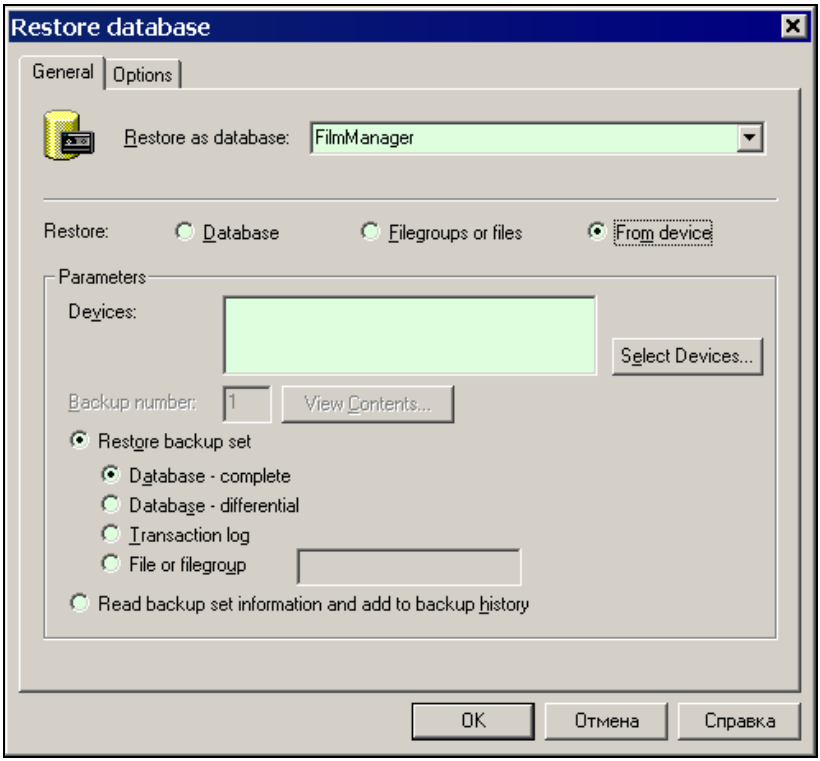


Рис. 4.48. Окно **Restore database** при выборе пункта **From device**

Когда архивная копия восстановится, появится информационное окно, в котором SQL Server проинформирует, что процесс завершен (рис. 4.49).

Рис. 4.49. Восстановление завершено

4.9. Разработка приложения "клиент-сервер" в Delphi

В данном разделе мы будем использовать компоненты, описанные в табл. 4.5—4.9.

Таблица 4.5. *Основные компоненты вкладки* ***ADO***

Название	Основные свойства	Комментарии
ADOConnection		Отвечает за связь с базой данных
	ConnectionString	Содержит настройки для соединения с базой
	LoginPromt	Логическое поле, отвечает за то, будет ли отображаться окно ввода имени пользователя и пароля каждый раз, когда происходит соединение с базой данных
	Connected	Логическое свойство. Если содержит True, то связь с базой данных установлена, в противном случае — False
ADOQuery		Позволяет отправлять запросы к базе данных и получать результат их выполнения
	Connection	Содержит имя компонента ADOConnection. Соответственно все запросы будут отправляться базе данных, на работу с которой настроен компонент ADOConnection
	SQL	SQL-запрос для работы с базой
	Active	Логическое свойство. Когда установлено в True, SQL-запрос выполняется и результат передается в клиентское приложение (туда, откуда он был отправлен)
ADOStoredProc		Предназначен для работы с хранимыми процедурами. Позволяет выполнять их, передавать входные данные и получать результат выполнения (выходные данные)
	Connection	Содержит имя компонента ADOConnection
	StoredProcName	Содержит имя хранимой процедуры

Таблица 4.6. *Основные компоненты вкладки* ***Standart***

Название	Основные свойства	Комментарии
Label		Предназначен для отображения текста
	Caption	Текст, отображаемый компонентом
Edit		Поле для ввода текста
	Text	В этом свойстве хранится введенный текст
ComboBox		Выпадающий список
	Items	Содержит элементы выпадающего списка

Таблица 4.7. *Основные компоненты вкладки* ***Win32***

Название	Основные свойства	Комментарии
DBRichEdit		Предназначен для отображения многостраничного текста
	Lines	Содержит текст

Таблица 4.8. *Основные компоненты вкладки* ***System***

Название	Основные свойства	Комментарии
Timer		Предназначен для выполнения одного и того же действия через заданные промежутки времени. Это действие может быть процедурой, функцией или любым другим кодом, указанным в обработчике события OnTimer
	Interval	Задает промежуток времени в миллисекундах, через который будет возникать событие OnTimer
	Enabled	Логическое свойство. Если установлено в True, то таймер включен, иначе находится в отключенном состоянии
MediaPlayer		Используется для воспроизведения звуковых и видеофайлов
	FileName	Имя файла, который будет воспроизводиться

Таблица 4.9. *Основные компоненты вкладки* ***Dialogs***

Название	Основные свойства	Комментарии
OpenDialog		Предназначен для отображения стандартного окна Windows для выбора файла
	FileName	Содержит имя выбранного файла

Также будут использоваться компоненты DataSource, DBGrid, DBNavigator, Button — их описание можно найти в *разд. 3.7*.

Замечание

Советую под новый проект создать отдельную папку. У меня она называется C:\VIDEOMANAGER.

Запускаем Delphi. Сначала будем создавать модуль безопасности, где можно будет заводить пользователей и давать им пароли. Создаем новое приложение. Название формы оставляем по умолчанию Form1, сохраняем ее в модуле под именем USecurityModule, а проект — под именем SecurityModule.

Нам понадобятся следующие компоненты, поместите их на форму:

- ADOConnection
- ADOQuery
- DataSource
- DBGrid
- DBNavigator
- 2 компонента Button

У меня форма с расположенными на ней компонентами выглядит как на рис. 4.50.

Теперь настроим свойства компонентов. Для ADOConnection щелкаем на свойстве ConnectionString, появляется окно (рис. 4.51).

Нажимаем кнопку **Build**. Появится окно **Data Link Properties**, которое предназначено для настройки свойств соединения с базой данных (рис. 4.52).

На вкладке **Provider** выбираем драйвер доступа к базе данных — **Microsoft OLE DB Provider for SQL Server**. На вкладке **Connection** выбираем имя сервера (поле **Select or enter a server name**), необходимо выбрать тот сервер, который у вас доступен. У меня (рис. 4.53) доступно несколько, но мой локальный (с которым я работаю) называется IT05, вот я его и выбираю.

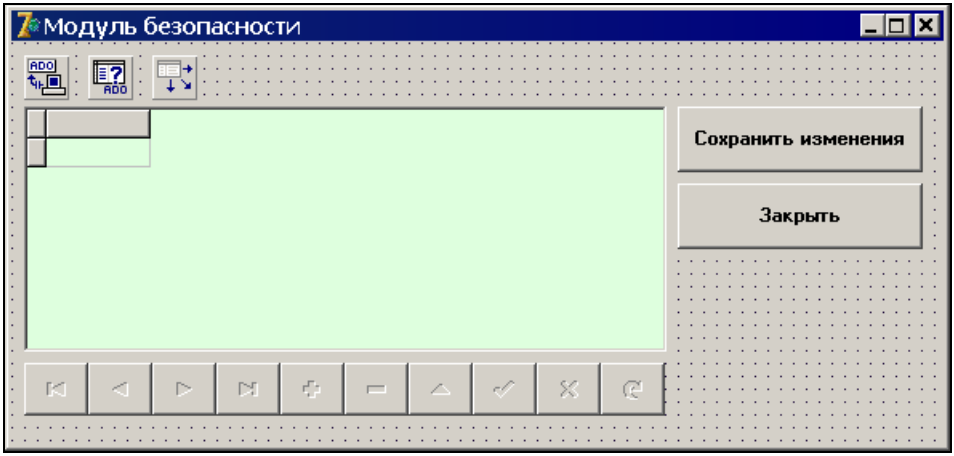


Рис. 4.50. Модуль безопасности

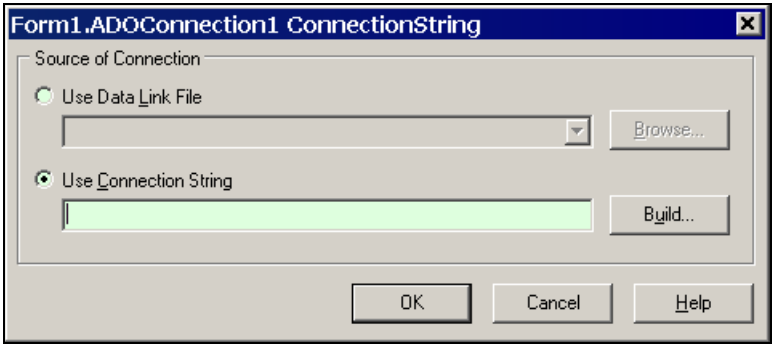


Рис. 4.51. Окно установки **ConnectionString**

Далее необходимо установить способ аутентификации пользователя при подключении к серверу (группа элементов **Enter information to log on the server**). Если выбрать переключатель **Use Windows NT Integrated Security**, то будет использоваться политика безопасности Windows. То есть если у вас локальная сеть с доменом, каждый пользователь входит в Windows под своим именем и паролем, то эти же имя и пароль будут посылаться серверу при попытке установить с ним соединение.

Если выбрать **Use a specific user name and password**, то необходимо будет заполнить поля:

- **User name** — имя пользователя;
- **Password** — пароль.

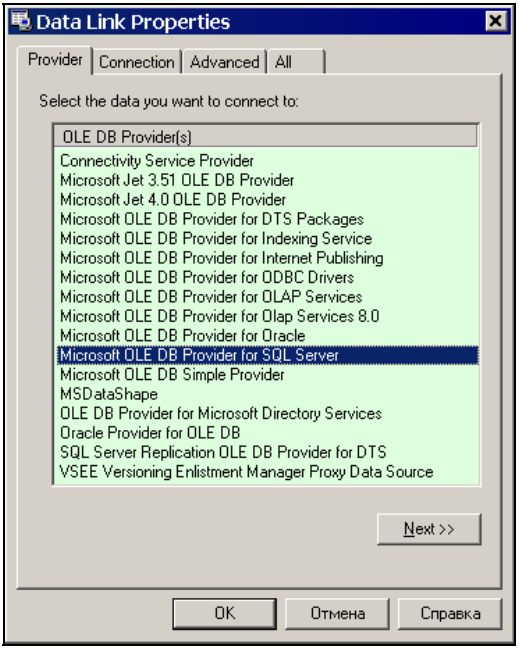


Рис. 4.52. Окно настройки свойств соединения с базой данных

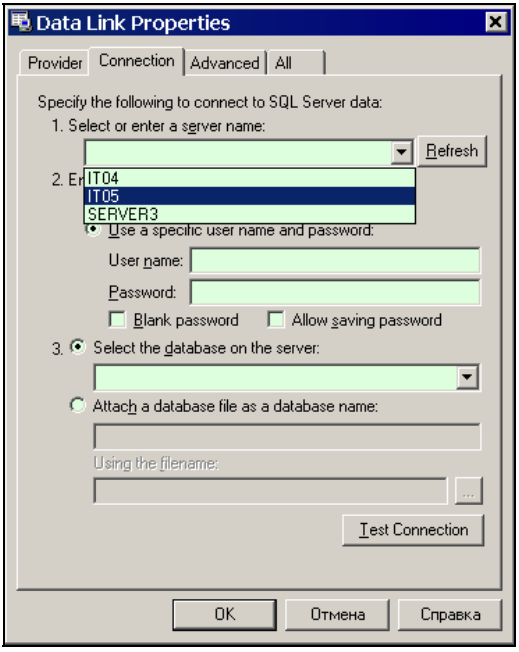


Рис. 4.53. Выбор сервера

Замечание

Если установка MS SQL Server на компьютер производилась самостоятельно, то обязательно будет существовать суперпользователь под именем sa, так вот его можно и использовать (конечно только в учебных целях). Также можно создать нового пользователя и соединяться с сервером от его имени.

Выбираем имя базы (**Select the database on the server**), в нашем случае это FilmManager (рис. 4.54.

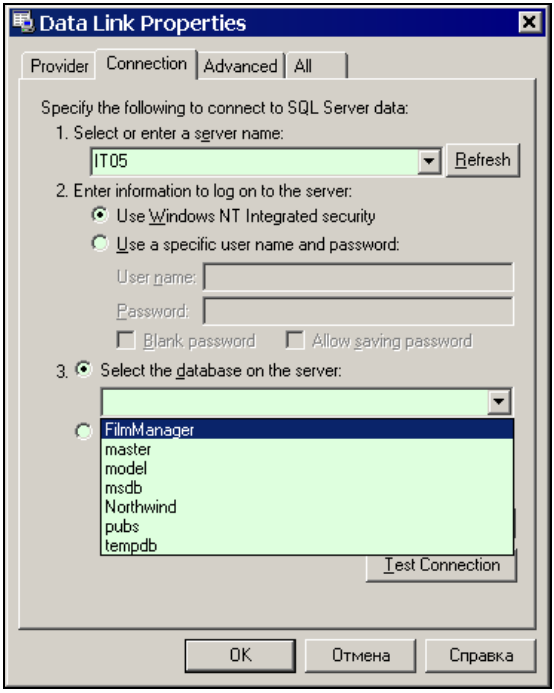


Рис. 4.54. Выбор имени базы данных

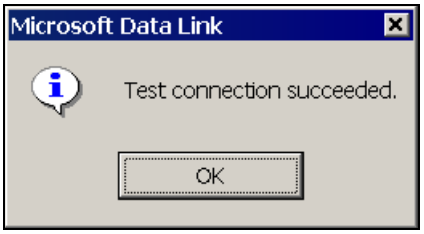


Рис. 4.55. Результат проверки соединения с базой данных

Нажимаем кнопку **Test Connection**. Если все было правильно настроено, то появится окно как на рис. 4.55.

Нажимаем **OK**. Соединение с базой настроено.

Свойство `LoginPrompt` необходимо установить в `False`, чтобы каждый раз при запуске программы не появлялось окошко ввода имени пользователя и пароля. Для компонента `ADOQuery` в свойстве `Connection` устанавливаем `ADOConnection1`, чтобы компонент смог посылать запросы к базе.

В свойстве `SQL` пишем запрос на выбор данных из таблицы SecurityTable:

```
SELECT * FROM SecurityTable
```

Щелкаем два раза левой кнопкой мыши на компоненте `ADOQuery` и в появившемся окошке щелкаем правой кнопкой мыши и в выпадающем меню выбираем пункт **Add fields** (рис. 4.56).

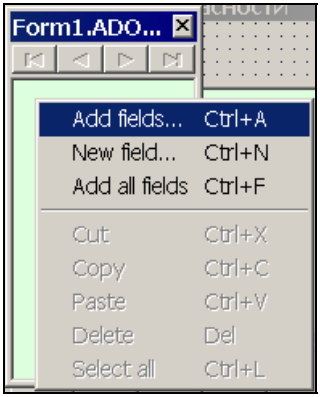


Рис. 4.56. Выпадающее меню окна со списком полей

Появится окно **Add Fields**, в котором будут отображены атрибуты таблицы SecurityTable (рис. 4.57).

Нажимаем **OK**, после чего можно будет увидеть выбранный список полей (рис. 4.58).

Выбираем `UserID` и в инспекторе объектов свойство `Visible` ставим в `False`. Теперь этот атрибут не будет отображаться на экране во время работы с программой. В свойство `DataSet` компонента `DataSource` устанавливаем `ADOQuery1`. В свойство `DataSource` компонентов `DBGrid` и `DBNavigator` ставим `DataSource1`.

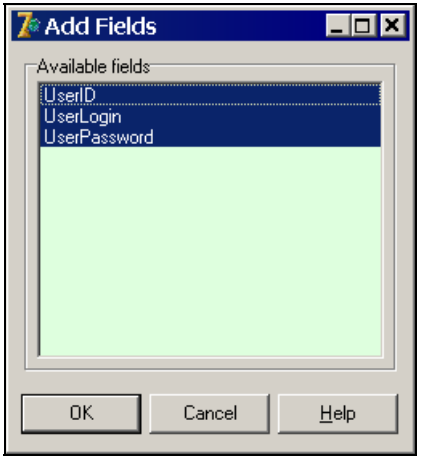


Рис. 4.57. Атрибуты таблицы SecurityTable

Рис. 4.58. Выбранный список полей таблицы SecurityTable

Для кнопки "Сохранить изменения" пишем код сохранения изменений в таблице SecurityTable:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 {Начинаем транзакцию}
 ADOConnection1.BeginTrans;

 {Ставим защитную конструкцию от сбоев}
 try
        {Если данные в режиме редактирования, то сохраняем
        методом Post}
        if DataSource1.DataSet.State in
                  [dsInsert,dsEdit] then

            {Сохраняем данные}
            DataSource1.DataSet.Post;
  {У нас возникло исключение или данные нельзя сохранить}
  except
    {откатываем транзакцию}
    ADOConnection1.RollbackTrans;

    {Говорим пользователю, что не получилось сохранить}
    ShowMessage('Невозможно сохранить данные! Повторите попытку.');
```

```
    {Отменяем изменения, чтобы у нас на рабочем месте
     отображались реальные данные, без этой строчки кода
     на сервер ничего не передастся, но
     мы будем видеть внесенные изменения}
    ADOQuery1.CancelUpdates;

    {Обновляем набор данных}
    ADOQuery1.Refresh;
    {Выход из процедуры}
    exit
 end;
  {Если не возникло исключений, подтверждаем транзакцию}
  ADOConnection1.CommitTrans;
end;
```

Для кнопки "Закрыть" пишем код завершения работы с модулем безопасности:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
   {Закрываем набор данных, полученный SQL-запросом}
   ADOQuery1.Close;
   {Закрываем сеанс связи с базой данных}
   ADOConnection1.Close;
   {Закрываем форму и, соответственно, все приложение,
   т. к. форма у нас одна}
   Form1.Close;
end;
```

Выбираем пункт меню **Project\View Source** и после строчки `Application.CreateForm(TForm1, Form1);` пишем свой код на открытие сеанса связи с базой данных и открытие наборов данных во время загрузки приложения:

```
  {Защищаем наш код от сбоев}
  try
   {Открываем сеанс связи с БД}
   Form1.ADOConnection1.Open;
   {Открываем доступ к таблице SecurityTable}
   Form1.ADOQuery1.Open;
  {Если произошел сбой, то сообщить об этом пользователю}
```

```
except
 ShowMessage('Не удалось установить связь с БД, сервер недоступен.');
end;
```

Еще нам понадобится подключить модуль `Dialogs`. Иначе компилятор будет ругаться на процедуру `ShowMessage`. Модуль безопасности готов. Теперь можно создавать пользователей, назначать им пароли.

Приступаем к более серьезным вещам, написанию второго модуля. Для этого нам потребуется создать новый проект. Форму называем `MainForm`. Выбираем пункт **Save All** главного меню. Модуль сохраняем под именем `UMain`, проект — под именем `VideoManager`. Создаем новый `DataModule` (пункт меню **File\New\DataModule**). Называем его `DM`, сохраняем его под именем `UDM`.

Помещаем на `UDM` компоненты `ADOConnection`, `ADOQuery` и три `ADOStoredProcedure`. Как это выглядит у меня, можно увидеть на рис. 4.59.

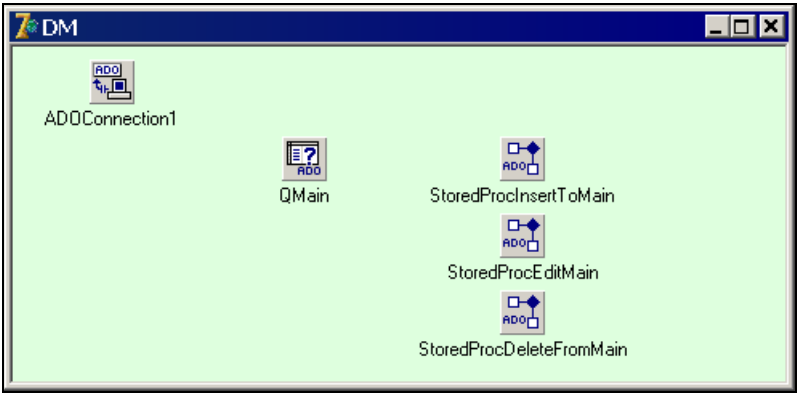


Рис. 4.59. Компоненты модуля **UDM**

Переходим к настройке компонентов. `ADOConnection` настраиваем точно так же, как для модуля безопасности. `ADOQuery1` переименовываем в `QMain`, в свойство `Connection` устанавливаем `ADOConnection1` и в свойстве `SQL` пишем запрос выбора данных из таблицы Main:

```
SELECT * FROM Main
```

Производим двойной щелчок левой кнопкой мыши на компоненте `QMain` и в появившемся окошке щелкаем правой кнопкой мыши и выбираем **Add all fields**. Выбираем `FilmID` и в инспекторе объектов свойство `Visible` устанавливаем в `False`, также делаем невидимым атрибут `FilmAbout`.

Для компонента `ADOStoredProc1` устанавливаем следующие свойства:

- Name — StoredProcInsertToMain
- Connection — ADOConnection1
- ProcedureName — MainInsert;1

Если произвести щелчок мыши на свойстве Parameters, то можно увидеть параметры, которые будет обрабатывать процедура.

Для компонента ADOStoredProc2 устанавливаем следующие свойства:

- Name — StoredProcEditMain
- Connection — ADOConnection1
- ProcedureName — MainEdit;1

Для компонента ADOStoredProc3 устанавливаем следующие свойства:

- Name — StoredProcDeleteFromMain
- Connection — ADOConnection1
- ProcedureName — MainDelete;1

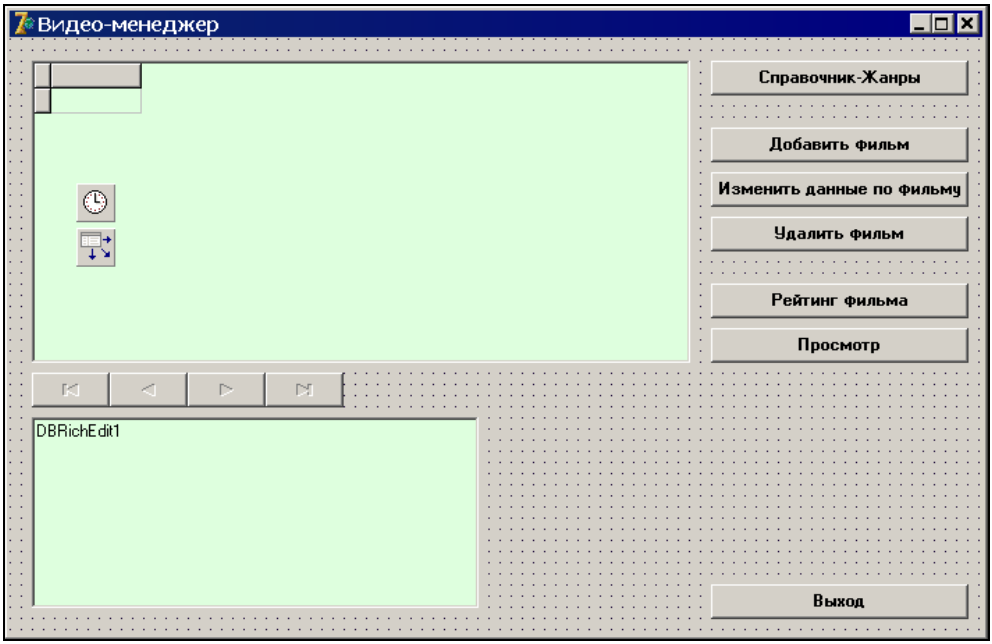


Рис. 4.60. Форма MainForm

Переходим к форме MainForm, необходимо расположить на ней следующие компоненты (рис. 4.60:

- DBGrid
- DBNavigator
- 7 штук Button
- DataSource
- Timer
- DBRichEdit

Свойство Enable компонента Timer устанавливаем в False.

Выбираем пункт **File\USE Unit** главного меню (рис. 4.61).

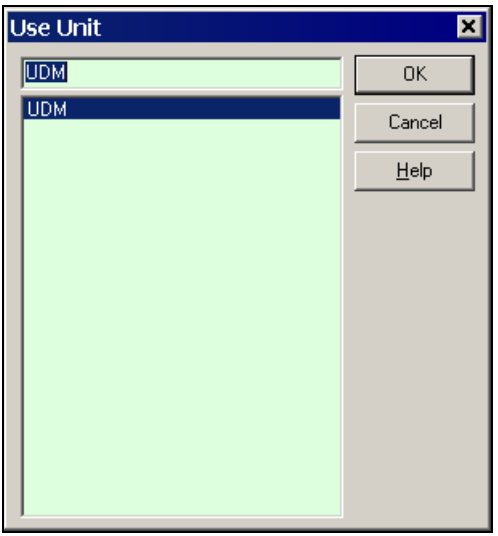


Рис. 4.61. Окно **Use Unit** (Delphi 7)

Выбираем UDM и нажимаем **OK**. Этими действиями мы подключаем модуль UDM к главной форме. Это нужно, чтобы можно было подключить компонент QMain к компоненту DataSource, так как они расположены у нас на разных формах.

В свойство DataSet компонента DataSource устанавливаем QMain. В свойство DataSource компонентов DBGrid, DBRichEdit и DBNavigator ставим DataSource1. У DBGrid свойство ReadOnly ставим в True, это нужно для того, чтобы пользователь мог менять информацию только с помощью специальных кнопок. Для компонента DBNavigator свойство VisibleButtons необходимо сделать как на рис. 4.62.

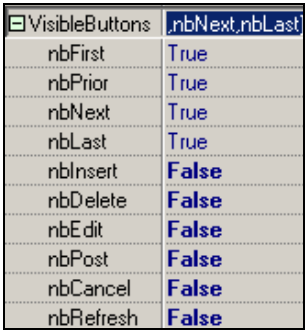


Рис 4.62. Настройка компонента `DBNavigator`

В свойство `DataField` компонента `DBRichEdit` устанавливаем `FilmAbout`. Таким образом, мы сообщаем нашей программе, что в этом компоненте будет отображаться содержимое атрибута `FilmAbout` для текущей записи таблицы `Main`.

Компонент `Timer` необходим, чтобы информация в базе обновлялась динамически через заданные промежутки времени. Свойство `Interval` зададим равным 10 000, то есть обновлять будем каждые 10 секунд (значение в мс). В событии `OnTimer` пишем код для обновления данных таблицы Main:

```
procedure TMainForm.Timer1Timer(Sender: TObject);
var
 N:integer; {будет содержаться код выбранной записи}
begin
 {запомним код выбранной записи, чтобы потом вернуться}
 N:=DM.QMainFilmID.Value;
 {Обновим набор данных}
 DM.QMain.Close;
 DM.QMain.Open;
 {Вернемся к записи, которая была активной, методом Locate
 первый — параметр имя поля, второй — искомое значение,
 третий — условия отбора при поиске}
 DM.QMain.Locate('FilmID',N,[]);
end;
```

Теперь мы будем получать доступ к базе, но нам надо, чтобы пользователь проходил аутентификацию перед входом в программу. Этим сейчас и займемся.

Создаем новую форму, называем ее FormSecurity. Сохраняем под именем USecurity. Помещаем на нее 2 компонента Label, 2 Edit, ADOQuery (не на DataModule, потому что этот компонент нам будет необходим на небольшой промежуток времени), 2 Button (рис. 4.63).

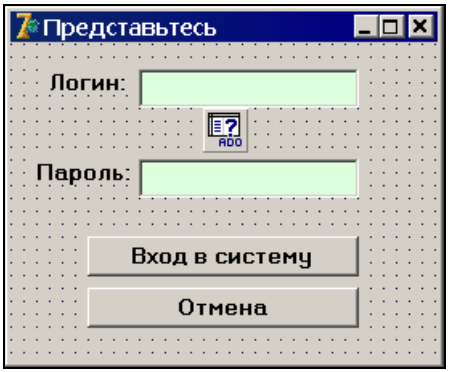


Рис. 4.63. Форма FormSecurity

ADOQuery1 переименовываем в QSecurityTable, свойство ConnectionString настраиваем аналогично тому, как мы настраивали свойство ConnectionString для компонента ADOConnection, то есть компонент QSecurityTable у нас будет соединяться с базой данных напрямую. В свойстве SQL пишем запрос проверки имени и пароля, введенного пользователем:

```
SELECT * FROM SecurityTable WHERE UserLogin=:Par1 AND UserPassword=:Par2
```

Данный запрос означает: выбрать пользователя из таблицы SecurityTable, у которого имя и пароль совпадают с введенными (и, соответственно, переданными через параметры). Если переданные данные никому не соответствуют, значит запрос возвратит пустой набор данных, и мы поймем, что это чужак, и не предоставим доступ к базе данных.

Щелкаем два раза левой кнопкой мыши на компоненте QSecurityTable, в появившемся окошке щелкаем правой кнопкой мыши и выбираем в выпадающем меню пункт **Add all fields**. Щелкаем по свойству Parameters компонента QSecurityTable, появляется окно (рис. 4.64).

Выделяем запись Par1 и убеждаемся, что в инспекторе объектов свойство DataType установлено в ftString. Те же действия повторяем и для Par2.

Выбираем пункт **Project\Option** главного меню. И с помощью стрелок переносим форму FormSecurity в правую половину окна; таким образом мы говорим Delphi, что во время работы программы мы будем сами следить за созданием этой формы и ее удалением (рис. 4.65).

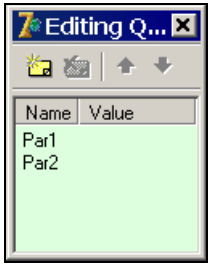


Рис. 4.64. Окно параметров запроса

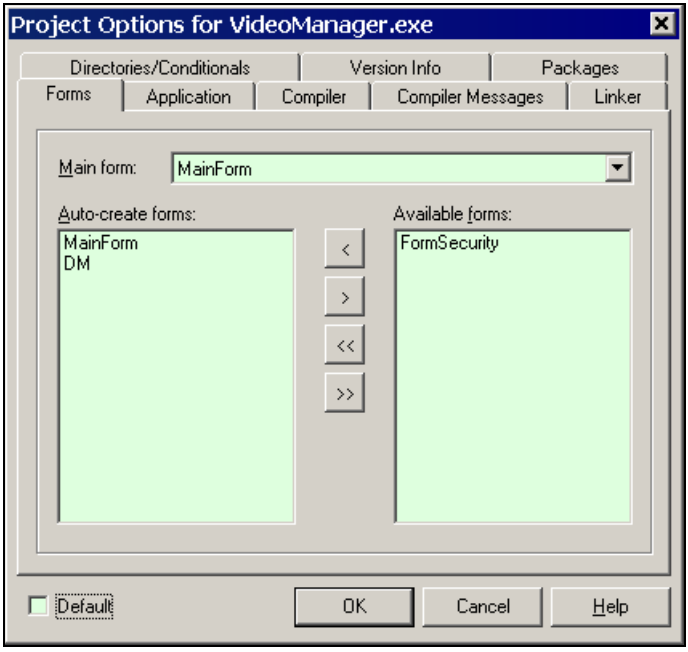


Рис. 4.65. Вкладка **Forms** окна **Projects Options**

Нажимаем **OK**.

Переходим на форму `MainForm` и создаем событие `OnShow`, в котором выводим форму `FormSecurity` для аутентификации пользователя при запуске программы:

```
procedure TMainForm.FormShow(Sender: TObject);
begin
 {Если свойство Tag установлено в 0,
 то значит, мы только что запустили приложение
```

```
 и нам нужно отобразить окно аутентификации}
 if MainForm.Tag=0 then
 begin
  {Создаем и отображаем окно аутентификации}
  FormSecurity:=TFormSecurity.Create(self);
  FormSecurity.ShowModal;
 end;
end;
```

Выбираем пункт **File\USE Unit** главного меню и подключаем модуль `USecurity`. Нажимаем <F9>. Должно запуститься приложение, появится окно аутентификации. Теперь закройте его. Ничего себе: у нас отобразилась Главная форма! Так не должно быть, если пользователь закрывает окно для ввода пароля, то должно закрываться и все приложение. Сейчас мы это исправим.

Делаем активной форму `FormSecurity`. Подключаем модуль `UMain`. Для события `OnCloseQuery` пишем код обработки ситуации, когда пользователь игнорирует окно аутентификации:

```
{Процедура, возникающая перед закрытием формы, переменная CanClose
указывает: можно ли закрыть форму или отменить закрытие}
procedure TFormSecurity.FormCloseQuery(Sender: TObject;
  var CanClose: Boolean);
begin
 {Если свойство Tag установлено в 0,
  значит, пользователь не прошел аутентификацию,
а просто хочет закрыть окно}
 if MainForm.Tag=0 then
  {Закрываем приложение. Зачем оно ему,
  если он не знает пароля?}
  Application.Terminate
 else
  {Пользователь прошел аутентификацию и поэтому
  окно аутентификации можно закрыть}
  CanClose:=True;
end;
```

Для кнопки "Отмена" пишем:

```
FormSecurity.Close;
```

Объявляем глобальную переменную `id_User`. В ней будет храниться идентификатор пользователя, вошедшего в систему. Сделайте активной форму `MainForm` нажмите <F12>. После раздела `Var`, но до раздела `Implementation` напишите `id_user:integer;`. У меня получилось так:

```
Var
  MainForm: TMainForm;
  id_user:integer;

implementation
```

Делаем активной форму `FormSecurity` и для кнопки "Вход в систему" пишем код проверки данных аутентификации и разрешения или запрета пользователю работать с программой:

```
procedure TFormSecurity.Button1Click(Sender: TObject);
begin
{Задаем значение параметров для запроса,
первый — логин, второй — пароль}
 with QSecurityTable.Parameters do
 begin
   ParamByName('Par1').Value:=Edit1.Text;
   ParamByName('Par2').Value:=Edit2.Text;
 end;

{Выполняем запрос}
 QSecurityTable.Open;

{Если количество возвращенных данных запросом больше 0,
значит, пользователь ввел правильные данные. Количество
возвращенных записей хранится в свойстве RecordCount}
 if QSecurityTable.RecordCount>0 then
 begin
  {Присваиваем свойству Tag главной формы 1,
  которая указывает на то, что аутентификация пройдена}
  MainForm.Tag:=1;

  {Запоминаем код пользователя}
  id_user:=QSecurityTableUserID.Value;
  QSecurityTable.Close;
```

```
  {Закрываем окно аутентификации}
  FormSecurity.Close;
  {Открываем пользовательский сеанс связи с БД}
  DM.ADOConnection1.Open;
  {Открываем набор данных, возвращающий список фильмов}
  DM.QMain.Open;
  {Открываем Timer}
  MainForm.Timer1.Enabled:=true;
  {Выходим из процедуры, т. к. дальше обрабатывать нечего}
  exit;

 end;

{Если пользователь ввел неправильные данные,
то проинформировать его об этом}
ShowMessage('В доступе отказано');
{Закрываем набор данных}
QSecurityTable.Close;

end;
```

Создаем новую форму. Называем ее `FormGanri` и сохраняем под именем `UGanri`. Помещаем компоненты `DBGrid`, `DBNavigator`, `Button` — 3 штуки, `ADOQuery`, `DataSource` (рис. 4.66).

Свойство `Enabled` кнопок "Выбрать" и "Сохранить изменения", устанавливаем в `False`. Кнопку "Выбрать" называем `ButtonChoose`, а кнопку "Сохранить изменения" — `ButtonSave`. Для компонента `DBNavigator` свойство `VisibleButtons` сделайте следующим, как на рис. 4.67.

`ADOQuery1` переименовываем в `QGanri`, подключаем модуль `UDM`. Свойство `Connection` заполняем `DM.ADOConnection1` и в свойстве `SQL` пишем запрос выборки всех данных из таблицы Ganri:

```
SELECT * FROM Ganri
```

Данный запрос означает: выбрать все записи из таблицы Ganri. Щелкаем два раза левой кнопкой мыши на компоненте `QGanri`, в появившемся окне щелкаем правой кнопкой мыши и выбираем в выпадающем меню пункт **Add all fields**. Поле `GanriID` делаем невидимым.

В свойство `DataSet` компонента `DataSource` устанавливаем `QGanri`. В свойство `DataSource` компонентов `DBGrid` и `DBNavigator` ставим `DataSource1`.

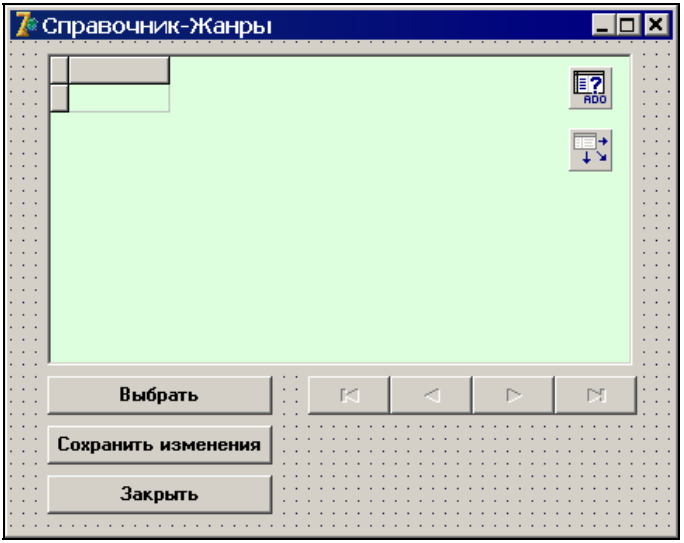


Рис. 4.66. Форма `FormGanri`

VisibleButtons	nbNext,nbLast
nbFirst	True
nbPrior	True
nbNext	True
nbLast	True
nbInsert	**False**
nbDelete	**False**
nbEdit	**False**
nbPost	**False**
nbCancel	**False**
nbRefresh	**False**

Рис. 4.67. Свойство `VisibleButtons` компонента `DBNavigator`

Для кнопки "Сохранить изменения" пишем код для сохранения изменений в таблице Ganri:

```
procedure TFormGanri.ButtonSaveClick(Sender: TObject);
begin
 {Запускаем транзакцию}
 DM.ADOConnection1.BeginTrans;
 {Ставим защитный блок try-except}
try
          {Если данные в режиме редактирования или вставки, то
           сохранить их}
```

```
        if DataSource1.DataSet.State in
                  [dsInsert,dsEdit] then

            DataSource1.DataSet.Post;
  {Если возникло исключение}
  except
    {Откатываем транзакцию}
    DM.ADOConnection1.RollbackTrans;
    {Сообщаем пользователю об ошибке}
    ShowMessage('Операция не удалась. Повторите попытку.');
   {Отменяем изменения, внесенные пользователем}
    QGanri.CancelUpdates;
    {Обновляем набор данных}
    QGanri.Refresh;
    {Больше обрабатывать нечего, выходим из процедуры}
    exit
 end;
  {Если исключений не возникло при сохранении,
  то подтвердить транзакцию}
   DM.ADOConnection1.CommitTrans;
end;
```

Для кнопки "Закрыть" пишем код закрытия формы `FormGanri`:

```
procedure TFormGanri.Button3Click(Sender: TObject);
begin
 FormGanri.Close;
end;
```

Убираем форму `FormGanri` из списка автоматически создаваемых. Переходим на форму `MainForm`, подключаем к ней модуль `UGanri`. Для кнопки "Справочник — жанры" пишем код вызова этого справочника:

```
procedure TMainForm.Button1Click(Sender: TObject);
begin
 {Создаем форму}
 FormGanri:=TFormGanri.Create(self);
 {Устанавливаем свойстве Tag, по нему будем
 определять, какую кнопку делать доступной:
  ButtonChoose или ButtonSave}
```

```
 FormGanri.Tag:=1;
 {Открываем набор данных}
 FormGanri.QGanri.Open;
 {Отображаем форму}
 FormGanri.ShowModal;
end;
```

Создаем обработчик события `OnShow` формы `FormGanri`, определяющий какие кнопки будут доступны на форме `FormGanri`:

```
procedure TFormGanri.FormShow(Sender: TObject);
begin
{Если свойство Tag=1 значит, делаем доступной кнопку
Сохранить изменения, иначе
делаем доступными обе кнопки}
 if FormGanri.Tag=1 then
  ButtonSave.Enabled:=true
 else
  begin
   ButtonSave.Enabled:=true;
   ButtonChoose.Enabled:=true;
  end;
end;
```

Для события `OnClose` формы `FormGanri` пишем код:

```
procedure TFormGanri.FormClose(Sender: TObject; var Action: TCloseAction);
begin
 {Закрываем набор данных перед закрытием формы}
 QGanri.Close;
end;
```

Создаем новую форму, называем ее `FormModifyData`, сохраняем под именем `UModifyData`. Подключаем к ней модуль `UGanri`. Делаем так, чтобы форма не создавалась автоматически.

Расположите на ней следующие компоненты: 3 штуки `Edit`, `RichEdit`, 5 штук `Button`, 4 штуки `Label`, `OpenDialog` (рис. 4.68).

Для кнопок "Добавить" и "Изменить" установите свойство `Enable` в `False`. Компоненты `Edit` назовите: `EditNameOfFile`, `EditGanr`, `EditPathToFilm`. А компонент `RichEdit` — `RichEditFilmAbout`. Кнопку "Добавить" назовите

ButtonAdd, кнопку "Изменить" назовите ButtonEdit. Свойство ReadOnly для компонентов EditGanr и EditPathToFilm установите в True. Свойство Text для всех Edit сделайте пустым.

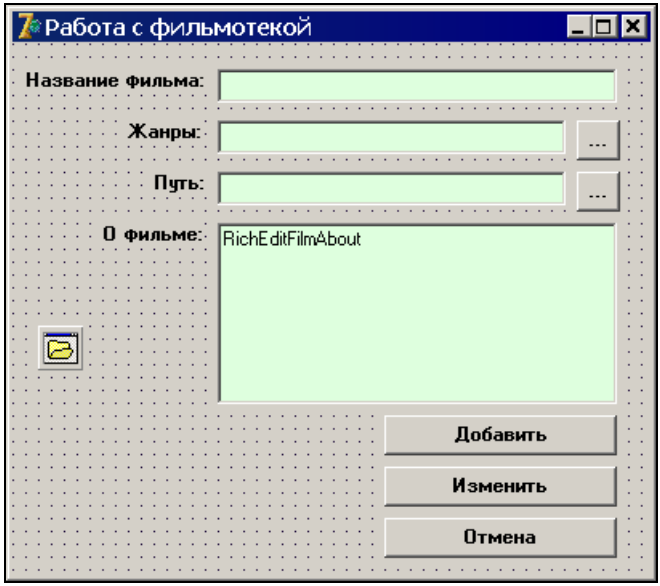


Рис. 4.68. Форма FormModifyData

Для кнопки "", расположенной напротив метки с текстом "Жанры", создаем код вызова справочника жанров:

```
procedure TFormModifyData.Button1Click(Sender: TObject);
begin
 {Создаем форму}
 FormGanri:=TFormGanri.Create(self);
 {Устанавливаем значение Tag, чтобы были
 доступны все кнопки}
 FormGanri.Tag:=0;
 {Открываем набор данных}
 FormGanri.QGanri.Open;
{Показываем окно}
 FormGanri.ShowModal;
end;
```

Переходим на форму FormGanri, подключаем модуль UModifyData и для кнопки "Выбрать" пишем код:

```
procedure TFormGanri.ButtonChooseClick(Sender: TObject);
begin
 {Присваиваем EditGanr выбранный пользователем жанр}
 FormModifyData.EditGanr.Text:=QGanriNameOfGanr.Value;
{Запоминаем код выбранного жанра, потому что
именно он пойдет в базу}
FormModifyData.EditGanr.Tag:=QGanriGanrID.Value;
{Закрываем справочник Жанры}
 FormGanri.Close;
end;
```

Переходим на форму FormModifyData. Для кнопки "", расположенной напротив метки с текстом "Путь к фильму", создаем код:

```
procedure TFormModifyData.Button2Click(Sender: TObject);
begin
{Если пользователь выбрал имя фала,
то присвоить полный путь к файлу свойству Text
компонента EditPathToFilm}
 If OpenDialog1.Execute then
  EditPathToFilm.Text:=OpenDialog1.FileName;
end;
```

Для кнопки "Отмена" пишем код:

```
FormModifyData.Close;
```

Подключаем модуль UDM. И для кнопки "Добавить" пишем код:

```
procedure TFormModifyData.ButtonAddClick(Sender: TObject);
{Объявляем переменные}
var
 b:variant; {Будет хранить многострочный текст, полученный из RichEdit}
 i:integer;
begin
{Проверяем, все ли необходимые поля заполнены}
If (EditNameOfFilm.Text='') or
    (EditGanr.Text='')       or
    (EditPathToFilm.Text='')
```

```
{Если нет, то выводим сообщение и выходим из
процедуры с помощью Exit}
 then
  begin
     ShowMessage('Заполнены не все поля!');
     exit;
  end;

try
 {Объявляем начало транзакции}
 DM.ADOConnection1.BeginTrans;
 {конструкция with}
 with DM.StoredProcInsertToMain do
 begin
   {Передаем значения для переменных хранимой процедуры MainInsert,
   за работу с которой у нас в программе отвечает
   компонент StoredProcInsertToMain}
   Parameters.ParamByName('@NameOfFilm').Value:=EditNameOfFilm.Text;
   Parameters.ParamByName('@GanrKod').Value:=EditGanr.Tag;
   {Обнуляем переменную}
   b:='';
   {Читаем по строке из компонента RichEdit
   и разделяем каждую строку символом конца строки
   #13}
   for i:=0 to RichEditFilmAboutLines.Count-1 do
   begin
     b:=b+RichEditFilmAbout.Lines.Strings[i]+#13;
   end;
   {Задаем значение последних двух параметров}
   Parameters.ParamByName('@FilmAbout').Value:=b;
   Parameters.ParamByName('@PathToFilm').Value:=EditPathToFilm.Text;

 end; {end with}

  {Передаем параметры на сервер в процедуру
  и просим сервер выполнить процедуру}
  DM.StoredProcInsertToMain.ExecProc;
```

```
  {Если сервер обработал то, что мы ему отправили,
  значит, он дойдет до этой строчки, и мы потверждаем
  транзакцию, иначе возникнет исключение
  см.  Except}
  DM.ADOConnection1.CommitTrans;

{Закрываем окно FormModifyData}
FormModifyData.Close;

{Сервер не смог выполнить процедуру и выдал ошибку,
вызвав исключение}
 Except
  {откатываем транзакцию}
  DM.ADOConnection1.RollbackTrans;
  {Сообщаем пользователю ситуацию}
  ShowMessage('Невозможно выполнить. Повторите.');
 end;

end;
```

Теперь переходим к форме `MainForm`, подключаем модуль `UModifyData` и для кнопки "Добавить фильм" пишем код:

```
try
 {Отключаем таймер}
 Timer1.Enabled:=false;
 {Создаем форму FormModifyData}
 FormModifyData:=TFormModifyData.Create(self);
 {Кнопку Добавить делаем доступной}
 FormModifyData.ButtonAdd.Enabled:=true;
 FormModifyData.ShowModal;
 {Этот блок выполнится, после того как пользователь
 закроет окно FormModifyData}
 finally
  {Включаем таймер}
  Timer1.Enabled:=true;
  {Запускаем событие таймера Timer,
  чтобы данные у нас обновились}
  Timer1Timer(self);
```

```
  {Переходим на последнюю запись}
  DM.QMain.Last;
 end;
```

Помещаем ADOQuery на форму FormModifyData, называем его QSelectGanr. Подключаем его к ADOConnection1, в свойство SQL пишем запрос:

```
SELECT * FROM Ganri WHERE GanrID=:Par
```

Запрос означает: выбор записи из таблицы Ganri по ее коду. Щелкаем по свойству Parameters. И убеждаемся, что свойство DataType для записи Par содержит ftInteger. Щелкаем два раза левой кнопкой мыши по компоненту QSelectGanr, в появившемся окне щелкаем правой кнопкой мыши и в выпадающем меню выбираем пункт **Add all fields**.

Для кнопки "Изменить" пишем код:

```
procedure TFormModifyData.ButtonEditClick(Sender: TObject);
{Объявляем переменные}
var
 b:variant; {Будет хранить многострочный текст, полученный из RichEdit}
 i:integer;
begin
{Проверяем, все ли необходимые поля заполнены}
If (EditNameOfFilm.Text='') or
    (EditGanr.Text='')       or
    (EditPathToFilm.Text='')
{Если нет, то выводим сообщение и выходим из
процедуры с помощью Exit}
 then
  begin
     ShowMessage('Заполнены не все поля!');
     exit;
  end;

try
{Объявляем начало транзакции}
 DM.ADOConnection1.BeginTrans;
{конструкция with}
with DM.StoredProcEditMain do begin
```

```
    {Передаем значения для переменных хранимой процедуры MainEdit,
    за работу, с которой у нас в программе отвечает
    компонент StoredProcEditMain}
    {В параметр @FilmID мы передаем код записи,
     которая подвергнется изменению}
    Parameters.ParamByName('@FilmID').Value:=DM.QMainFilmID.Value;
    Parameters.ParamByName('@NameOfFilm').Value:=EditNameOfFilm.Text;
    Parameters.ParamByName('@GanrKod').Value:=EditGanr.Tag;
    {Обнуляем переменную}
    b:='';
    {Читаем по строке из компонента RichEdit
    и разделяем каждую строку символом конца строки
    #13}
    for i:=0 to RichEditFilmAbout.Lines.Count-1 do
    begin
      b:=b+RichEditFilmAbout.Lines.Strings[i]+#13;
    end;
    {Задаем значение последних двух параметров}
    Parameters.ParamByName('@FilmAbout').Value:=b;
    Parameters.ParamByName('@PathToFilm').Value:=EditPathToFilm.Text;

 end; {end with}

   {Передаем параметры на сервер в процедуру
   и просим сервер выполнить процедуру}
DM.StoredProcEditMain.ExecProc;
   {Если сервер обработал то, что мы ему отправили,
   значит, он дойдет до этой строчки, и мы подтверждаем
   транзакцию, иначе возникнет исключение
   см. Except}
   DM.ADOConnection1.CommitTrans;

{Закрываем окно FormModifyData}
FormModifyData.Close;

{Сервер не смог выполнить процедуру и выдал ошибку,
вызвав исключение}
```

```
 Except
  {откатываем транзакцию}
  DM.ADOConnection1.RollbackTrans;
  {Сообщаем пользователю ситуацию}
  ShowMessage('Невозможно выполнить. Повторите.');
 end;

end;
```

Теперь переходим к форме `MainForm` и для кнопки "Изменить данные по фильму" пишем код:

```
try
  {Отключаем таймер}
  Timer1.Enabled:=false;
  {Создаем форму FormModifyData}
  FormModifyData:=TFormModifyData.Create(self);
  {Кнопку Изменить делаем доступной}
  FormModifyData.ButtonEdit.Enabled:=true;
  {Загружаем данные, чтобы пользователь смог их изменить}
  FormModifyData.EditNameOfFilm.Text:=DM.QMainNameOfFilm.Value;
  {Так как в таблице Main у нас храниться только код жанра, нам надо
  получить его название, чтобы отобразить пользователю. Этого
  мы достигаем с помощью запроса, передаваемого серверу компонентом
  QSelectGanr}
  FormModifyData.QSelectGanr.Parameters.ParamByName('Par').Value:=
                      DM.QMainGanrKod.Value;
  FormModifyData.QSelectGanr.Open;

  {Присваиваем двум Edit полученные код и жанр}
  FormModifyData.EditGanr.Text:=
                FormModifyData.QSelectGanrNameOfGanr.Value;
  FormModifyData.EditGanr.Tag:=
                FormModifyData.QSelectGanrGanrID.Value;

  FormModifyData.QSelectGanr.Close;
```

```
  FormModifyData.EditPathToFilm.Text:=DM.QMainPathToFilm.Value;
  {Загружаем данные в компонент RichEditFilm из компонента DBRichEdit1
  с помощью метода Assign, которые просто копирует данные из второго
  компонента в первый}
  FormModifyData.RichEditFilmAbout.Lines.
                Assign(MainForm.DBRichEdit1.Lines);

  FormModifyData.ShowModal;
{Этот блок выполнится после того, как пользователь
закроет окно FormModifyData}
finally
  {Включаем таймер}
  Timer1.Enabled:=true;
  {Запускаем событие таймера Timer,
  чтобы данные у нас обновились}
  Timer1Timer(self);
end;

end;
```

Для кнопки "Удалить фильм" пишем код:

```
procedure TMainForm.Button4Click(Sender: TObject);
begin
 {Отключаем таймер}
 Timer1.Enabled:=false;

 {Спрашиваем пользователя, правда ли он хочет удалить запись}
 if MessageDlg('Вы уверены, что хотите удалить запись?',
               mtConfirmation,[mbYes,mbNo],0)=mrYes then
 {если да, то}
 begin
  {передаем код выбранной записи
   в процедуру MainDelete}
  DM.StoredProcDeleteFromMain.Parameters.
    ParamByName('@FilmID').Value:=DM.QMainFilmID.Value;

  {Пытаемся удалить запись}
  try
```

```
    {Запускаем транзакцию}
    DM.ADOConnection1.BeginTrans;
    {Пытаемся выполнить процедуру}
    DM.StoredProcDeleteFromMain.ExecProc;

   {если не получилось, то сообщаем об этом пользователю}
   except
    ShowMessage('Удаление не прошло!'+#13+
                'Запись заблокирована, либо уже удалена!');
    {откатываем транзакцию}
    DM.ADOConnection1.RollbackTrans;
    exit;
  end;
```

Для кнопки "Выход" пишем код:

```
DM.QMain.Close;
MainForm.Close;
```

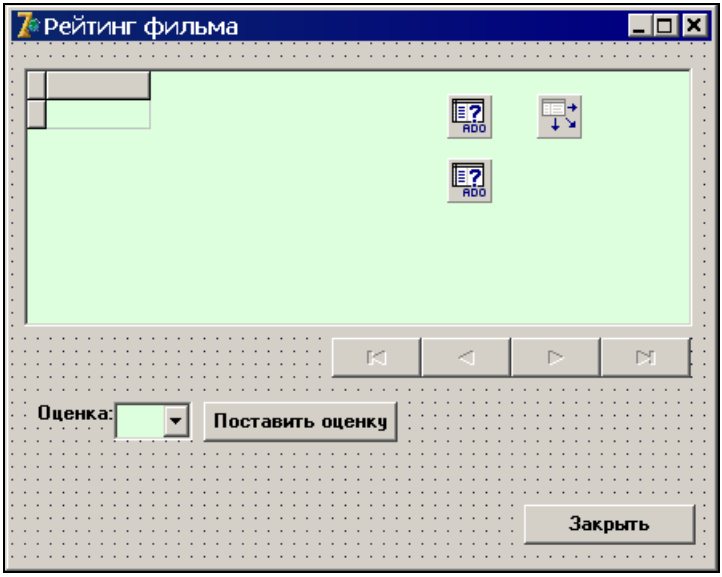


Рис. 4.69. Форма `FormReiting`

Создаем новую форму. Называем ее `FormReiting`, а модуль сохраняем под именем `UReiting`. Делаем, чтобы форма не создавалась автоматически. Рас-

положите на ней следующие компоненты: 2 штуки ADOQuery, DataSource, DBGrid, Label, ComboBox, 2 Button (рис. 4.69.

Подключаем модуль UDM. ComboBox переименовываем в ComboBoxBall. Щелкаем на свойстве Items и вносим 5 строк: 1, 2, 3, 4, 5 — это будут оценки, которые пользователь будет ставить фильму. Свойство Text делаем пустым. Компоненты ADOQuery переименовываем в QViewBall и QAddBall. Подключаем их к ADOConnection1. В свойство SQL компонента QViewBall пишем запрос:

```
SELECT UserLogin,Ball
FROM SecurityTable, Reiting
WHERE (FilmKod=:Par)
          AND
      (Reiting.UserKod=SecurityTable.UserID)
```

Запрос означает: выбрать имена пользователей из таблицы SecurityTable и оценки, поставленные ими, из таблицы Reiting для указанного фильма.

Щелкаем два раза левой кнопкой мыши по компоненту QViewBall, в появившемся окне щелкаем правой кнопкой мыши и в выпадающем меню выбираем пункт **Add all fields**. В свойство SQL компонента QAddBall пишем запрос:

```
INSERT INTO Reiting(FilmKod,UserKod,Ball)
VALUES (:ParFilmKod,:ParUserKod,:ParBall)
```

Запрос означает: добавить новую запись в таблицу Reiting и значение атрибутов взять из параметров. Код фильма мы будем брать из QMain, код пользователя — из глобальной переменной id_user, оценку — из ComboBoxBall.

В свойство DataSet компонента DataSource устанавливаем QViewBall. В свойство DataSource компонентов DBGrid и DBNavigator ставим DataSource1. Свойство ReadOnly компонента DBGrid установите в True. Для компонента DBNavigator свойство VisibleButtons установите так же как раньше, чтобы были доступны только 4 кнопки.

Переходим к форме MainForm, подключаем модуль UReiting и для кнопки "Рейтинг" фильма пишем код:

```
try
  {Выключаем таймер}
  Timer1.Enabled:=false;
  {Создаем форму}
  FormReiting:=TFormReiting.Create(self);
  {Задаем код фильма, для которого будет отображать рейтинг}
```

```
   FormReiting.QViewBall.Parameters.ParamByName('Par').Value:=
          DM.QMainFilmID.Value;
   {Выполняем запрос}
   FormReiting.QViewBall.Open;
   {Отображаем окно Рейтинг}
   FormReiting.ShowModal;
 finally
  {Включаем таймер}
  Timer1.Enabled:=true;
  {Запускаем событие таймера Timer,
  чтобы данные у нас обновились}
  Timer1Timer(self);
 end;
```

Переходим к форме `FormReiting`, подключаем модуль `UMain` и для кнопки "Поставить оценку" пишем код:

```
{Проверяем, выбрал ли пользователь оценку,
если нет, то сообщаем ему об этом
и выходим из процедуры}
if ComboBoxBall.Text='' then
 begin
    ShowMessage('Вам необходимо выбрать оценку');
    exit;
 end;

{Задаем значение параметров для добавления информации
в таблицу Reiting}
 with QAddBall.Parameters do
 begin
   ParamByName('ParFilmKod').Value:=DM.QMainFilmID.Value;
   ParamByName('ParUserKod').Value:=id_user;
   ParamByName('ParBall').Value:=StrToInt(ComboBoxBall.Text);
 end;

 try
  {Объявляем начало транзакции}
   DM.ADOConnection1.BeginTrans;
   {Выполняем запрос на добавление записи}
```

```
    QAddBall.ExecSQL;
    {Если сервер обработал то, что мы ему отправили,
    значит он дойдет до этой строчки и мы подтверждаем
    транзакцию, иначе возникнет исключение
    см.  Except}
    DM.ADOConnection1.CommitTrans;

   {Сервер не смог выполнить процедуру и выдал ошибку,
    вызвав исключение}
  Except
    {откатываем транзакцию}
    DM.ADOConnection1.RollbackTrans;
    {Сообщаем пользователю ситуацию}
    ShowMessage('Невозможно выполнить. Повторите.');
  end;

{Обновляем набор данных}
QViewBall.Close;
    FormReiting.QViewBall.Parameters.ParamByName('Par').Value:=
           DM.QMainFilmID.Value;
  QViewBall.Open;
```

Для события `OnClose` формы `FormReiting` пишем код:

```
QViewBall.Close;
```

Для кнопки "Закрыть" пишем код:

```
FormReiting.Close;
```

Подходим к финалу. Нам осталось только закодировать форму для просмотра фильмов. Создаем новую форму. Называем ее `FormViewFilm`, а модуль сохраняем под именем `UViewFilm`. Делаем, чтобы форма не создавалась автоматически. Расположите на ней компонент `MediaPlayer` (рис. 4.70).

Рис. 4.70. Форма `FormViewFilm`

Для события `OnClose` напишите код:

```
{Закрытие сеанса}
MediaPlayer1.Close;
```

Для кнопки "Просмотр" формы `MainForm` напишите код:

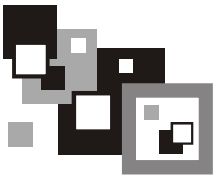
```
try
  FormViewFilm:=TFormViewFilm.Create(self);

  {Задаем фильм, который будем просматривать}
  FormViewFilm.MediaPlayer1.FileName:=DM.QMainPathToFilm.AsString;
  FormViewFilm.MediaPlayer1.Open;

  FormViewFilm.ShowModal;

 except
  ShowMessage('Невозможно просмотреть!');
 end;
```

На компакт-диске в каталоге ГЛАВА 4\SOURCE можно увидеть пример данной программы.

В каталоге ГЛАВА 4\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 4\VIDEO можно посмотреть видеоролики, посвященные созданию разрабатываемого приложения.

Глава 5



FireBird

5.1. Вступление

FireBird — это клон InterBase. То есть принцип работы с ним точно такой же, как с InterBase. Обе СУБД могут работать через одну библиотеку GDL32.DLL, только вот "подружить" их на одном компьютере не всегда получается; чтобы это сделать, необходимо останавливать одну из СУБД или удалять ее из системы.

В этой главе рассмотрим написание программы Библиотекарь. Данная программа будет вести учет книг в библиотеке и состоять из двух модулей. Первый предназначен для администратора, который будет добавлять книги в базу, заводить абонементы, выдавать книги и т. д. Второй предназначен исключительно для читателей библиотеки, которые смогут с помощью него просматривать свою статистику по абонементу, а также узнавать какие книги есть в библиотеке в настоящий момент времени.

5.2. Установка FireBird

Замечание

В данной главе рассматривается установка FireBird SuperServer версии 1.5, работа с другими версиями серьезно ничем не отличается.

Если на компьютере запущена СУБД InterBase, то ее обязательно следует остановить. Для этого нужно войти в **Панель управления** операционной системы Windows. Произвести двойной щелчок левой кнопкой мыши на пиктограмме InterBase Manager. Появится окно **InterBase Manager** (рис. 5.1).

Нажимаем кнопку **Stop**. Сервер остановится и появится надпись **Stopped** красного цвета, которая указывает на то, что работа сервера остановлена (рис. 5.2).

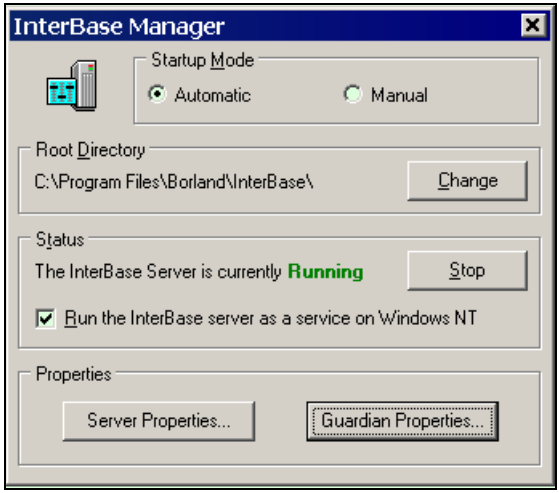


Рис. 5.1. Окно **InterBase Manager**

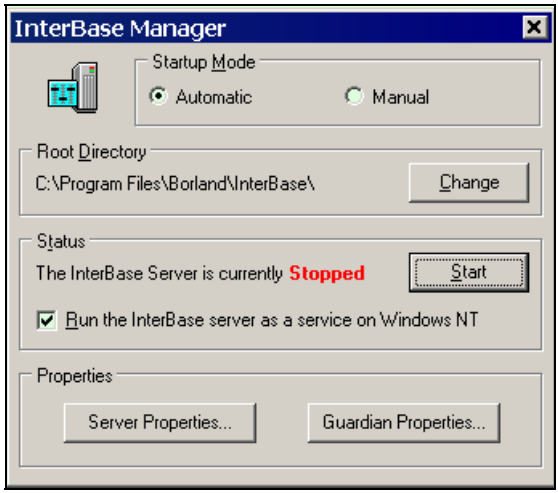


Рис. 5.2. Окно **InterBase Manager** при остановленной работе сервера

Запускаем файл Firebird-1.5.0.4306-Win32.exe, который предназначен для установки СУБД. После запуска файла появляется окно приветствия **Welcome to the Firebird Database Server 1.5 Setup Wizard** (рис. 5.3).

В этом окне нам рекомендуют закрыть все открытые приложения (это не является обязательным условием). Нажимаем **Next**. Появляется окно **License Agreement**, которое содержит лицензионное соглашение (рис. 5.4).

Рис. 5.3. Шаг **Welcome** мастера установки

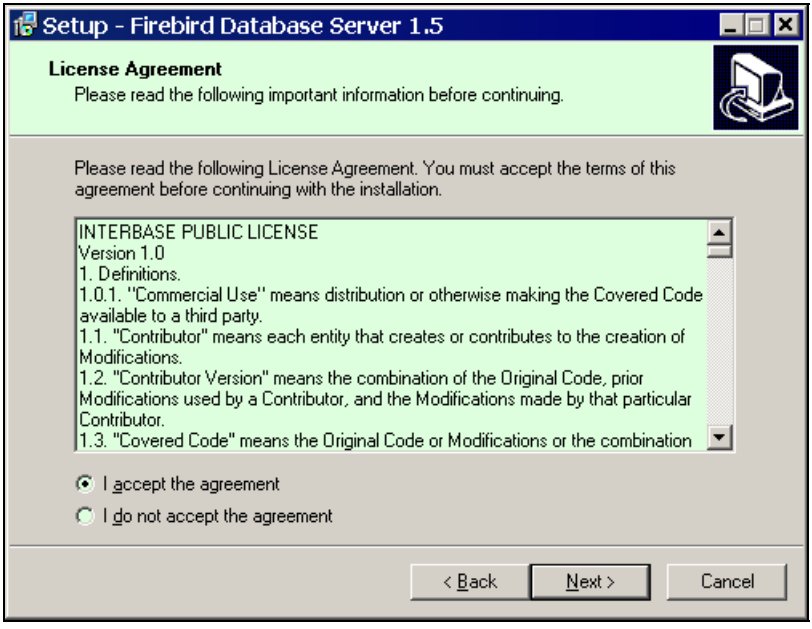


Рис. 5.4. Шаг **License Agreement** мастера установки

Устанавливаем переключатель **I accept the agreement**. Нажимаем **Next**. Появляется окно с дополнительной информацией **Information**, где мы также нажимаем кнопку **Next**. В следующем окне (рис. 5.5) нам предлагается выбрать путь установки. Рекомендуется оставить его по умолчанию.

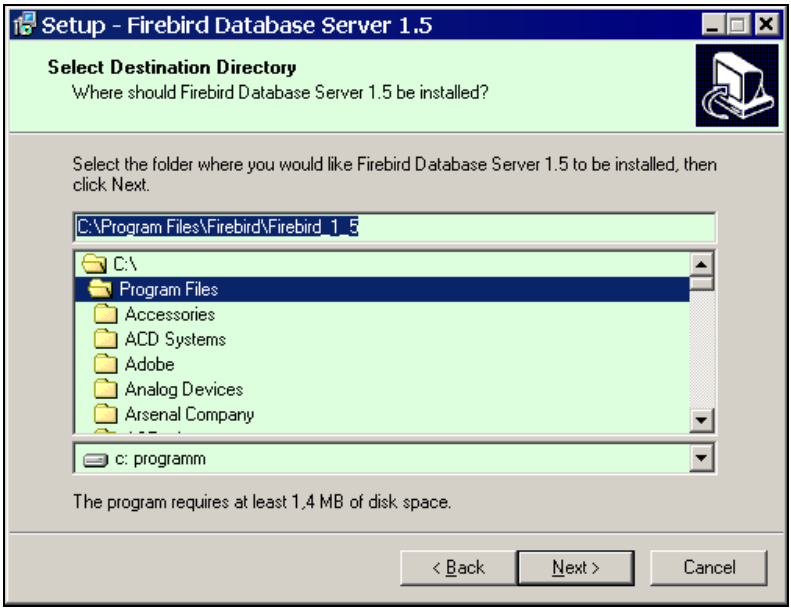


Рис. 5.5. Шаг **Select Destination Directory** мастера установки

Нажимаем **Next**. Появляется окно **Select Components**, где предлагается выбрать тип установки (рис. 5.6).

Нам необходима полная инсталляция, поэтому оставляем **Full installation of Super Server and development tools**.

Замечание

Если производится установка FireBird на клиентском месте, то необходимо выбрать пункт **Minimum client install — no server, no tools** из выпадающего списка.

Нажимаем **Next**. Появляется окно **Select Start Menu Folder**. Здесь предлагается выбрать группу в меню **Программы**, в которой будут созданы ярлыки приложения (рис. 5.7). Если установить флаг **Don't create any icons**, то ярлыки не будут созданы.

Оставляем группу **Firebird_1_5**, как и предложено по умолчанию. Нажимаем **Next**. Появляется окно **Select Additional Tasks**, где предлагается выбрать дополнительные параметры инсталляции (рис. 5.8).

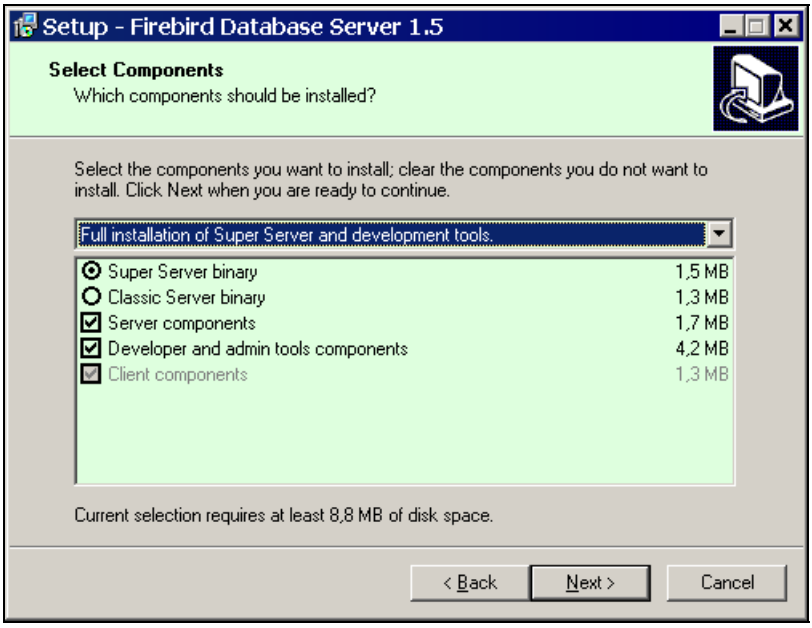


Рис. 5.6. Шаг **Select Components** мастера установки

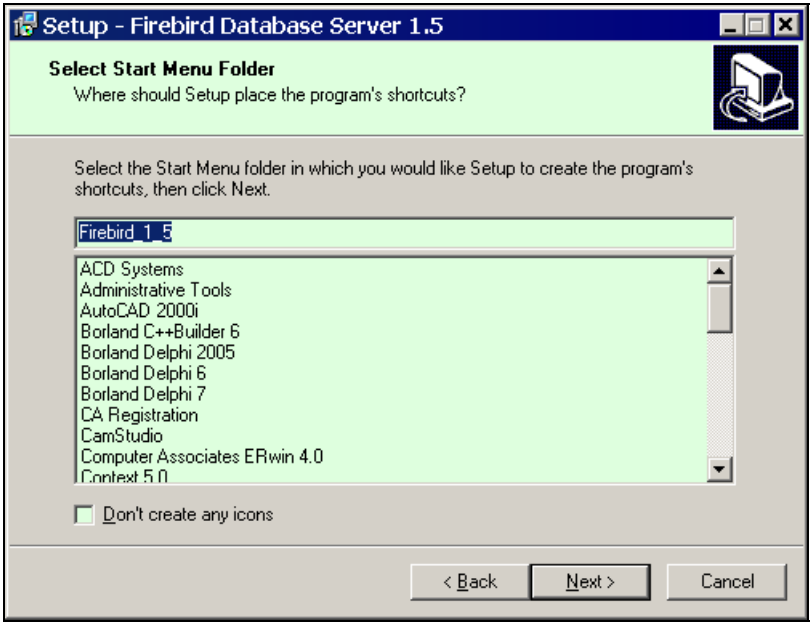


Рис. 5.7. Шаг **Select Start Menu Folders** мастера установки

Рис. 5.8. Шаг **Select Additional Tasks** мастера установки

Интерес представляют переключатели группы **Run Firebird server as**:

- **Run as an Application?** — сервер будет запускаться как отдельное приложение и его иконка будет висеть в трее;
- **Run as a Service?** — FireBird будет работать как служба Windows и его можно будет увидеть, выбрав **Пуск\Панель управления\Администрирование\Службы**.

Устанавливаем флаг **Start Firebird automatically everytime you boot up**, что приведет к тому, что FireBird будет стартовать автоматически при загрузке Windows. Также необходимо установить переключатель **Copy Firebird client library to <system> directory** *(см. замечание ниже)*.

Замечание

При установке FireBird на клиентские рабочие места (если был выбран тип инсталляции **Minimum client install**), вид окна **Select Additional Tasks** будет другим (рис. 5.9).

Здесь необходимо проследить, чтобы переключатель **Copy Firebird client library to <system> directory** был установлен. Таким образом, библиотеки для работы с FireBird-сервером скопируются в системную папку Windows, и не будет проблем с запуском клиентского приложения (проблемы могут заключаться в том, что необходимая библиотека может быть не найдена).

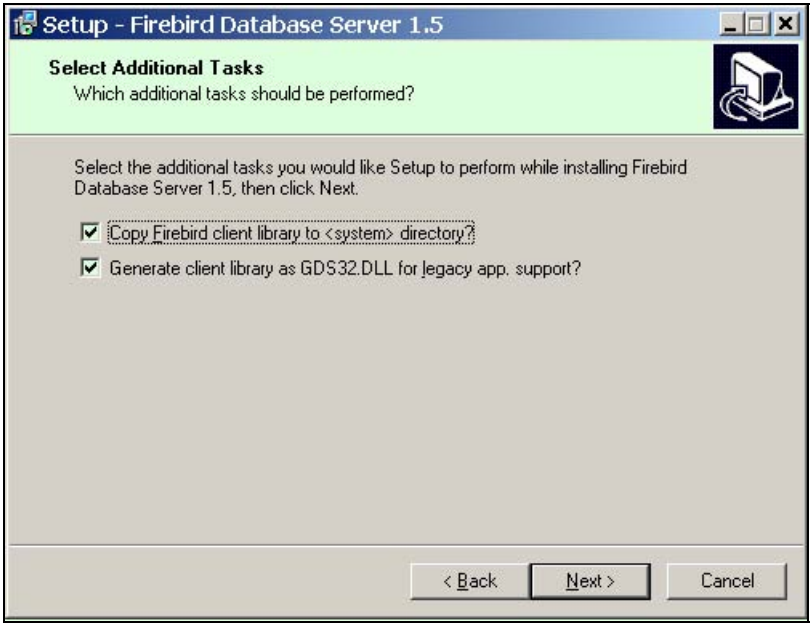


Рис. 5.9. Шаг **Select Additional Tasks** при выборе типа инсталляции **Minimum client install**

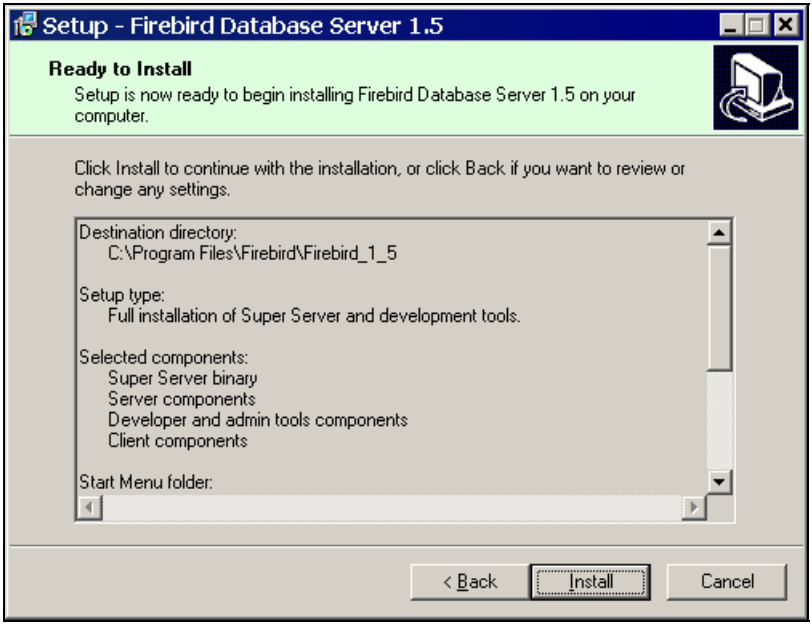


Рис. 5.10. Шаг **Ready to Install** мастера установки

Нажимаем **Next**. Появляется окно, где отображаются выбранные параметры на предыдущих этапах и предлагается нажать кнопку **Install**, чтобы начать установку Firebird (рис. 5.10).

После того как установка будет завершена, появится окно **Information** с дополнительной информацией. Нажимаем **Next**. Появляется окно **Completing the Firebird Database Server 1.5 Setup Wizard**, где нам предлагается перезагрузить компьютер (рис. 5.11).

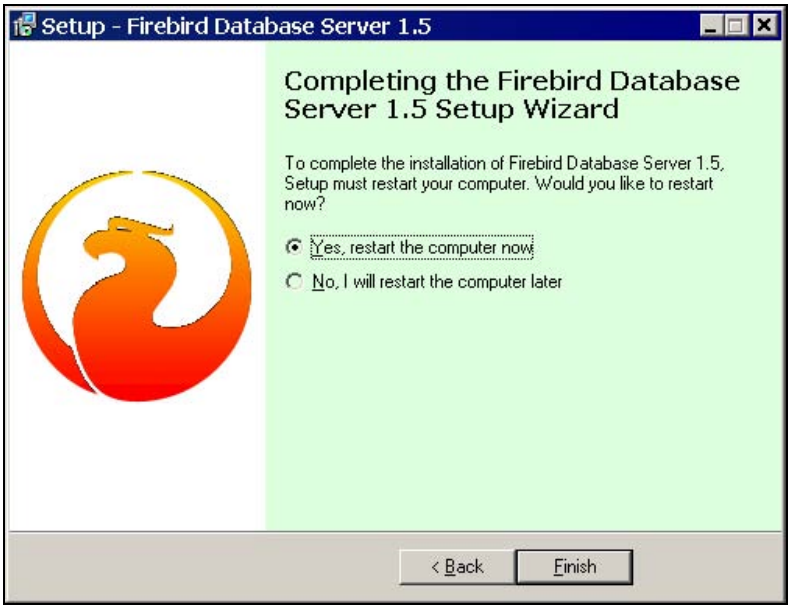


Рис. 5.11. Завершающий шаг мастера установки

Оставляем установленным переключатель **Yes, restart the computer now** и нажимаем кнопку **Finish**. После того как компьютер перезагрузится, нажимаем **Пуск\Панель управления**. Можно будет увидеть, что появился ярлык менеджера Firebird — **Firebird 1.5 Server Manager** (рис. 5.12).

Производим двойной щелчок левой кнопкой мыши по ярлыку. Появляется менеджер Firebird — окно **Firebird Server Control** (рис. 5.13).

В этом окне можно остановить сервер — кнопка **Stop**. Если сервер отключен, то можно запустить его — кнопка **Start** (она будет располагаться на месте кнопки **Stop**). Также можно поменять способ работы сервера, для этого предназначены переключатели группы элементов **Run**:

- **As a Service** — сервер работает как служба Windows;
- **As and application** — сервер работает как приложение.

Рис. 5.12. Панель управления

Рис. 5.13. Firebird Server Control

С помощью группы элементов **Start** можно поменять тип запуска сервера:

- **Automatically** — сервер стартует автоматически после загрузки Windows;
- **Manually** — ручной запуск, если установлен этот переключатель, то надо будет войти в менеджер Firebird, чтобы запустить сервер.

5.3. Постановка задачи

Как уже говорилось в начале главы, мы будем писать программу Библиотекарь, которая будет обладать следующими возможностями:

- вести учет книг библиотеки;
- хранить данные по читателям и их абонементам;
- учитывать движения книг по пути библиотека — читатель — библиотека;
- отслеживать книги, срок нахождения которых на руках у читателя истек;
- предоставлять читателю интерфейс для просмотра данных по абонементу, а также производить поиск необходимых ему книг.

Для реализации данной задачи нам потребуется база данных, состоящая из таблиц, описанных в табл. 5.1—5.4.

Таблица 5.1. *Таблица "Читатели" (Reader)*

Имя атрибута	Описание	Тип данных
ReaderID	Поле-счетчик	Счетчик
FIO	ФИО читателя	Текст(100)
Pasport	Номер и серия паспорта	Текст(10) Текст, потому что в серии паспорта содержатся символы

Таблица 5.2. *Таблица "Абонементы" (Abonement)*

Название	Описание	Тип данных
AbonementID	Поле-счетчик	Счетчик
ReaderKod	Код читателя, которому принадлежит абонемент	Числовой
DateOfEnd	Дата окончания абонемента	Дата
PriznakOfEnd	Признак действительности абонемента, если установлен, то абонемент недействителен	Логический

Таблица 5.3. *Таблица "Книги" (Book)*

Название	Описание	Тип данных
BookID	Поле-счетчик	Счетчик
NameOfBook	Название книги	Текстовый(100)
Shifr	Шифр книги	Числовой
BookAbout	Краткое содержание	Текстовый(1000)
OnHand	Признак нахождения книги у читателя, если установлен, то книга у читателя	Логический

Таблица 5.4. *Таблица "Движение книг" (Move)*

Название	Описание	Тип данных
MoveID	Поле-счетчик	Счетчик
BookKod	Код книги	Числовой
AbonementKod	Код абонемента	Числовой
DateOfReturn	Дата возврата книги	Дата
PriznakOFReturn	Признак возврата книги, если установлен, то книга была возвращена, а запись представляет ценность как архивная информация	Логический

5.4. Разработка базы данных в FireBird

FireBird не имеет собственной визуальной среды разработки, как InterBase или Microsoft SQL Server. Поэтому для создания базы данных необходимо пользоваться дополнительными инструментами. Мы будем использовать *IBExpert*. Данный выбор сделан из-за того, что эта программа проста в обращении, достаточно популярна, а также совершенно бесплатна (рис. 5.14).

5.5. Работа в IBExpert

Создаем новую папку для проекта. У меня путь к папке — C:\Library. Запускаем файл IBExpert.EXE. Выбираем пункт главного меню **Database\Create Database** (создание базы данных) — рис. 5.15.

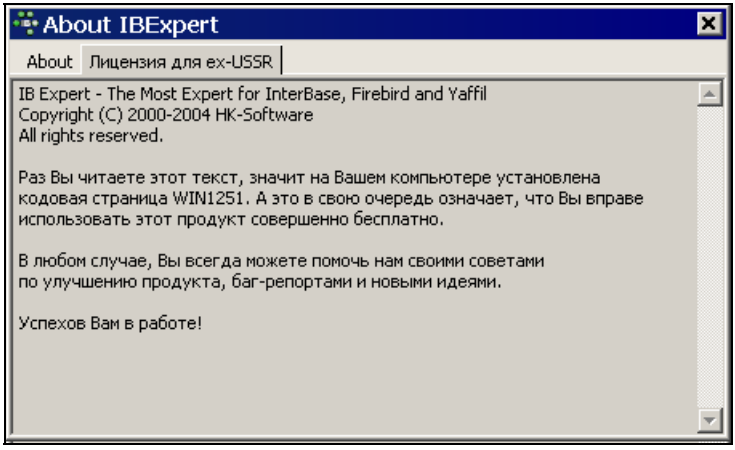


Рис. 5.14. Окно **About IBExpert**

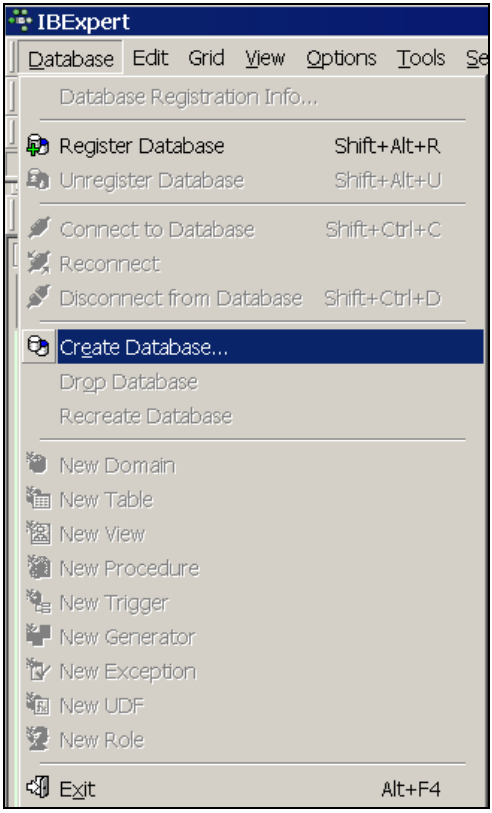


Рис. 5.15. Пункт главного меню **Database\Create Database**

На экране появится окно **Create Database** (рис. 5.16).

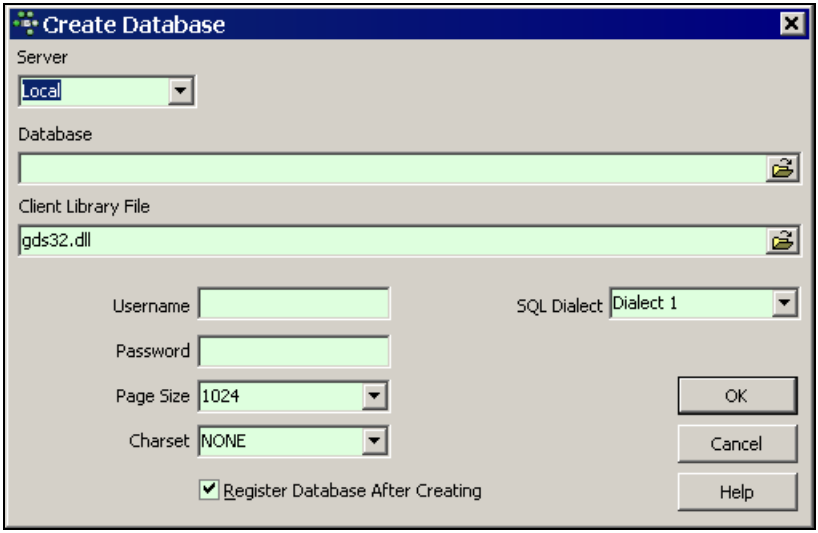


Рис. 5.16. Окно **Create Database**

Так как мы будем создавать базу на своем компьютере, выбираем **Local** в выпадающем списке **Server**. Если хочется поэкспериментировать и создать базу на удаленном сервере, если он конечно у вас есть, то выбирайте **Remote** (дополнительно надо будет выбрать имя сервера и протокол связи).

В поле **Database** выбираем имя файла будущей базы, щелкаем на кнопке с изображением открывающейся папки, появится диалоговое окно **Открыть** (Open), как показано на рис. 5.17, где необходимо выбрать:

- путь файла базы данных — поле **Папка** (у меня C:\Library);
- имя файла базы данных — поле **Имя файла** (у меня Library);
- тип файла (для Firebird нужно выбрать FDB) — поле **Тип файлов**.

Нажимаем кнопку **Открыть**. В поле **UserName** вводим SYSDBA, в поле **Password** вводим masterkey — это имя и пароль администратора (суперпользователя) по умолчанию. В выпадающем списке **Charset** выбираем **WIN1251** — это кодировка, с которой будет работать база (рис. 5.18).

Замечание

В поле **SQL Dialect** необходимо выбрать **Dialect 3**, в противном случае будут проблемы при разработке клиентского приложения. У меня, например, в Delphi не было видно хранимых процедур, хотя реально они были созданы. Очень

важно этот выбор произвести на этапе создания базы данных, иначе потом придется переделывать некоторые элементы базы.

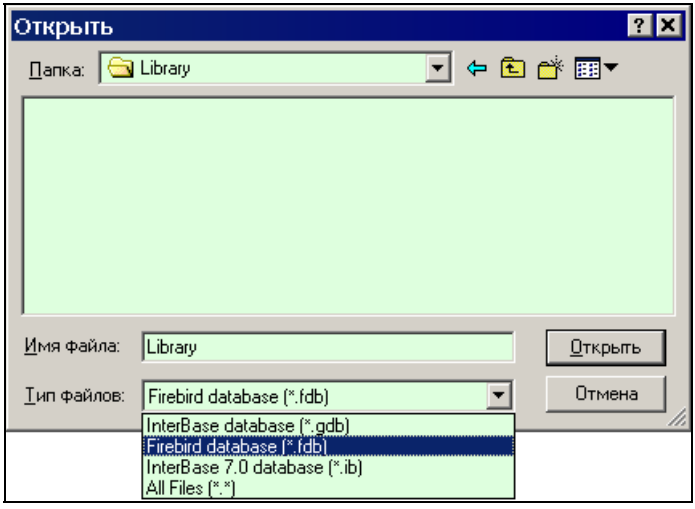


Рис. 5.17. Диалоговое окно **Открыть**

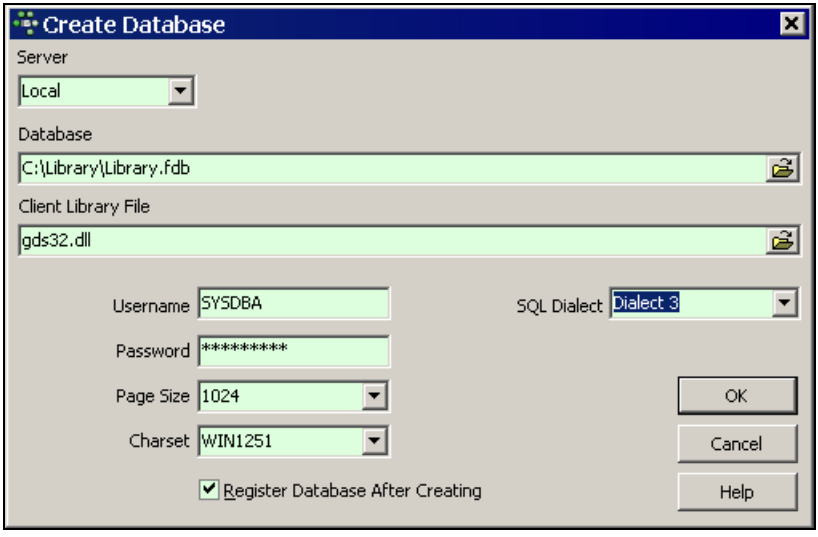


Рис. 5.18. Параметры создаваемой базы данных

Нажимаем **OK** — база создана. Появляется окно **Database Registration**. IBExpert предлагает зарегистрировать базу, иначе работать мы с ней не смо-

жем. В поле **Server Version** выбираем **FireBird 1.5**, чтобы IBExpert "знал", с какой СУБД будет вестись работа. Нажимаем кнопку **Register** (рис. 5.19).

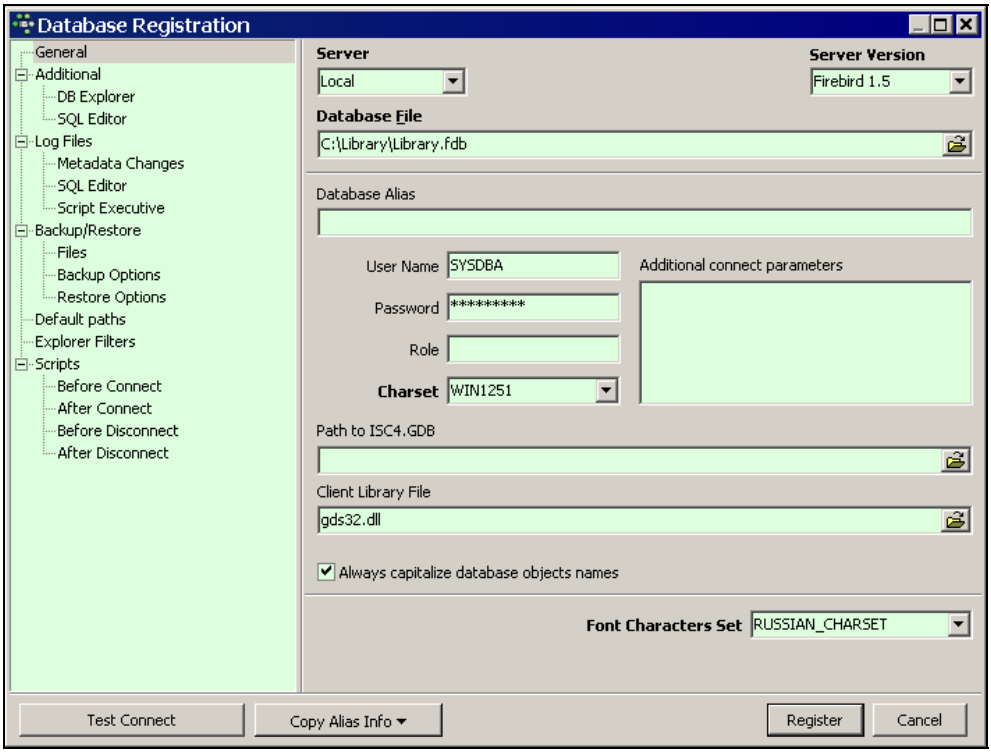


Рис. 5.19. Регистрация базы данных

После того как база зарегистрирована, она отобразится в **Database Explorer** (данное окно находится в левой части экрана) — рис. 5.20.

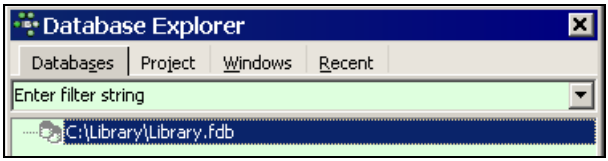


Рис. 5.20. Вкладка **Databases** окна **Database Explorer**

Производим двойной щелчок левой кнопкой мыши на имени зарегистрированной базы, после чего в окне **Database Explorer** отобразятся все элементы, доступные в базе данных (рис. 5.21).

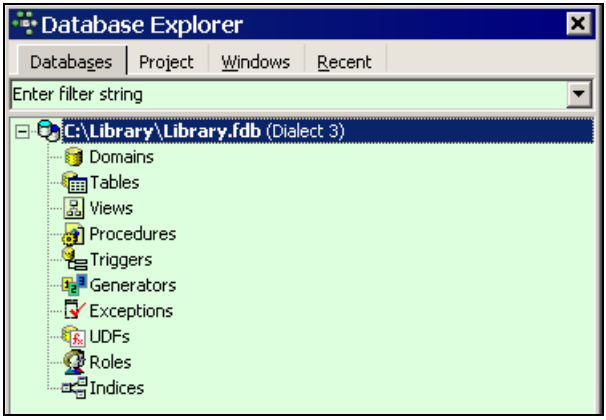


Рис. 5.21. Все элементы вкладки **Databases** окна **Database Explorer**

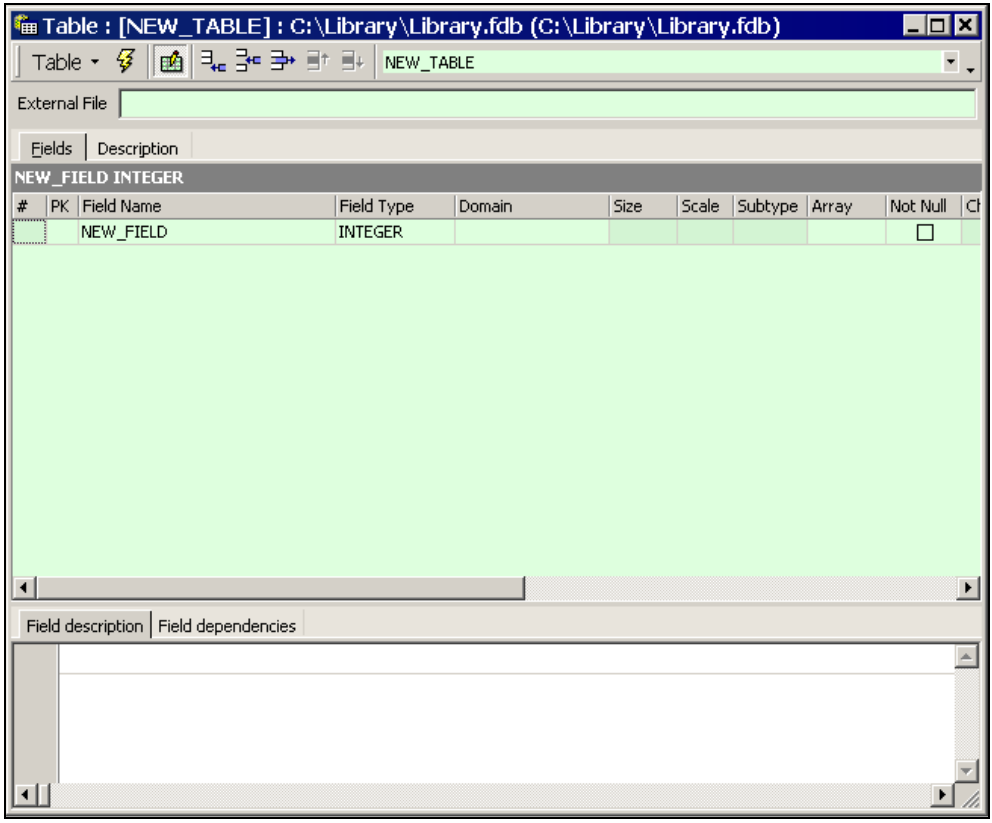


Рис. 5.22. Окно **Table**, предназначенное для создания таблицы

Приступим к созданию таблиц. Первым делом создадим таблицу "Читатели" (Reader). Щелкаем правой кнопкой мыши на пиктограмме **Tables**, в выпадающем меню выбираем пункт **NewTable**, после чего появится окно **Table**, предназначенное для создания таблицы (рис. 5.22).

Первым делом меняем имя будущей таблицы. По умолчанию она называется New_Table, мы изменим на Reader (рис. 5.23).

Рис. 5.23. Окно смены имени таблицы

В окне создания таблицы (см. рис. 5.22) есть сетка, которая содержит поля таблицы. Щелкаем пока на единственной записи левой кнопкой мыши и меняем ее название с New_Field на ReaderID, также устанавливаем тип Integer для нового атрибута (поле Field Type). Для более удобного выбора типа атрибута можно воспользоваться выпадающим списком (рис. 5.24).

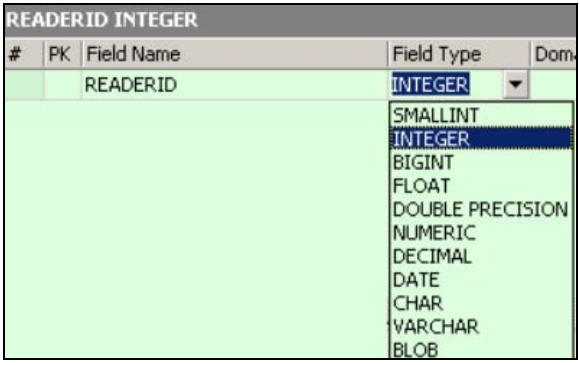


Рис. 5.24. Выпадающий список типов атрибутов

Для создания нового атрибута можно нажать кнопку добавления нового атрибута (**Add field**), расположенную в окне создания таблицы, либо стрелку вниз на клавиатуре. Необходимо добавить в таблицу Reader остальные атрибуты, как показано на рис. 5.25.

Нажимаем <Ctrl>+<F9> или кнопку **Compile** (пиктограмма с изображением молнии). Появится окно **Creating Table READER** (рис. 5.26).

Замечание

С помощью таких окон вы можете запросто выучить весь SQL, да к тому же на примерах (смотрите нижнюю часть окна на рис. 5.26).

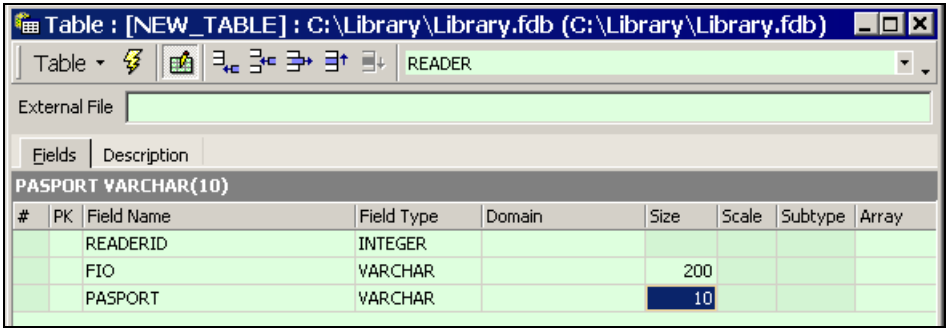


Рис. 5.25. Атрибуты таблицы **READER**

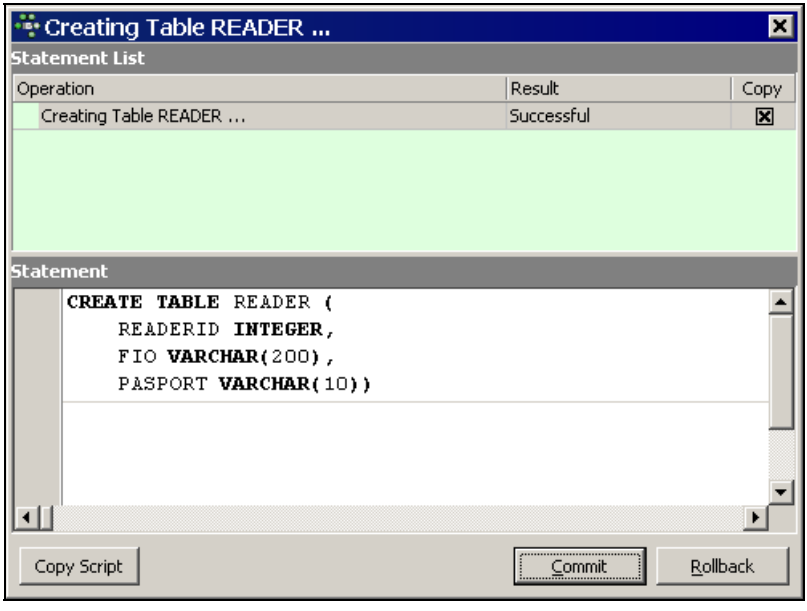


Рис. 5.26. Листинг SQL-запроса создания таблицы

В этом окне нам сообщают о том, что создается таблица Reader. В нижней части окна представлен листинг SQL-запроса для производимой операции. Нажимаем кнопку **Commit**. После этого все атрибуты таблицы пронумеруются и появятся дополнительные закладки (рис. 5.27).

Приступим к созданию генератора (счетчика). Производим двойной щелчок левой кнопкой мыши на атрибуте ReaderID. Появляется окно **Edit field READERID**. В нем нам надо установить флаг **Not NULL**, так как данный атрибут будет первичным ключом (как создавать первичный ключ рассмот-

рим чуть позже), а, как известно, первичный ключ может хранить только уникальные значения (рис. 5.28).

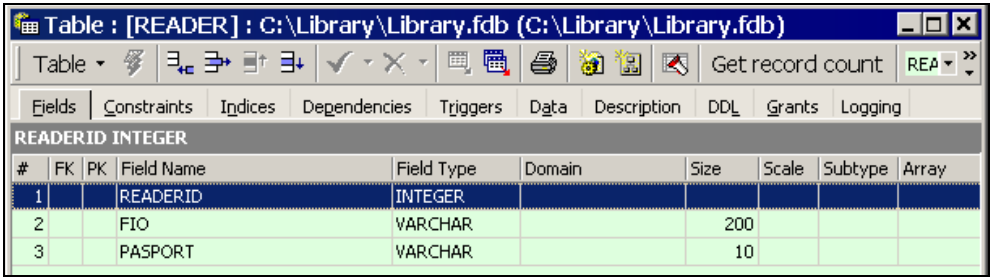


Рис. 5.27. Вид вкладки **Fields** после сохранения

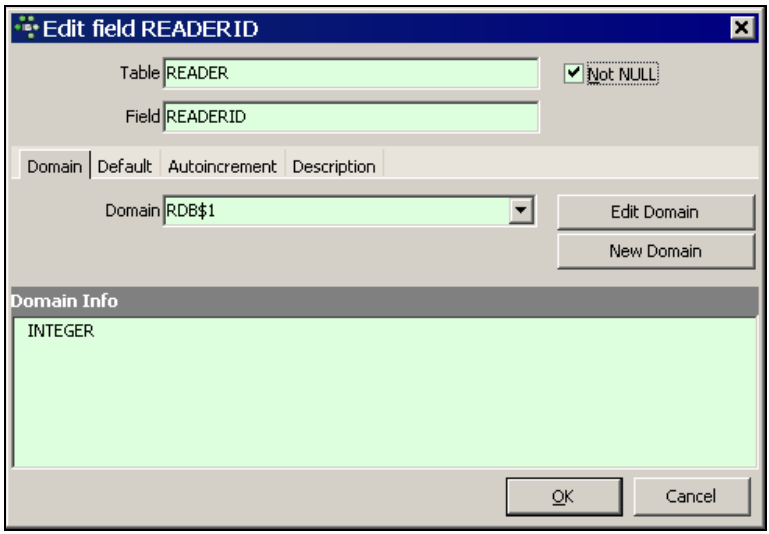


Рис. 5.28. Вкладка **Domain** окна редактирования атрибута **READERID**

Далее, в этом же окне, переходим на вкладку **Autoincrement**, ниже появятся дополнительные вкладки, они для нас и будут представлять интерес (рис. 5.29).

Устанавливаем флаг **Create generator** (Создать генератор), доступными становятся поля:

- **Generator Name** — имя генератора, оставляем предложенное программой название: GEN_READER_ID;
- **Initial Name** — начальное значение генератора, оставляем 0.

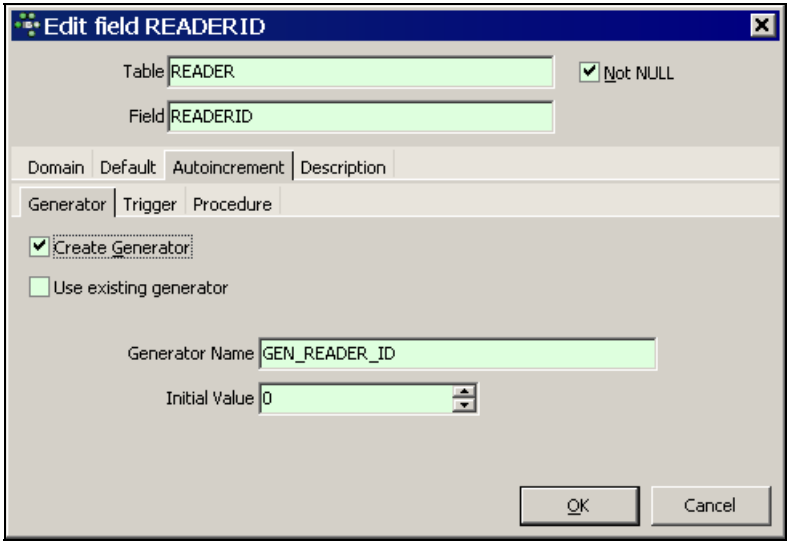


Рис. 5.29. Вкладка **Generator** окна редактирования атрибута **READERID**

Делаем активной вкладку **Trigger** и устанавливаем флаг **Create Trigger**. Окно изменит свой вид, в нем появится SQL-команда на создание триггера (рис. 5.30).

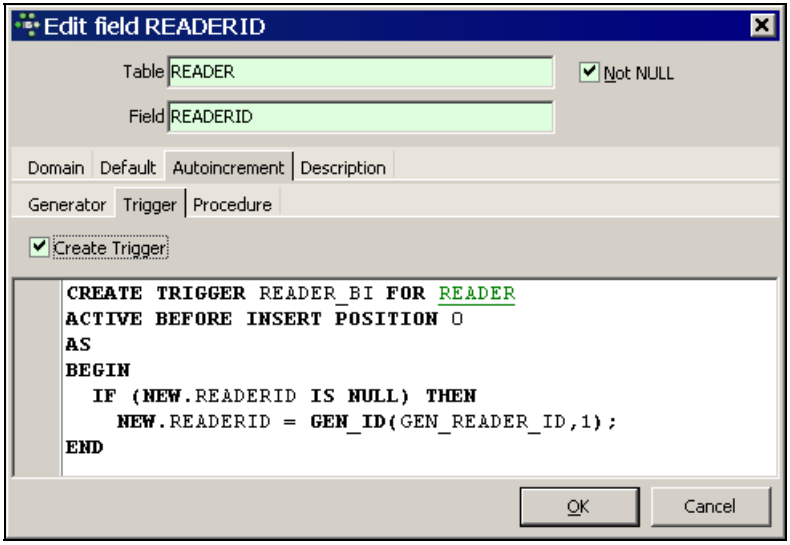


Рис. 5.30. Вкладка **Trigger** окна редактирования атрибута **READERID**

Мы только что обеспечили, что при создании новой записи номер ей будет присваиваться автоматически. Выполнять данное действие будет триггер, который будет срабатывать перед вставкой новой записи. Нажимаем **OK**. Появится окно **Changing field** для подтверждения внесенных изменений. В верхней части этого окна (**Statement List**) перечисляется список внесенных изменений (например, **Altering Field** — изменения атрибута; **Creating Generator GEN_READER_ID** — создание генератора с именем GEN_READER_ID), в нижней части (**Statement**) расположены SQL-команды для выполнения операции (рис. 5.31).

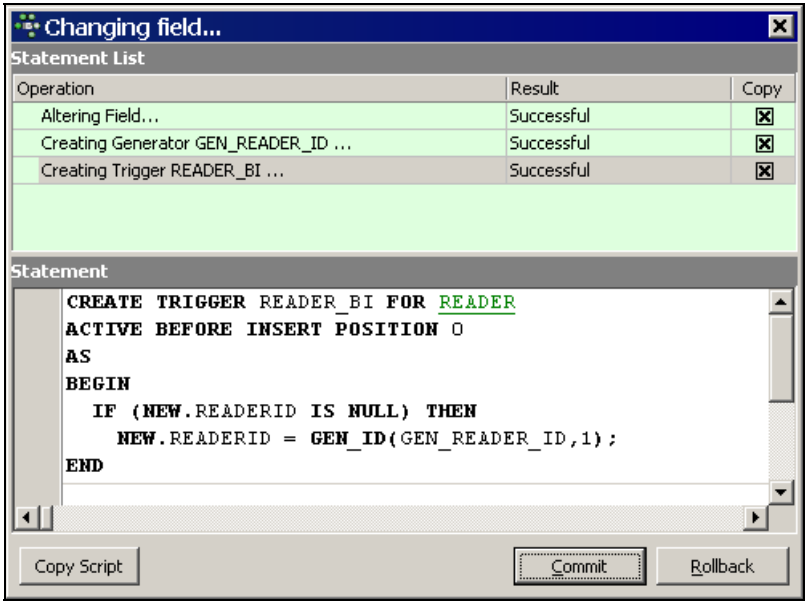


Рис. 5.31. SQL-команды для выполнения операции создания триггера

Нажимаем **Commit**.

Теперь нам необходимо создавать первичный ключ. Делаем активной вкладку **Constraints** (рис. 5.32).

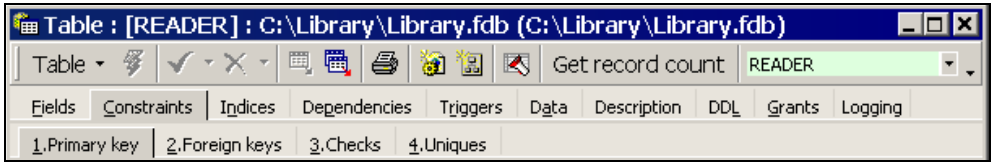


Рис. 5.32. Вид вкладки **Constraints**

Появляются дополнительные вкладки. Щелкаем правой кнопкой мыши в свободном пространстве окна **Table** и в выпадающем меню выбираем пункт **New primary key** (рис. 5.33).

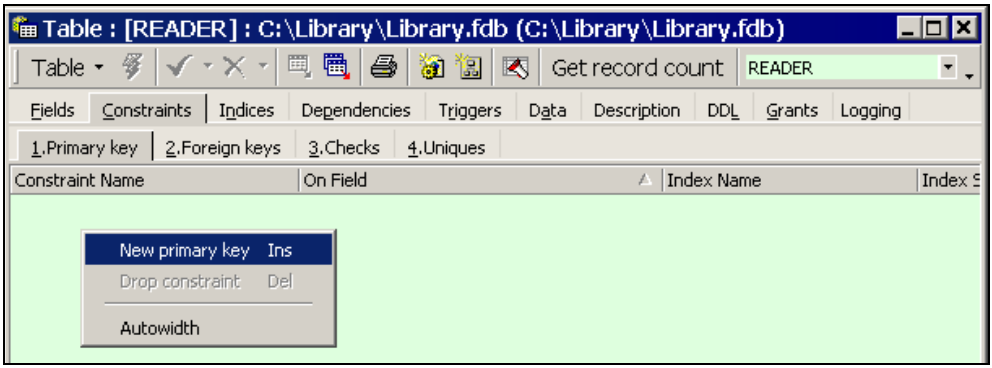


Рис. 5.33. Пункт **New primary key**

Появится новая запись, отвечающая за первичный ключ. Автоматически первичному ключу присвоится имя — PK_READER (рис. 5.34).

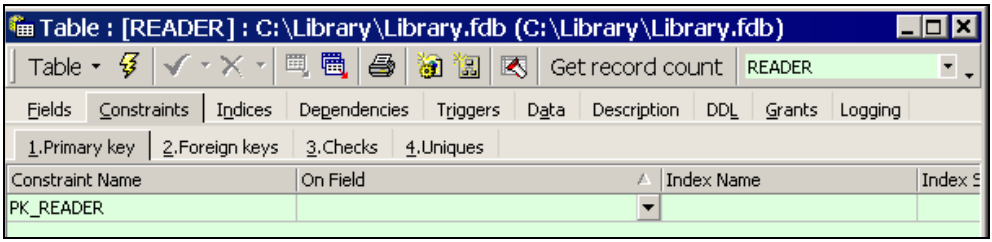


Рис. 5.34. Выбор имени первичного ключа

Теперь необходимо установить, для какого атрибута создается первичный ключ. Щелкаем левой кнопкой мыши в поле **On Field** и устанавливаем атрибут `ReaderID` (рис. 5.35).

Нажимаем <Ctr>+<F9> или кнопку **Compile** (пиктограмма с изображением молнии). Появится окно **Compiling Constraints**, в котором отображаются SQL-команды на создание первичного ключа, а также комментарии по поводу запроса (**Statement List**) (рис. 5.36).

Нажимаем кнопку **Close**. Делаем активной вкладку **Field** и видим, что в таблице Reader появился первичный ключ (запись ReaderID с изображением ключа) (рис. 5.37).

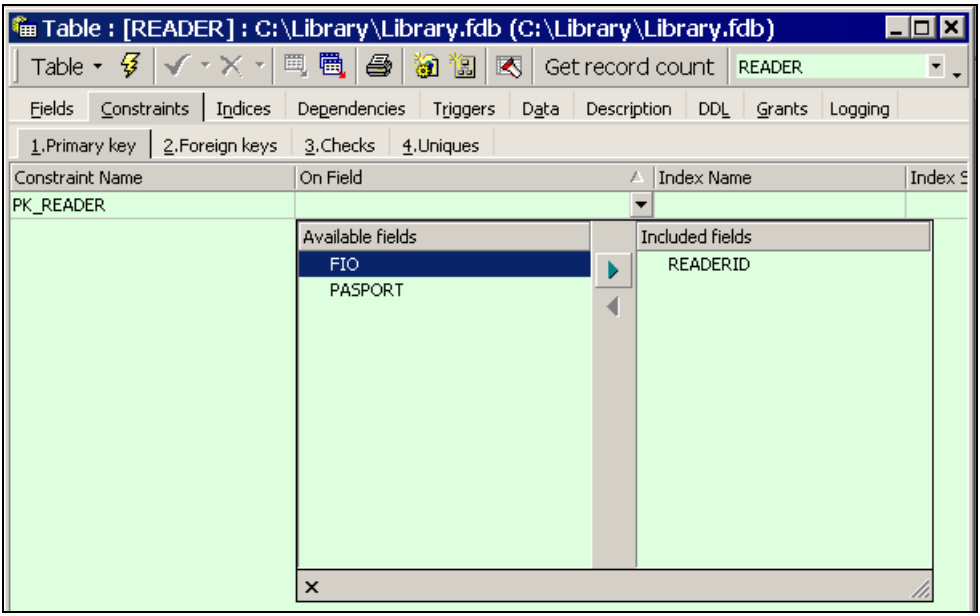


Рис. 5.35. Доступные поля для создания первичного ключа

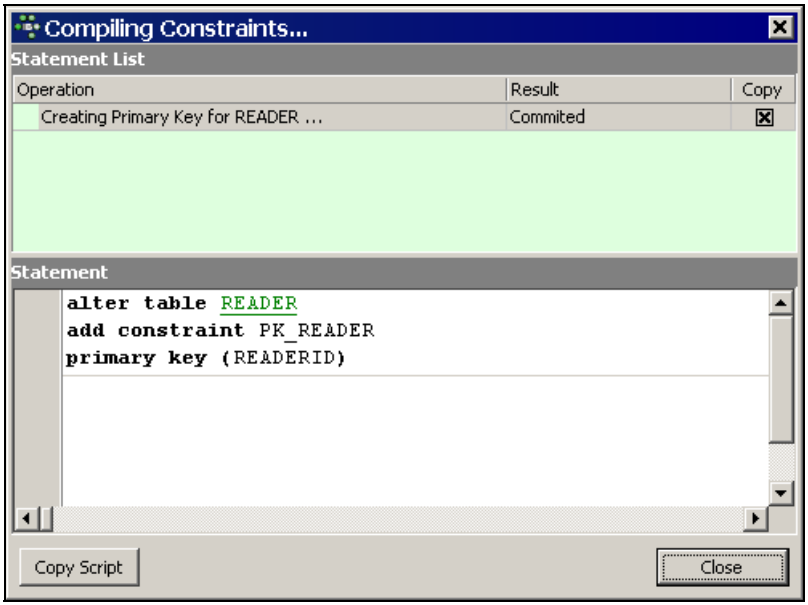


Рис. 5.36. SQL-команды создания первичного ключа

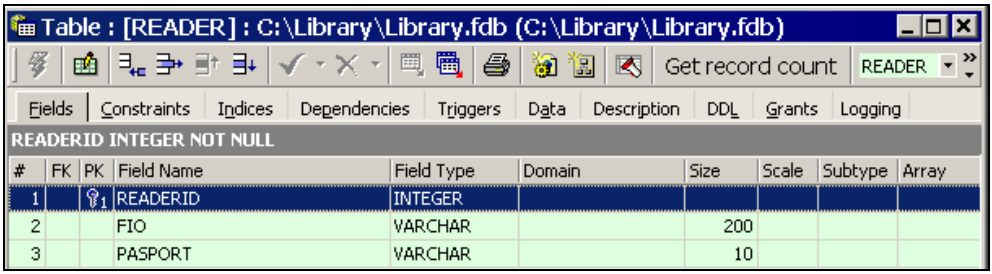


Рис. 5.37. Результат создания первичного ключа

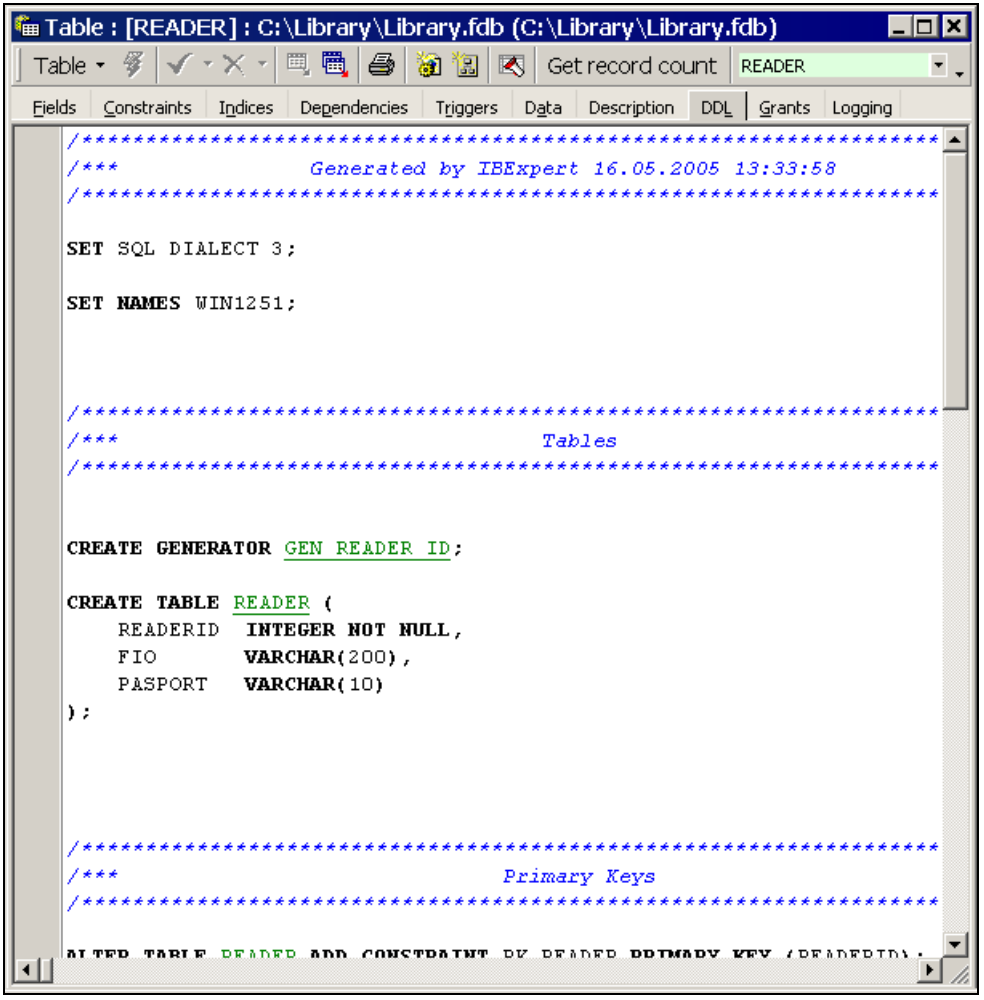


Рис. 5.38. Полный текст SQL-запроса на создание таблицы Reader

Интерес представляет вкладка **DDL**, где можно посмотреть полный текст SQL-запроса на создание таблицы Reader и ее составляющих, таких как триггеры, генераторы и других (все запросы даны с комментариями на английском языке) — рис. 5.38.

Для того чтобы внести данные в таблицу, необходимо сделать активной вкладку **Data** (рис. 5.39).

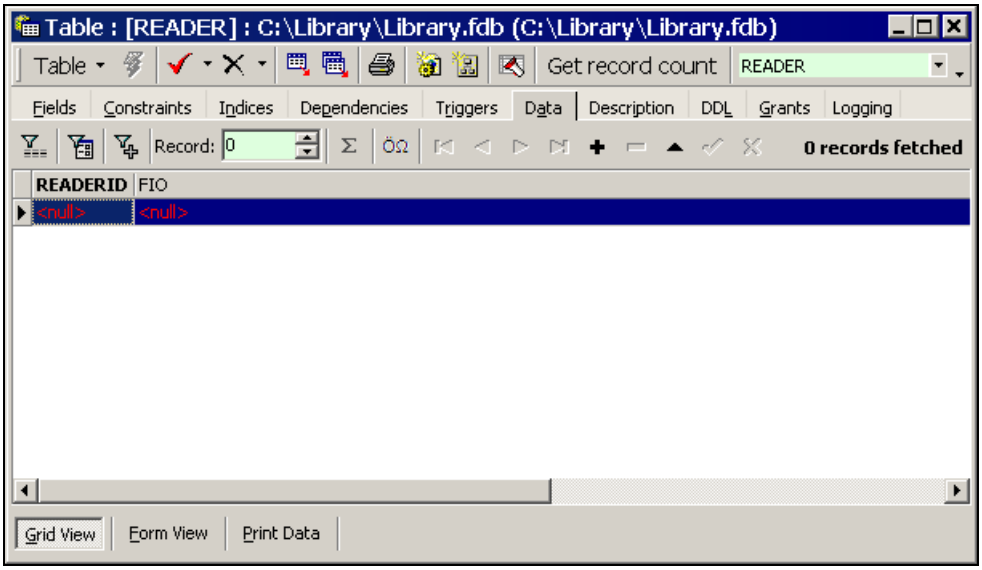


Рис. 5.39. Активная вкладка **Data** окна создания таблицы

Давайте добавим в таблицу Reader записи, как в табл. 5.5.

Таблица 5.5

READERID	FIO	PASPORT
Заполняется автоматически	Иванов В. В.	1B2345CA
Заполняется автоматически	Сидоров К. Р.	FFF456TR8
Заполняется автоматически	Миронов П. О.	678RR22UY

Замечание

При добавлении данных можно воспользоваться навигатором, который аналогичен компоненту `DBNavigator` в Delphi (рис. 5.40).

Рис. 5.40. Компонент `DBNavigator`

На рис. 5.41 представлено окно **Table** после добавления данных.

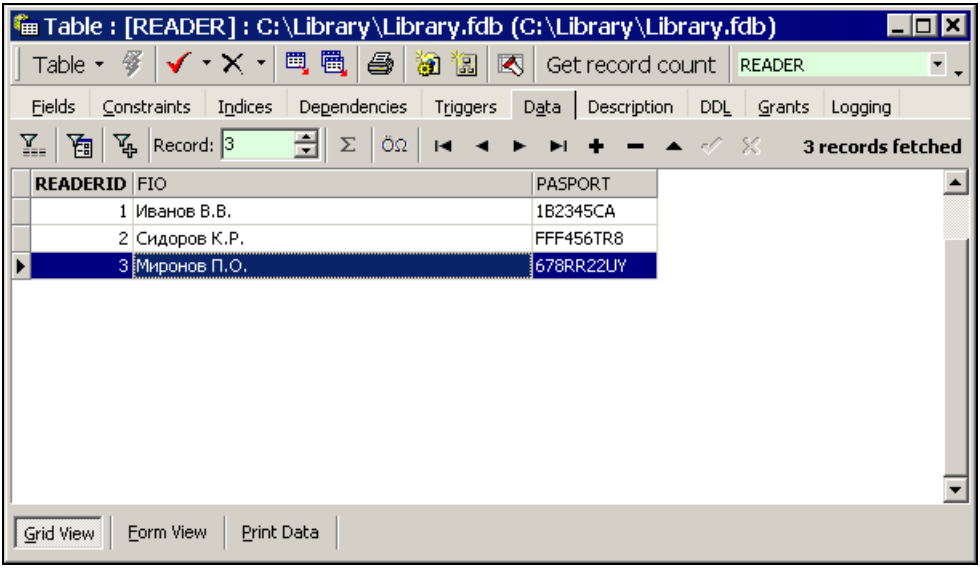


Рис. 5.41. Окно **Table** после добавления данных

С помощью расположенных в нижней части окна **Table** кнопок можно менять вид представления данных. Например, кнопка **Form View** включает плоский режим отображения данных (видно только одну запись), переключение между записями осуществляется с помощью навигатора (рис. 5.42).

Приступим к созданию таблицы Abonement. На рис. 5.43 представлена уже готовая таблица (именно такую нужно создать на своем компьютере). Атрибут `ReaderID` — первичный ключ и для него нужно создать генератор.

Так как атрибут `PriznakOfEnd` отражает информацию о том, действителен абонемент или нет, то, соответственно, этот атрибут является обязательным для заполнения. Не может быть такой ситуации, когда значение атрибута будет неопределенно, так как абонемент либо действителен, либо не действителен. Если же абонемент никому не выдан, то соответственно и не будет записи в таблице Abonement.

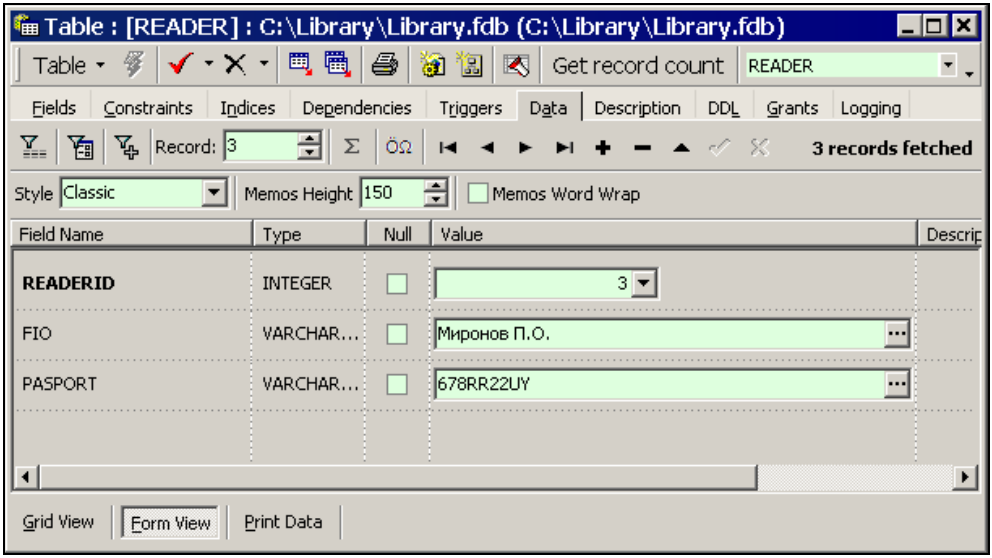


Рис. 5.42. Просмотр таблицы по одной записи

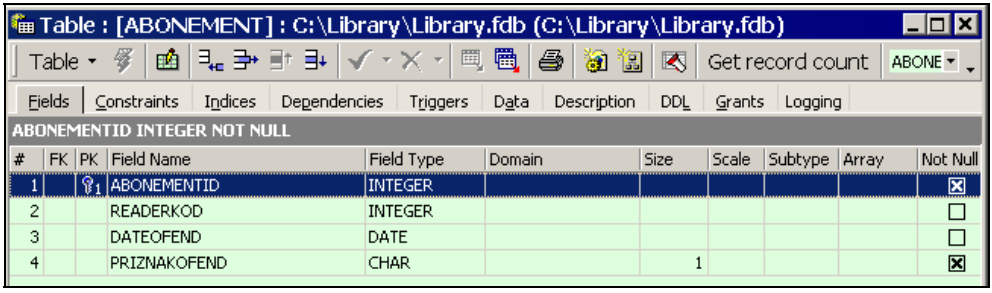


Рис. 5.43. Просмотр таблицы **Abonement**

Замечание

Для того чтобы атрибуту `PriznakOfEnd` установить признак **Not Null**, необходимо произвести двойной щелчок левой кнопкой мыши на PRIZNAKOFEND, в появившемся окне **Edit field PRIZNAKOFEND** установить флаг **Not NULL**, нажать кнопку **OK**, а потом на клавиатуре <Ctrl>+<F9>.

Атрибут `ReaderKod` содержит код читателя, которому принадлежит абонемент. Через данный атрибут происходит связь с таблицей Reader. Создадим для атрибута `ReaderKod` таблицы Abonement индекс. Делаем активной вкладку **Constraints**. Переключаемся на вкладку **Foreign keys**. Щелкаем правой

кнопкой мыши в свободном пространстве окна **Table** и в выпадающем меню выбираем пункт **New foreign key** (рис. 5.44).

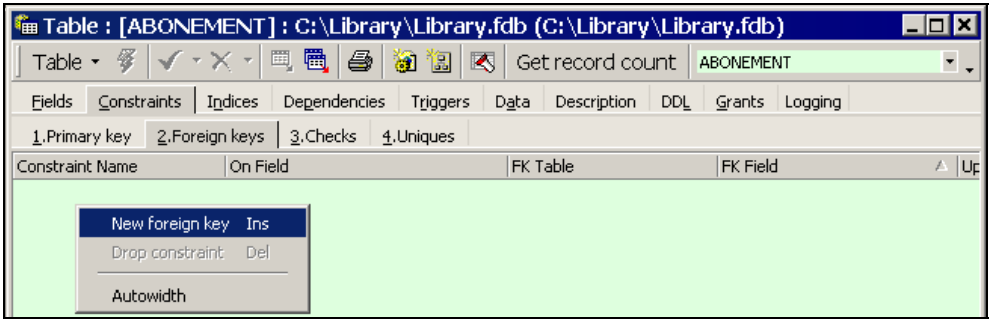


Рис. 5.44. Пункт **New foreign key**

Появится новая запись (рис. 5.45).

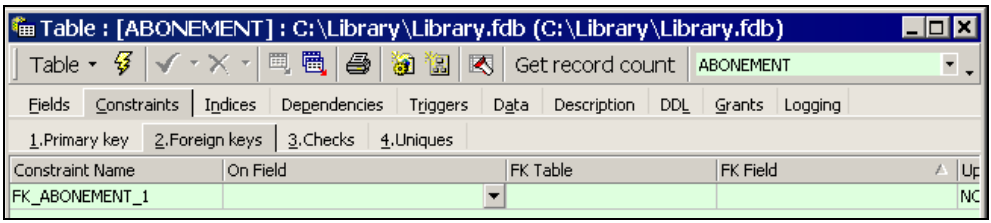


Рис. 5.45. Выбор имени внешнего ключа

Теперь необходимо установить, для какого атрибута создается индекс. Щелкаем левой кнопкой мыши в поле **On Field** и устанавливаем атрибут `ReaderKod` (рис. 5.46).

Теперь необходимо указать, на какую таблицу (поле **FK Table**) и на какой атрибут (поле **FK Field**) будет указывать создаваемый индекс. Выбираем таблицу Reader, и значение поля **FK Field** заполнится автоматически (главное условие этого, чтобы в таблице Reader был создан первичный ключ) — рис. 5.47 и 5.48.

Нажимаем <Ctrl>+<F9> или кнопку **Compile**. Появится окно **Compiling Constraints**, в котором отображаются SQL-команды на создание первичного ключа, а также комментарии по поводу запроса (**Statement List**) — рис. 5.49.

Нажимаем кнопку **Close**. Делаем активной вкладку **Fields** окна **Table** и видим, что в таблице Reader появился индекс (запись READERKOD с изображением ключа с буквой F) — рис. 5.50.

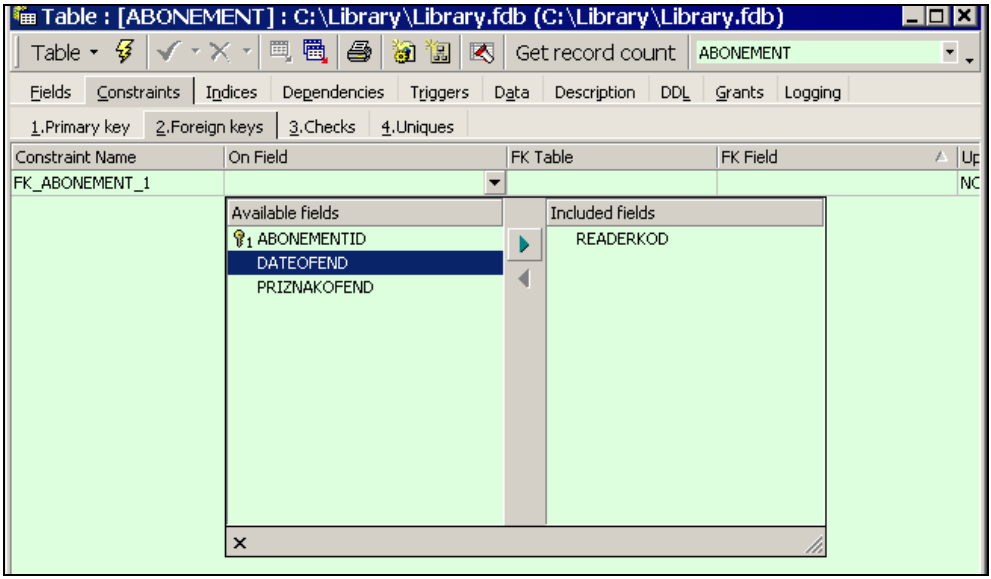


Рис. 5.46. Выбор полей внешнего ключа

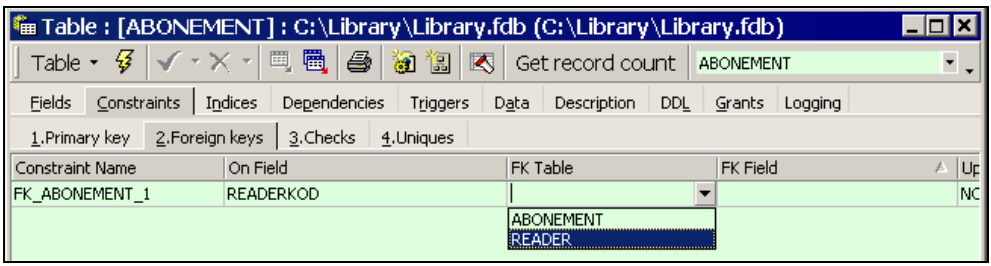


Рис. 5.47. Выбор связанной таблицы для внешнего ключа

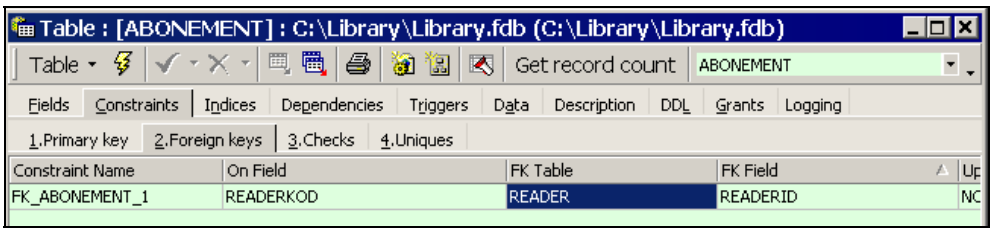


Рис. 5.48. Результат создания внешнего ключа

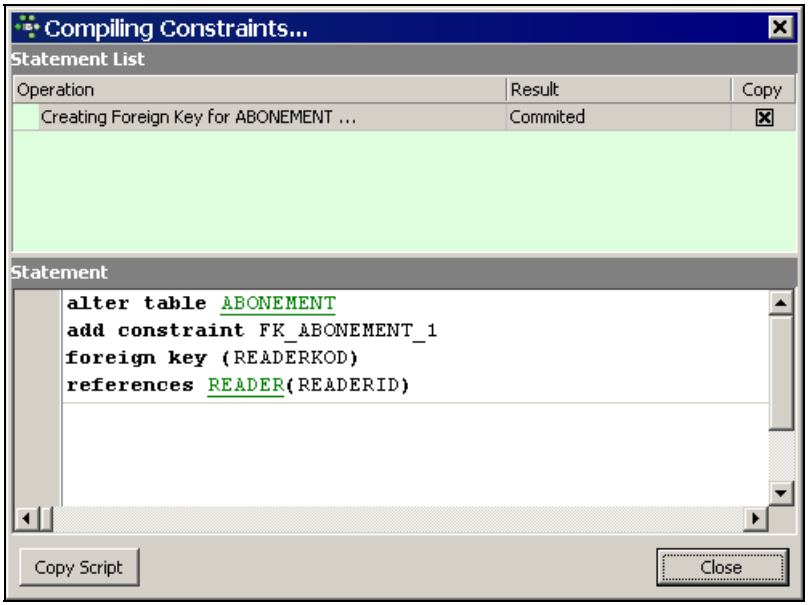


Рис. 5.49. SQL-команда создания внешнего ключа

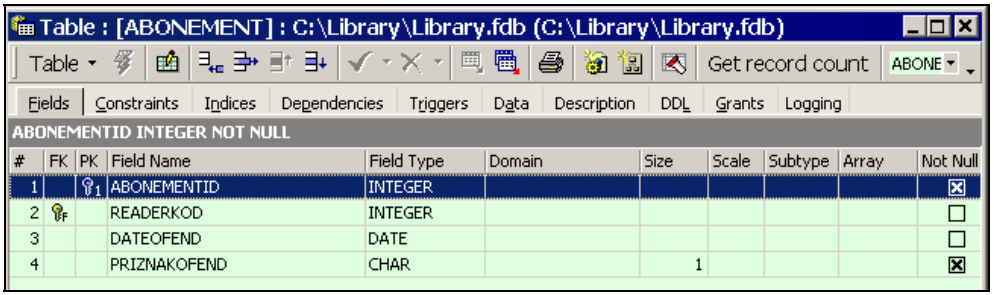


Рис. 5.50. Отображение признака создания внешнего ключа

Замечание

Если по ошибке индекс создан неправильно, то его можно удалить, для этого нужно щелкнуть правой кнопкой мыши на записи, соответствующей индексу, и в выпадающем списке выбрать **Drop foreign key**.

Создаем таблицу Book. Атрибут `BookID` будет ключевым, атрибут `OnHand` не должен содержать пустых значений — необходимо установить свойство **Not NULL** для этого атрибута. На рис. 5.51 представлена структура таблицы Book.

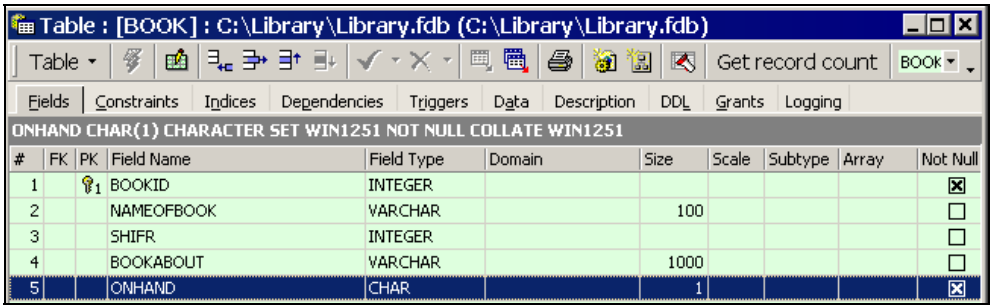


Рис. 5.51. Структура таблицы Book

Создаем последнюю таблицу Move — "Движение книг". Атрибут `MoveID` будет ключевым. `BookKod` будет содержать код книги, через данный атрибут осуществляется связь с таблицей Book, поэтому создаем индекс для данного атрибута, который будет ссылаться на поле `BookID` таблицы Book. Атрибут `AbonementKod` содержит код абонемента, через него осуществляется связь с таблицей Abonement, поэтому для данного атрибута также создаем индекс, который будет ссылаться на поле `AbonementID` таблицы Abonement. Атрибуты `DateOfReturn` и `PriznakOfReturn` являются обязательными, так как у каждой отданной читателю книги должна быть определенная дата возврата (атрибут `DateOfReturn`). Также нужно вести учет тех книг, которые были возвращены (атрибут `PriznakOfReturn`).

На рис. 5.52 представлена структура таблицы Move.

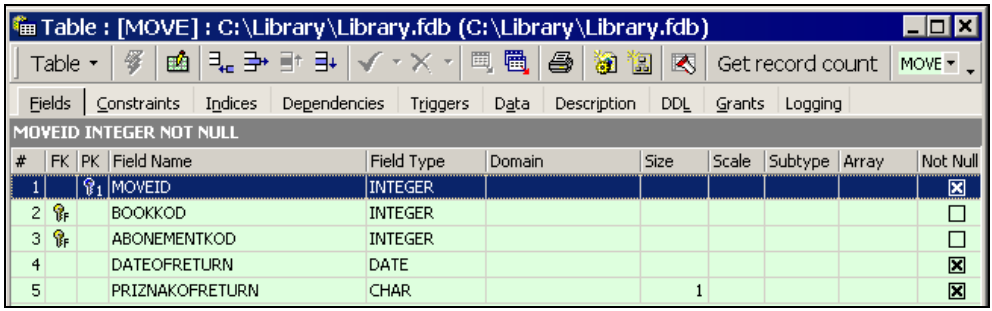


Рис. 5.52. Структура таблицы Move

Построение базы данных завершено.

В предыдущих разделах процесс добавления данных в таблицу реализовывался непосредственно в клиентском приложении. При разработке программы Библиотекарь мы пойдем другим путем. Мы создадим набор хранимых процедур, в которых будет реализован механизм добавления, изменения

и удаления данных, а также некоторые дополнительные действия. В клиентском приложении будет происходить обращение к процедурам.

5.6. Работа с хранимыми процедурами

Щелкаем правой кнопкой в окне **Database Explorer** на пиктограмме **Procedures**, в выпадающем меню выбираем **New Procedure**. Появится окно **Procedure**, в котором необходимо ввести следующий код для работы с данными таблицы Reader (окно **Procedure** с кодом представлено на рис. 5.53):

```
begin
--Присваиваем значение по умолчанию
result_output=1;

 --Если параметр равен 1, то
 if (choose_par=1) then
 begin
   -- вставка новой записи
     INSERT INTO READER (FIO, PASPORT)
             VALUES (:fio_par, :pasport_par);
  end

 -- Если параметр равен 2, то
 if (choose_par=2) then
 begin
   -- изменение записи
     UPDATE READER SET
            FIO=:fio_par,
            PASPORT=:pasport_par
     WHERE  ReaderID=:readerid_par;
  end

 -- Если параметр равен 3, то if (choose_par=3) then
 if (choose_par=3) then
 begin
   -- Если у читателя есть абонементы, то возвращаем 0
   -- Иначе удаление записи
     SELECT COUNT(*) FROM ABONEMENT
      WHERE READERKOD=:readerid_par INTO :countofabonement;
     if (Countofabonement>0) then
```

```
      begin
       result_output=0;
      end
         else
      begin
       DELETE FROM READER
              WHERE  ReaderID=:readerid_par;
      end
  end

end
```

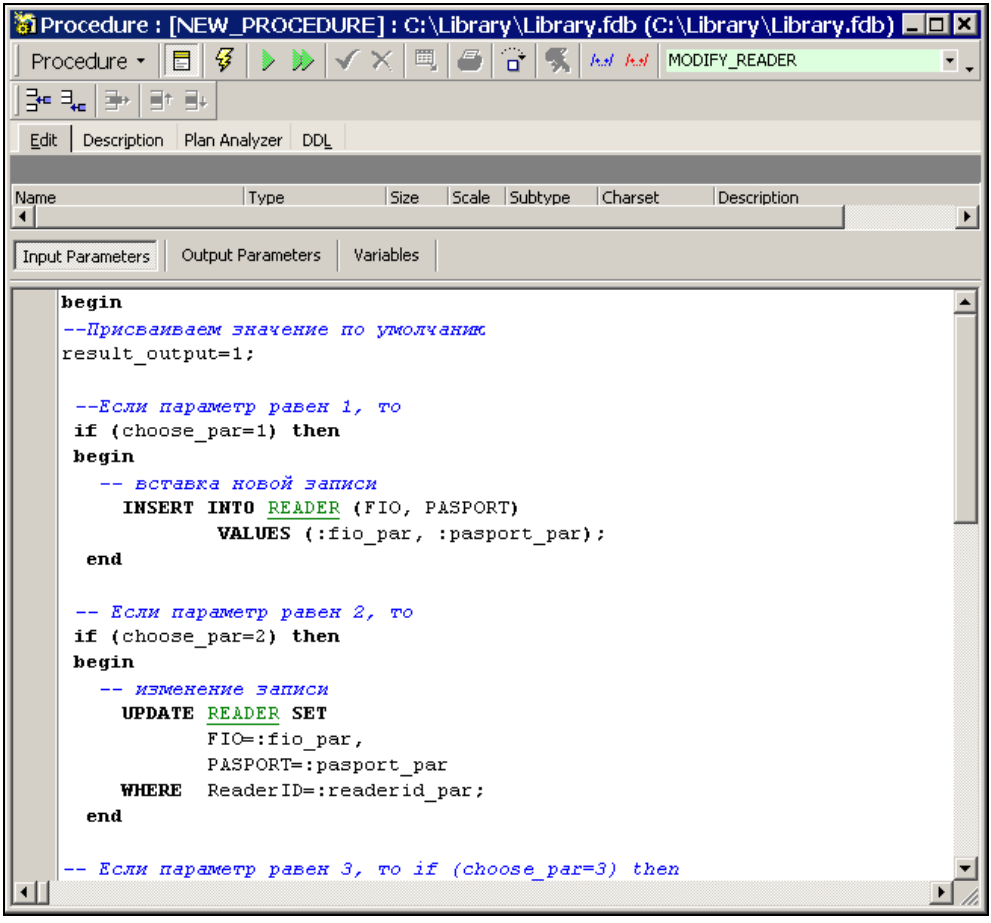


Рис. 5.53. Код для работы с данными таблицы Reader

Обо всем по порядку. Сначала необходимо задать имя будущей процедуры в правом верхнем углу окна **Procedure** (имя задается точно также, как и при создании таблицы). Назовем ее `MODIFY_READER`. В этой процедуре реализован механизм работы с таблицей Reader. То есть в зависимости от значения параметра `choose_par` выполнится одно из трех действий:

- Если `choose_par = 1`, то произойдет добавление новой записи, значения будут взяты из параметров `:fio_par`, `:pasport_par`.
- Если `choose_par = 2`, то произойдет изменение данных в уже существующей записи, значения будут взяты из параметров `fio_par`, `pasport_par`, `readerid_par`.
- Если `choose_par = 3`, то произойдет удаление данных в уже существующей записи, значение будет взято из параметра `readerid_par`. Перед тем как удалить запись, выполнится проверка: есть ли у удаляемого читателя абонементы. Если да, то выходному параметру `result_output` присвоится значение 0, что будет означать, что удаление не произошло. Если же у читателя нет абонементов, то запись удалится.

Замечание

У процедуры могут быть как входные, так и выходные параметры. Входные необходимы, чтобы клиентское приложение могло передавать данные в процедуру. Выходные позволяют процедуре возвращать значения обратно в программу. Входными параметрами являются: `readerid_par`, `fio_par`, `passport_par`, `choose_par`. Выходным параметром является `result_output`.

В самом начале у нас выходному параметру `result_output` присваивается единица. Таким образом, процедура будет всегда возвращать единичный результат, который будет говорить, что она обработала данные. Исключение составляет, когда процедуре передается `choose_par = 3`, и у удаляемого читателя есть абонемент, тогда процедура вернет 0, и, следовательно, мы сможем проинформировать пользователя о такой ситуации.

Всего в этом запросе используются 4 входных параметра, рассмотрим назначение каждого:

- `readerid_par` — код записи (используется для изменения и удаления);
- `fio_par` — фамилия читателя (используется для создания и изменения записи);
- `pasport_par` — паспортные данные читателя (используется для создания и изменения записи);
- `choose_par` — значение данного параметра указывает процедуре, какое именно действие необходимо совершить: создать, изменить или удалить.

Параметры, которые используются в командах SQL, начинаются с двоеточия. В других случаях двоеточия перед параметром не нужно, как, например, в этом фрагменте кода процедуры MODIFY_READER:

```
if (choose_par=1) then
 begin
-- вставка новой записи
     INSERT INTO READER (FIO, PASPORT)
             VALUES (:fio_par, :pasport_par);
  end
```

Перед параметром choose_par двоеточия нет, потому что он используется не в запросе. Также в процедуре используется конструкция IF—Then, аналогичная используемой в Delphi. Я думаю, с этим проблем возникнуть не должно. Еще один момент заключается в том, что после END разделителя точки с запятой ставить не нужно.

Замечание

Комментарии в процедуре начинаются после "--".

Рассмотрим еще один фрагмент кода процедуры MODIFY_READER:

```
SELECT COUNT(*) FROM ABONEMENT
WHERE READERKOD=:readerid_par INTO :countofabonement;
```

В данном запросе выбирается количество абонементов, которые принадлежат читателю с кодом, переданнем в параметре readerid_par. Причем результат запроса передается с помощью команды INTO в переменную countofabonement, которая пишется с двоеточием в начале, так как используется в запросе. Переменная нужна, потому что это значение необходимо только в процессе выполнения небольшого фрагмента кода процедуры и данное значение никуда не нужно передавать.

Теперь нам необходимо создать входные параметры, описанные в процедуре, для этого щелкаем правой кнопкой мыши в поле для параметров и выбираем в выпадающем меню пункт **Insert parameter/variable** (рис. 5.54).

После этого появится новая запись, в которой необходимо задать имя параметра (поле **Name**) и тип данных (поле **Type**), для некоторых типов данных, необходимо будет задать размер (поле **Size**). Надо создать 4 параметра, как на рис. 5.55.

Теперь необходимо создать выходные параметры. Для этого надо нажать на кнопку **Output Parameters**, которая расположена под самими параметрами (сейчас у нас нажата кнопка **Input Parameters**). Затем щелкнуть правой кнопкой мыши в поле для параметров и выбрать в выпадающем меню пункт

Insert parameter/variable, после этого добавить один-единственный выходной параметр, как показано на рис. 5.56.

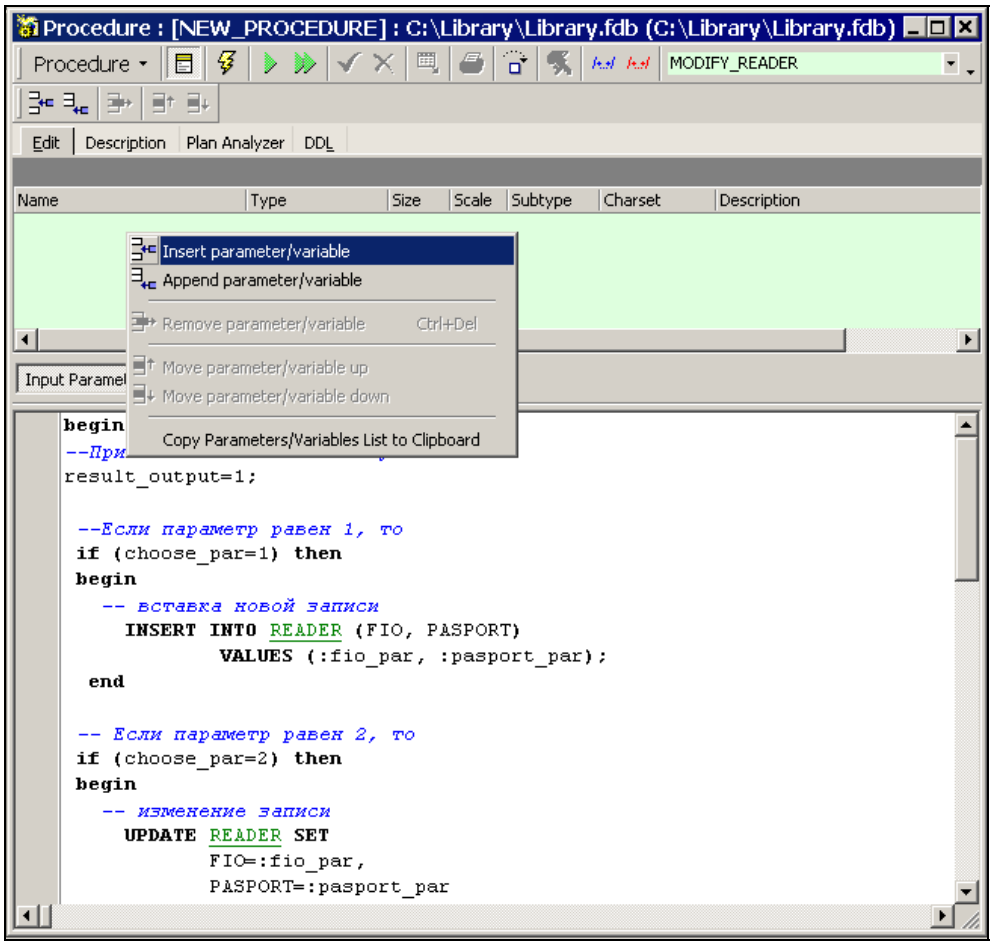


Рис. 5.54. Пункт **Insert parameter/variable**

Теперь создаем переменную. Для этого надо нажать на кнопку **Variables**. Затем щелкнуть правой кнопкой мыши в поле для переменных, выбрать в выпадающем меню пункт **Insert parameter/variable** и добавить переменную `countofabonement`, как показано на рис. 5.57.

Замечание

Полный текст по созданию процедуры можно увидеть, выбрав вкладку **DDL**.

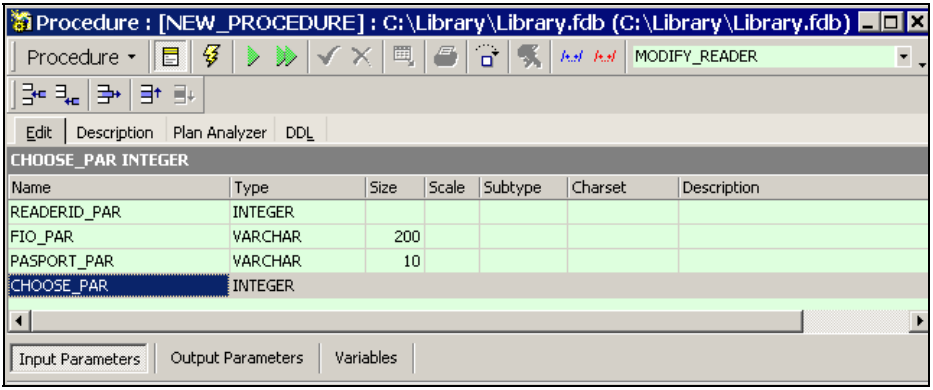


Рис. 5.55. Добавление входных параметров

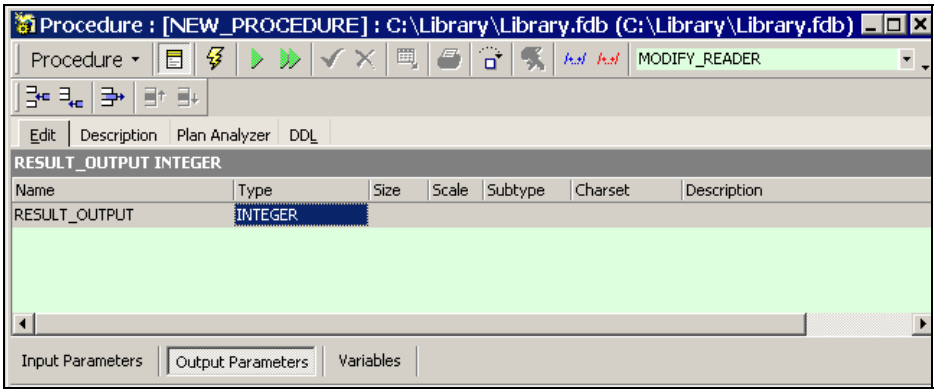


Рис. 5.56. Добавление выходного параметра

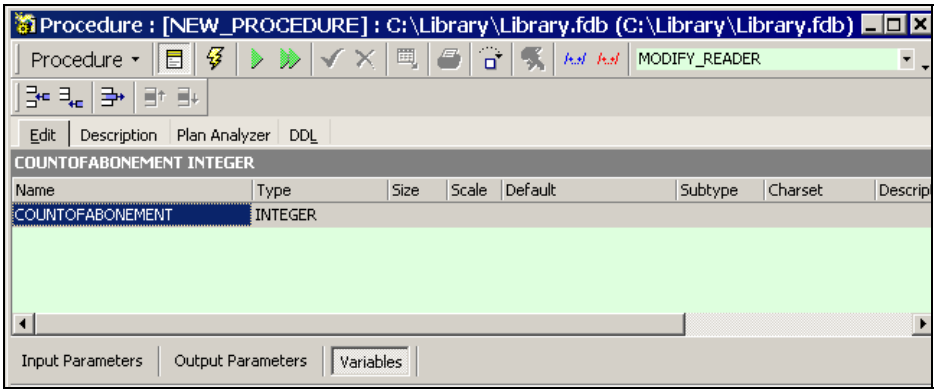


Рис. 5.57. Добавление переменной `countofabonement`

Нажимаем кнопку **Compile** или <Ctrl>+<F9>, появится окно **Creating Procedure** (рис. 5.58).

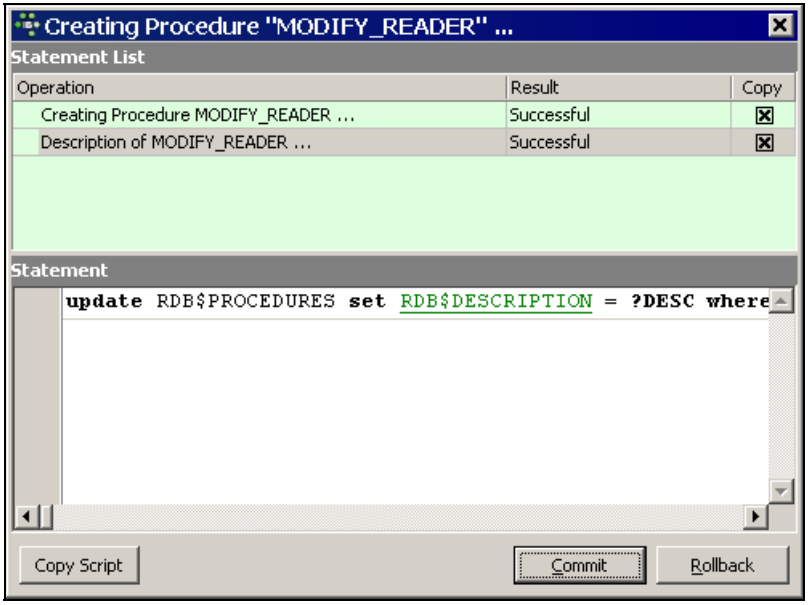


Рис. 5.58. Окно **Creating Procedure**

Необходимо нажать кнопку **Commit**, что приведет к созданию процедуры.

Замечание

Аналогичные процедуры необходимо создать для остальных трех таблиц.

Создаем новую процедуру, называем ее `MODIFY_ABONEMENT`, она будет предназначена для таблицы Abonement. Код процедуры представлен ниже:

```
begin
 -- Если параметр равен 1, то
 if (choose_par=1) then
 begin
   -- вставка новой записи
     INSERT INTO ABONEMENT (READERKOD, DATEOFEND, PRIZNAKOFEND)
     VALUES (:readerkod_par, :dateofend_par, :priznakofend_par);
  end
```

```
 -- Если параметр равен 2, то
 if (choose_par=2) then
 begin
   -- изменение записи
     UPDATE ABONEMENT SET
            READERKOD=:readerkod_par,
            DATEOFEND=:dateofend_par,
            PRIZNAKOFEND=:priznakofend_par
     WHERE  ABONEMENTID=:abonementid_par;
  end

end
```

Замечание

В данной процедуре используется тот же самый алгоритм, что и в предыдущей, за исключением того, что нет удаления данных.

Процедура предназначена для работы с таблицей Abonement, в зависимости от переданного значения в параметр `choose_par` выполняется либо добавление, либо изменение. Удаление абонементов производиться не будет, так как все абонементы представляют ценность как архивные данные. Достаточно, что на недействующих абонементах будет установлен признак окончания (атрибут `PriznakOfEnd` таблицы Abonement).

Замечание

Может возникнуть вопрос, зачем загромождать базу лишними записями и какая ценность в архивных данных. Предположим, что необходимо будет получить аналитический отчет о том, сколько раз бралась та или иная книга, для этого нам и потребуются эти абонементы, потому что к ним привязываются записи в таблице Move. Или потребуется узнать, сколько отдельный читатель брал книг. Невозможно будет подсчитать это значение, если данные о прошлых абонементах читателя будут удалены.

Очень важно думать о возможных путях развития разрабатываемого приложения в дальнейшем. Так как всегда легче дополнять, чем создавать заново. А чтобы дополнять, должны быть возможности для расширения. Использование атрибута `PriznakOfEnd`, как варианта для реализации формирования различных отчетов, в дальнейшем является неплохим решением.

В данной процедуре используется 5 входных параметров, которые необходимо будет создать:

- `abonementid_par` — код записи в таблице (используется для изменения), тип `Integer`;

- readerkod_par — код читателя, которому принадлежит данный абонемент (используется для создания и изменения записи), тип Integer;
- dateofend_par — дата окончания абонемента (используется для создания и изменения записи), тип Date;
- priznakofend_par — признак действительности абонемента (используется для создания и изменения записи), тип Char(1);
- choose_par — значение данного параметра указывает процедуре, какое именно действие необходимо совершить: создать или изменить, тип Integer.

Создаем новую процедуру, называем ее MODIFY_BOOK, она будет предназначена для таблицы Book. Код процедуры представлен ниже:

```
begin
 -- Если параметр равен 1, то
 if (choose_par=1) then
 begin
   -- вставка новой записи
     INSERT INTO BOOK (NAMEOFBOOK, SHIFR, BOOKABOUT, ONHAND)
             VALUES  (:nameofbook_par,  :shifr_par,  :bookabout_par,
:onhand_par);
  end

 -- Если параметр равен 2, то
 if (choose_par=2) then
 begin
   -- изменение записи
     UPDATE BOOK SET
            NAMEOFBOOK=:nameofbook_par,
            SHIFR=:shifr_par,
            BOOKABOUT=:bookabout_par,
            ONHAND=:onhand_par
     WHERE  BOOKID=:bookid_par;
  end

 -- Если параметр равен 3, то
 if (choose_par=3) then
 begin
   -- удаление записи
```

```
    DELETE FROM BOOK
            WHERE BOOKID=:bookid_par;
  end

end
```

Процедура предназначена для работы с таблицей Book. В зависимости от переданного значения в параметр `choose_par` выполняется добавление, изменение или удаление данных. Здесь важно учесть информацию, хранящуюся в атрибуте `OnHand`. Если данный атрибут установлен, значит, книга на руках у читателя и, соответственно, удалять ее нельзя. В данном случае проверка на то, можно ли производить удаление, будет реализована в клиентском приложении. Данный способ выбран специально, чтобы показать читателю различные методы реализации одних и тех же действий.

В данной процедуре используется 6 входных параметров, которые необходимо будет создать:

- `bookid_par` — код записи в таблице (используется для изменения и удаления), тип `Integer`;
- `nameofbook_par` — название книги (используется для создания и изменения записи), тип `Varchar(100)`;
- `shifr_par` — шифр книги (используется для создания и изменения записи), тип `Integer`;
- `bookabout_par` — краткое содержание книги (используется для создания и изменения записи), тип `Varchar(1000)`;
- `onhand_par` — признак нахождения книги у читателя (используется для создания и изменения записи), тип `Char(1)`;
- `choose_par` — значение данного параметра указывает процедуре, какое именно действие необходимо совершить: создать, изменить или удалить, тип `Integer`.

Создаем новую процедуру, называем ее `CLOSE_ABONEMENT`, она будет предназначена для закрытия абонемента. Код процедуры представлен ниже:

```
begin
 result_output=1;

   SELECT COUNT(*) FROM MOVE
   WHERE (ABONEMENTKOD=:abonementid_par AND PRIZNAKOFRETURN='0')
   INTO :countofbook;
     if (countofbook>0) then
```

```
    begin
     result_output=0;
    end
       else
    begin
      UPDATE ABONEMENT SET
           PRIZNAKOFEND='1'
      WHERE  ABONEMENTID=:abonementid_par;
    end
end
```

Сначала мы проверяем, есть ли невозвращенные книги на этом абонементе. Это достигается за счет выбора записей из таблицы Move, у которых атрибут `AbonementKod` содержит значение, равное переданному в параметре `abonemenid_par`, и у которых в атрибуте `PriznakOfReturn` установлен ноль (ноль означает, что книга не возвращена).

Если количество записей больше 0 (`countofbook>0`), значит, абонемент закрывать нельзя, поэтому процедура вернет нулевой результат и можно будет обработать его в клиентском приложении. Если количество возвращенных записей — ноль, то закрываем абонемент (то есть устанавливаем атрибут `PriznakOfEnd` в единицу).

В данной процедуре используются следующие элементы:

- входной параметр `abonementid_par` — код записи в таблице Abonement, тип `Integer`;
- выходной параметр `result_output` — возвращает результат выполнения процедуры, тип `Integer`;
- переменная `countofbook` — хранит количество записей, возвращенных проверяющим запросом, тип `Integer`.

Создаем новую процедуру, называем ее `GIVE_BOOK`, она будет предназначена для выдачи книги читателю. Код процедуры представлен ниже:

```
begin
 INSERT INTO MOVE(BOOKKOD,ABONEMENTKOD,DATEOFRETURN,PRIZNAKOFRETURN)
 VALUES (:bookkod_par,:abonementkod_par,:dateofreturn_par,'0');

 UPDATE BOOK
 SET OnHand='1'
 WHERE BOOKID=:bookkod_par;

end
```

В данной процедуре мы добавляем новую запись (INSERT INTO) в таблицу Move, значения атрибутов берутся из входных параметров. Далее командой UPDATE в таблице Book мы устанавливаемой выданной книге атрибут OnHand в единицу. Это нужно для того, чтобы знать, что книга не в библиотеке, и пока она не будет возвращена (пока атрибут OnHand не будет установлен в 0), ее никому нельзя будет выдать.

В данной процедуре используется 3 входных параметра, которые необходимо будет создать:

- bookkod_par — код записи в таблице Book, тип Integer;
- abonementkod_par — код записи в таблице Abonement, тип Integer;
- dateofreturn_par — дата возврата книги (используется для создания и изменения записи), тип Date.

Создаем новую процедуру, называем ее RETURN_BOOK, она будет предназначена для проведения операции возврата книги. Код процедуры представлен ниже:

```
begin
  UPDATE BOOK
  SET OnHand='0'
  WHERE BOOKID=:bookid_par;

  UPDATE MOVE
  SET PRIZNAKOFRETURN='1'
  WHERE (BOOKKOD=:bookid_par)
            AND
        (ABONEMENTKOD=:abonementkod_par);

end
```

В данной процедуре мы устанавливаем в 0 атрибут OnHand таблицы Book. Затем устанавливаем в 1 атрибут PriznakOfReturn таблицы Move. Тем самым мы указываем, что книга возвращена.

В данной процедуре используется 2 входных параметра, которые необходимо будет создать:

- bookid_par — код записи в таблице Book, тип Integer;
- abonementkod_par — код абонемента, тип Integer.

5.7. Резервное копирование и восстановление базы данных

Для того чтобы выполнить резервное копирование базы, нужно выбрать пункт главного меню **Services\Backup Database**. На экране появится окно **Database Backup** (рис. 5.59).

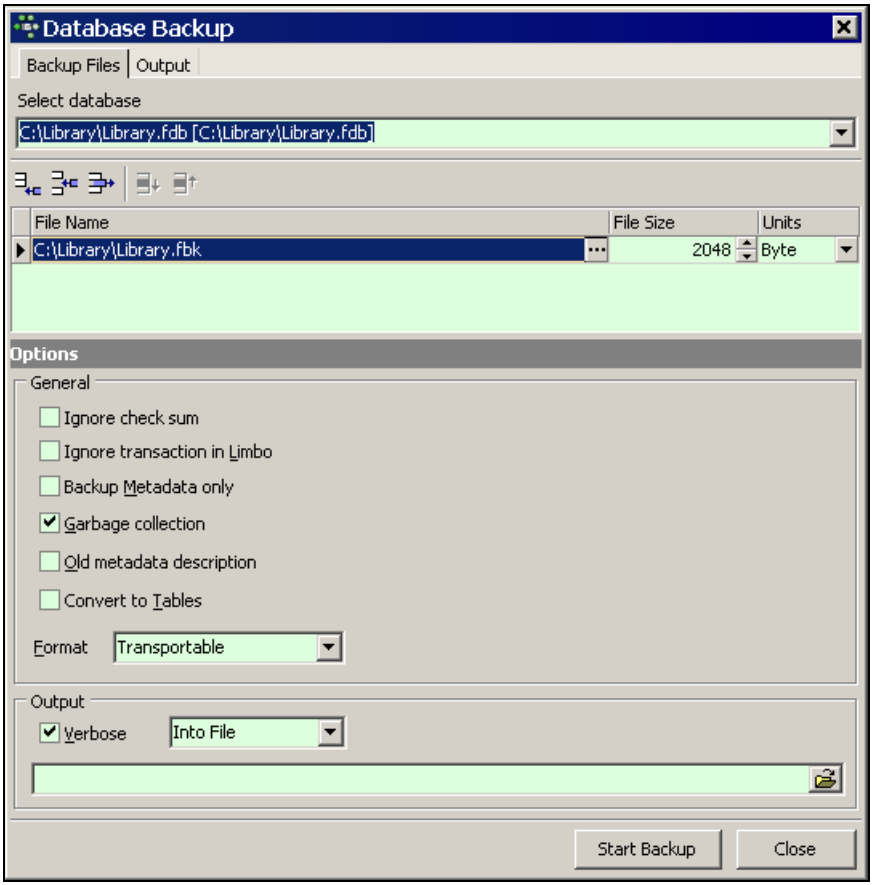


Рис. 5.59. Вкладка **Backup Files** окна **Database Backup**

В выпадающем списке **Select database** выбирается имя базы данных, для которой создается архивная копия. В столбце **File Name** выбирается путь и имя файла архивной копии. После этого необходимо нажать на кнопку **Start-Backup**. В окне **Database Backup** активной станет вкладка **Output**, в которой

будет отображаться ход процесса резервного копирования. После того как он закончится, нажмите кнопку **Close** (рис. 5.60).

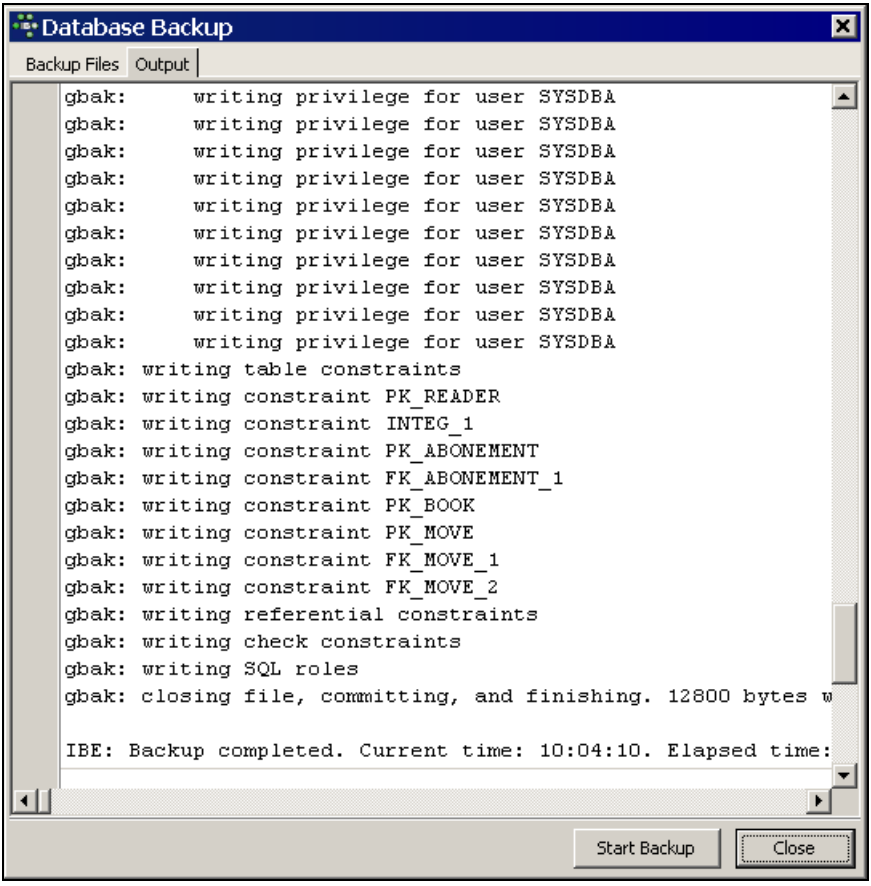


Рис. 5.60. Вкладка **Output** окна **Database Backup**

Чтобы приступить к восстановлению базы данных, нужно выбрать пункт главного меню **Services\Restore Database**. На экране появится окно **Database Restore** (рис. 5.61).

В выпадающем списке **Restore into** выбирается, куда будет производиться восстановление:

- **Existing database** — в существующую базу (оставляем данный пункт);
- **New database** — в новую базу.

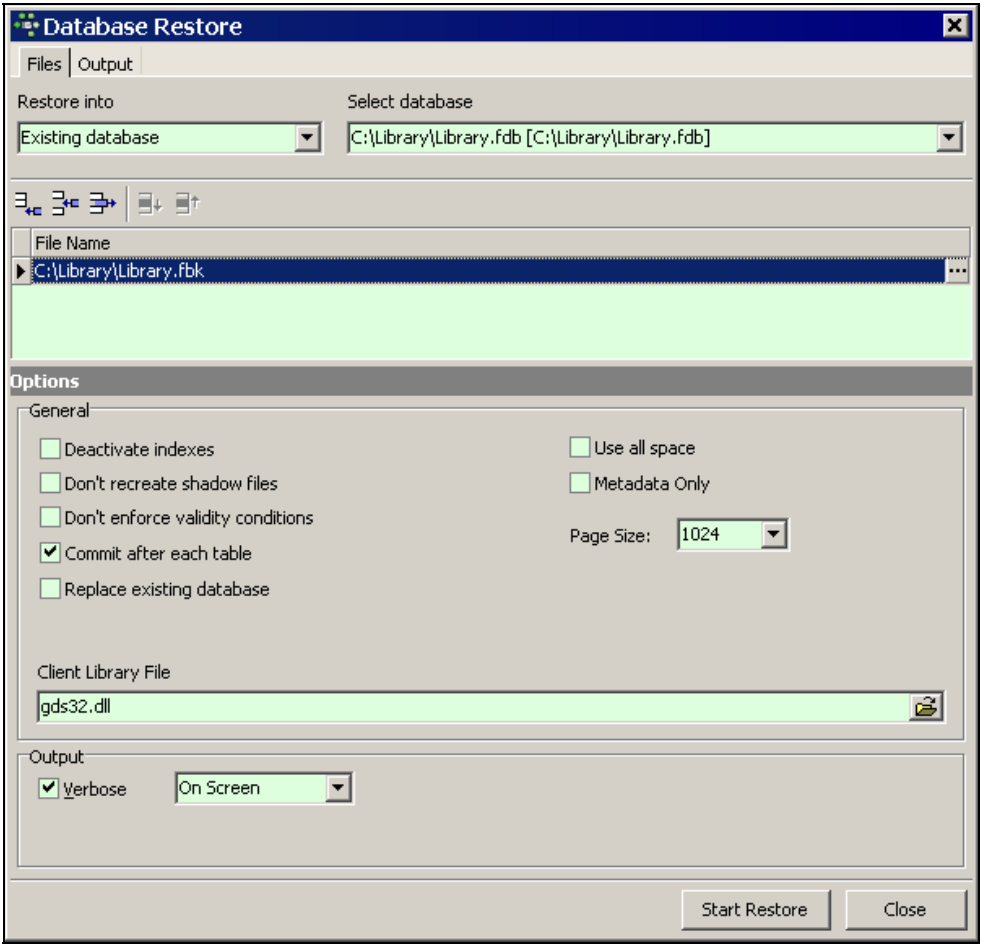


Рис. 5.61. Вкладка **Files** окна **Database Restore**

После этого в поле **Select database** выбирается либо существующая база, либо задается новая. В столбце **File Name** указывается файл резервной копии, затем нажимаем кнопку **Start Restore**.

5.8. Разработка приложения "клиент-сервер" в Delphi

В данном разделе мы будем использовать компоненты, описанные в табл. 5.6—5.8.

Таблица 5.6. *Описание компонента* `IBStoredProc` *с вкладки* ***InterBase***

Название	Основные свойства	Комментарии
IBStoredProc		Предназначен для работы с хранимыми процедурами. Позволяет выполнять их, подавать входные данные и получать результат выполнения (выходные данные)
	DatabaseName	Содержит имя компонента IBDatabase
	StoredProcName	Содержит имя хранимой процедуры, работа с которой будет осуществляться посредством данного компонента

Таблица 5.7. *Описание компонента* `MaskEdit` *с вкладки* ***Additional***

Название	Основные свойства	Комментарии
MaskEdit		Поле для ввода данных в соответствии с заданной маской
	EditMask	Содержит маску
	Text	Содержит введенный текст

Таблица 5.8. *Описание компонента* `PageControl` *с вкладки* ***Win32***

Название	Основные свойства	Комментарии
PageControl		Позволяет создать многостраничный элемент. Переключение между страницами осуществляется с помощью вкладок

Также будут использоваться компоненты DataSource, DBGrid, DBNavigator, Button, Label, Edit, IBDatabase, IBTransaction, IBQuery — их описание можно найти в *разд. 3.7 и 4.9*.

Как упоминалось раньше, разработка клиентского приложения будет состоять из двух частей. Сначала мы разработаем модуль Администратора, где работник библиотеки сможет менять информацию по книгам (добавлять, изменять, удалять), выдавать читателям абонементы, регистрировать выдачу и возврат книг, а также отслеживать просроченные абонементы и не сданные в срок книги.

Второй модуль будет предназначен для читателей, где каждый читатель, введя имя и пароль, сможет просмотреть информацию о доступных книгах в библиотеке, а также те книги, которые у него на руках. Именем будет значение атрибута `FIO`, таблицы Reader, а паролем — значение атрибута `ReaderKod` таблицы Abonement.

Замечание

В данной главе будем использовать уже знакомые по третьей главе компоненты с вкладки InterBase (как уже ранее упоминалось, FireBird — клон InterBase, поэтому работа с этими СУБД практически не отличается).

Создаем новый проект в Delphi. Форму называем `MainForm`, в свойство `Caption` пишем "Администратор библиотеки". Сохраняем форму под именем `UMain`, а проект под именем `Library_proj`. В свойстве `Align` выбираем `alTop`. Помещаем компонент `MainMenu` на форму. Создаем пункты меню по следующей схеме:

- ❒ Работа с данными
  - !!! Внимание !!!
  - Выдача\Возврат книг
- ❒ Справочники
- ❒ Аналитика
  - История по книге
  - История по читателю
- ❒ Выход

Изменяем размеры формы `MainForm`, чтобы она выглядела как на рис. 5.62.

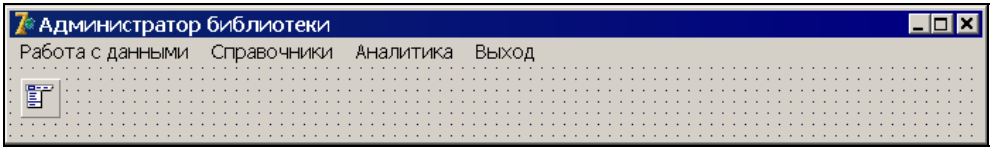


Рис. 5.62. Форма `MainForm` после изменения размера

Для пункта меню "Выход" пишем следующий код:

```
Application.Terminate
```

Создаем DataModule. Здесь у нас будут располагаться основные компоненты доступа к базе данных. Называем его `DM` и сохраняем под именем `UDM`.

Помещаем компонент `IBDatabase`. Свойство `Name` меняем на `DBLibrary`. Щелкаем на нем два раза левой кнопкой мыши (можно также один раз пра-

вой и в выпадающем меню выбрать пункт **Database Editor**), появится окно **Database Component Editor**. В поле **Database** прописываем путь к базе (можно воспользоваться кнопкой **Browse**), у меня он — C:\Library\Library.fdb.

В группе элементов **Database Parameters** устанавливаем параметры подключения к базе данных:

- поле **User Name** — пользователь `SYSDBA`;
- поле **Password** — пароль `masterkey`;
- **Character Set** — кодировка `WIN1251`;
- переключатель **Login Promt** необходимо сбросить, чтобы каждый раз при подключении к базе не появлялось окно ввода имени пользователя и пароля;
- в элементе **Settings** отображаются установленные параметры.

На рис. 5.63 представлено окно **Database Component Editor** с установленными параметрами подключения.

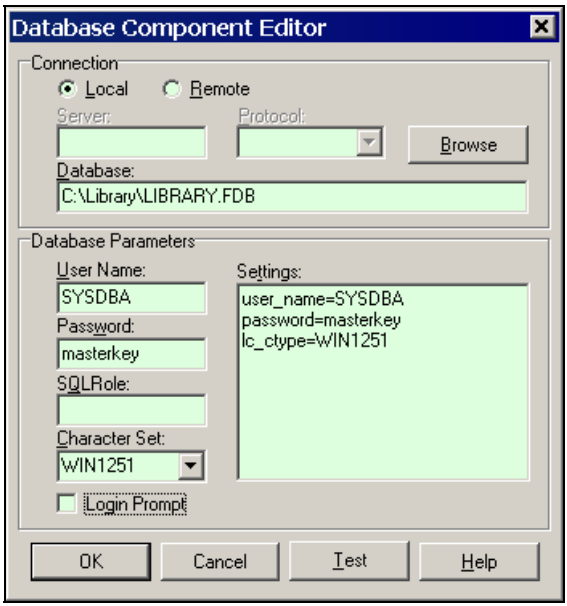


Рис. 5.63. Окно **Database Component Editor**

Замечание

Для проверки соединения с базой данных можно нажать кнопку **Test**. Если все параметры настроены правильно, то появится окно с текстом **Successful Connection** (Соединение успешно), в противном случае высветится окно с ошибкой.

Нажимаем **OK**. Помещаем компонент `IBTransaction`. В свойство `DefaultDatabase` устанавливаем `DBLibrary`.

Замечание

Как уже упоминалось, данный компонент отвечает за работу всех транзакций для связанной с данным компонентом базой данных.

Помещаем компонент `IBQuery` (он будет обращаться к таблице Book), называем его `QAllBook`. В свойство `DataBase` устанавливаем `DBLibrary`. В свойство `SQL` пишем запрос для выбора всех данных из таблицы Book:

```
SELECT * FROM BOOK ORDER BY NameOfBook
```

Данный запрос означает выбрать (команда `SELECT`) все данные из таблицы Book и отсортировать (команда `ORDER`) по названию книги (атрибут `NameOfBook`).

Производим двойной щелчок левой кнопкой мыши по компоненту `QAllBook`. Щелкаем правой кнопкой мыши в появившемся окне и выбираем **Add all fields** (рис. 5.64).

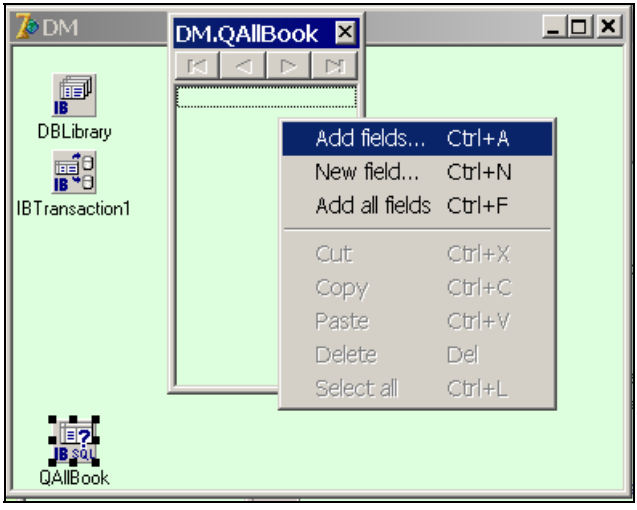


Рис. 5.64. Выбор пункта **Add all fields**

После этого действия данное окно примет вид, как показано на рис. 5.65.

Замечание

В дальнейшем я более не буду расписывать процесс добавления атрибутов таблицы в набор данных, а буду просто употреблять выражение: "Необходимо создать объекты-столбцы в редакторе столбцов".

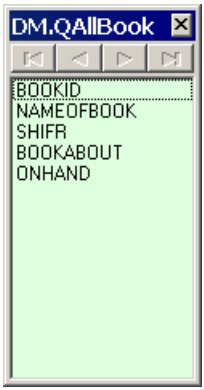


Рис. 5.65. Вид окна со списком полей компонента `QAllBook` после выбора всех полей

Предлагаю сразу обратится к рис. 5.67, чтобы не возникло проблем с размещением компонентов доступа к базе на модуле данных.

Атрибуты `BookID`, `BookAbout`, `OnHand` делаем невидимыми. Для остальных правим свойство `DisplayLabel`.

Теперь помещаем компонент `DataSource`, называем его `DSAllBook`, связываем его с набором данных `QAllBook`, для этого в свойство `DataSet` устанавливаем `QAllBook`.

Размещаем компонент `IBStoredProc`, называем его `ProcModifyBook`. В свойстве `DatabaseName` устанавливаем `DBLibrary`. В свойстве `StoredProcName` выбираем процедуру `MODIFY_BOOK`, название компоненту даем `ProcModifyBook`. Щелкаем по свойству `Params`. На экране появится окно **Editing ProcModifyBook.Params**, содержащее все используемые в процедуре параметры (рис. 5.66).

Помещаем компонент `IBQuery` (он будет обращаться к таблице Reader), называем его `QAllReader`. В свойство `DataBase` устанавливаем `DBLibrary`. В свойство `SQL` пишем запрос для выбора всех данных из таблицы Reader:

```
SELECT * FROM READER ORDER BY FIO
```

Данный запрос означает выбрать (команда `SELECT`) все данные из таблицы Reader и отсортировать (команда `ORDER`) по фамилии (атрибут FIO). Необходимо создать объекты-столбцы в редакторе столбцов. Для объекта-столбца ReaderID и в инспекторе объектов свойство `Visible` устанавливаем в `False`.

Остальным объектам-столбцам правим свойство `DisplayLabel`, можно также и свойство `DisplayWidth`.

Теперь помещаем компонент `DataSource`, называем его `DSAllReader`, связываем его с набором данных `QAllReader`.

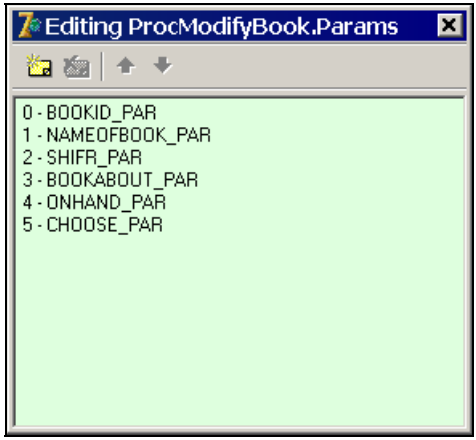


Рис. 5.66. Окно параметров процедуры

Размещаем компонент `IBStoredProc`, называем его `ProcModifyReader`. В свойство `DatabaseName` устанавливаем `DBLibrary`. В свойстве `StoredProcName` выбираем процедуру `MODIFY_READER`, название даем `ProcModifyReader`.

Помещаем компонент `IBQuery` (он будет обращаться к таблице Abonement), называем его `QAllAbonement`. В свойство `DataBase` устанавливаем `DBLibrary`. В свойство `SQL` пишем запрос для выбора всех данных из таблицы Abonement и Reader по условию:

```
SELECT * FROM ABONEMENT, READER
WHERE Abonement.ReaderKod=Reader.ReaderID
ORDER BY DateOfEnd
```

Данный запрос означает: выбрать (команда `SELECT`) все данные из таблиц Abonement и Reader, учесть, что таблицы связаны по коду читателя, и отсортировать (команда `ORDER`) по дате окончания абонемента (атрибут `DateOfEnd`). Необходимо создать объекты-столбцы в редакторе столбцов.

Теперь помещаем компонент `DataSource`, называем его `DSAllAbonement`, связываем его с набором данных `QAllBook` (в свойство `DataSet` устанавливаем `QAllBook`).

Размещаем компонент `IBStoredProc`. В свойство `DatabaseName` устанавливаем `DBLibrary`. В свойстве `StoredProcName` выбираем процедуру `MODIFY_ABONEMENT`, название даем `ProcModifyAbonement`.

Размещаем еще один компонент `IBStoredProc`. В свойство `DatabaseName` устанавливаем `DBLibrary`. В свойстве `StoredProcName` выбираем процедуру `CLOSE_ABONEMENT`, название даем `ProcCloseAbonement`.

Размещаем еще один компонент IBStoredProc. В свойство DatabaseName устанавливаем DBLibrary. В свойстве StoredProcName выбираем процедуру RETURN_BOOK, название даем ProcReturnBook.

Размещаем еще один компонент IBStoredProc. В свойство DatabaseName устанавливаем DBLibrary. В свойстве StoredProcName выбираем процедуру GIVE_BOOK, название даем ProcGiveBook.

Вид модуля данных после размещения на нем описанных выше компонентов представлен на рис. 5.67.

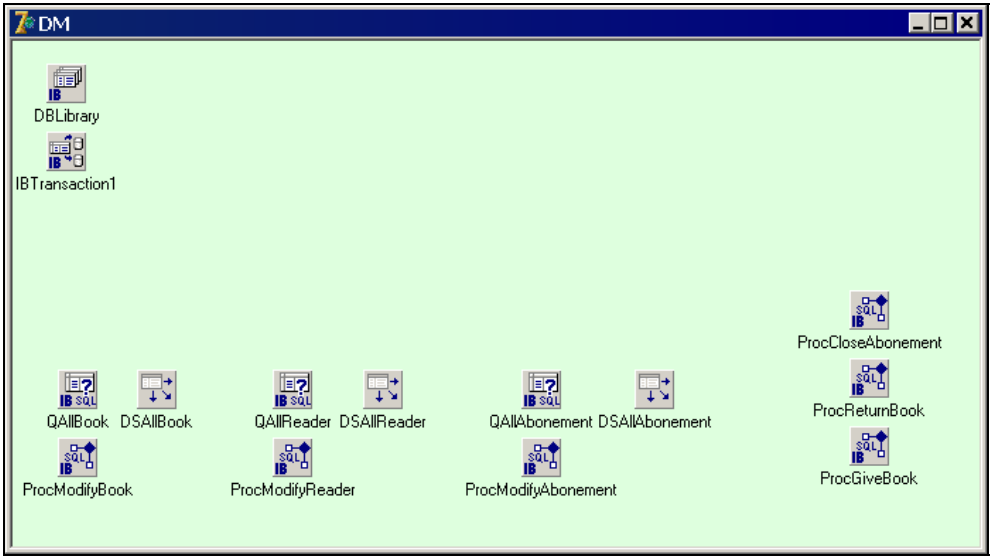


Рис. 5.67. Вид модуля данных после размещения на нем компонент

5.8.1. Разработка формы "Справочники"

Создаем новую форму, называем ее FormHandBook, сохраняем под именем UHandBook. В свойство Caption пишем "Справочники". Выбираем пункт главного меню **Project\Options** и удаляем ее из списка автоматически создаваемых форм. Располагаем на этой форме компонент PageControl. После этого создаем в нем 3 вкладки и называем их "Книги", "Читатели", "Абонементы".

На вкладке "Книги" располагаем компонент DBGrid. Называем его DBGridBook. Подключаем к форме модуль UDM. Связываем DBGridBook и DSAllBook с помощью свойства DataSource.

Теперь необходимо разместить Label с текстом "Название книги", под ним — разместить Edit, у которого нужно очистить свойство Text и свойство Name изменить на EditNameOfBook.

Размещаем еще один Label с текстом "Шифр", под ним размещаем MaskEdit, у которого нужно очистить свойство Text и свойство Name — изменить на EditShifr. В свойстве EditMask выбираем маску под названием Extension; таким образом, пользователь не сможет ввести ничего кроме чисел (рис. 5.68).

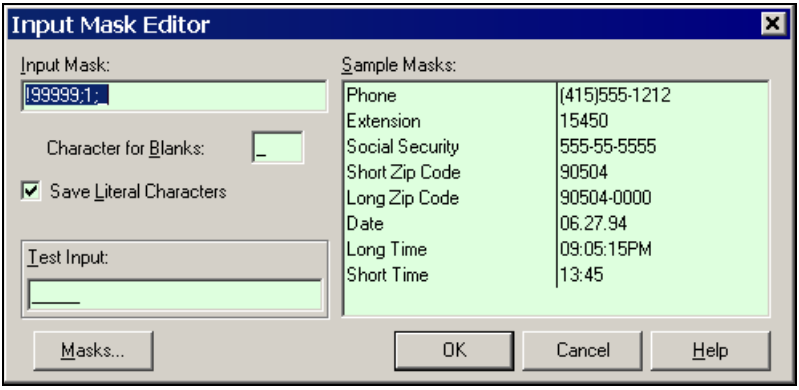


Рис. 5.68. Установка свойства EditMask

Замечание

Данное действие с маской называется защитой данных на уровне приложения. Если бы условие ввода корректных данных отслеживалось средствами базы данных, то была бы защита на уровне базы данных.

Необходимо разместить еще один Label с текстом "Краткое содержание", под ним поместить Memo, у которого свойство Name изменить на MemoBookAbout.

После этого располагаем три компонента Button, в свойство Caption которых пишем "Добавить", "Изменить", "Удалить". А в свойство Name, соответственно, — ButtonAddBook, ButtonChangeBook, ButtonDeleteBook.

Назначение кнопок будет состоять в следующем: когда пользователь хочет добавить данные, он просто заполняет поля "Название книги", "Шифр", "Краткое содержание" и нажимает кнопку **Добавить**. Если хочет изменить данные, то выделяет нужную запись в DBGrid и исправляет данные в этих же полях, после этого нажимает кнопку **Изменить**. Если собирается удалить, то выделяет в DBGrid нужную запись и нажимает **Удалить**. Таким вот про-

стым способом мы избежали того, чтобы создавать отдельные формы для каждого действия.

Вид формы с активной вкладкой **Книги** после размещения всех компонентов представлен на рис. 5.69.

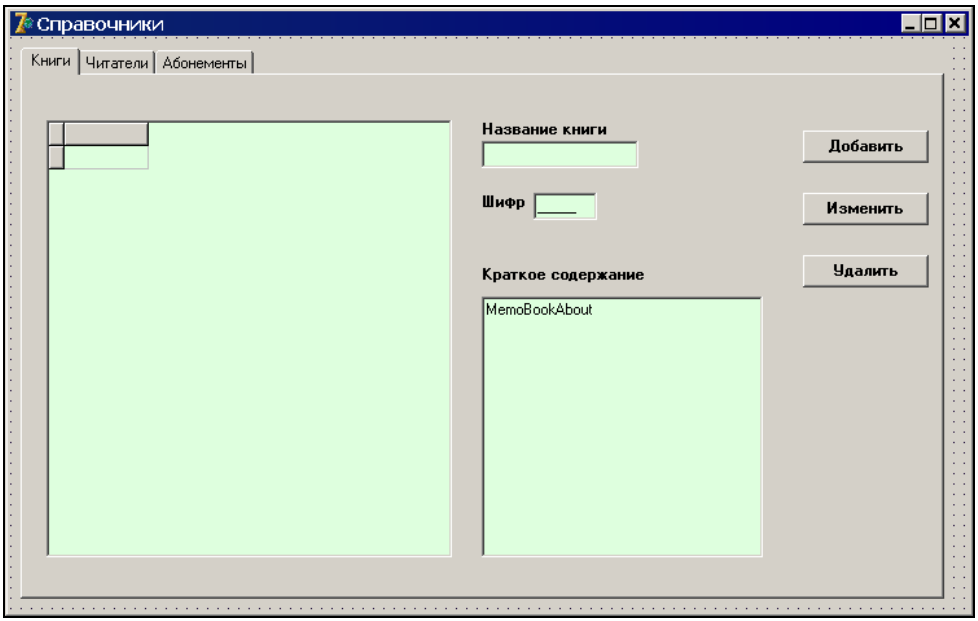


Рис. 5.69. Вид формы с активной вкладкой **Книги**

Теперь переходим к модулю `UMain` и объявляем там глобальную переменную `ExistsFormHandBook`:

```
var
  MainForm: TMainForm;
  {Создана ли форма HandBook}
  ExistsHandBook:boolean=false;
```

Если форма `FormHandBook` создана, то переменная будет содержать значение `True`, иначе — `False`.

Переходим к модулю `UDM`, подключаем модуль `UMain` и `UHandBook` и для компонента `QAllBook` создаем обработчик события `AfterScroll`:

```
procedure TDM.QAllBookAfterScroll(DataSet: TDataSet);
begin
 {если форма - Справочник создана, то}
 if ExistsHandBook then
```

```
 begin
  {Помещаем в EditNameBook название книги}
  FormHandBook.EditNameOfBook.Text:=QAllBookNAMEOFBOOK.Value;
  {Помещаем в EditShifr название книги}
  FormHandBook.EditShifr.Text:=IntToStr(QAllBookSHIFR.Value);
  {Помещаем в MemoBookAbout краткое содержание книги}
  FormHandBook.MemoBookAbout.Lines.Text:=QAllBookBOOKABOUT.Value;

  {Устанавливаем доступные кнопки}

  {Если книга на руках, то
  данные в справочник можно только добавлять,
  удалять и изменять такую запись нельзя}
  if QAllBookONHAND.Value='1' then
  begin
   {Блокируем кнопку Изменить}
   FormHandBook.ButtonChangeBook.Enabled:=false;
   {Блокируем кнопку Удалить}
   FormHandBook.ButtonDeleteBook.Enabled:=false;
  end
    else {Иначе, все кнопки доступны}
  begin
   FormHandBook.ButtonChangeBook.Enabled:=true;
   FormHandBook.ButtonDeleteBook.Enabled:=true;
  end;

 end;
end;
```

Замечание

Событие `AfterScroll` возникает после того, как в наборе данных происходит переход от одной записи к другой.

Смысл данного кода заключается в следующем: когда происходит переход между записями, возвращенных компонентом `QAllBook`, данные из текущей записи помещаются в поля **Название книги**, **Шифр**, **Краткое содержание**. Это сделано для более наглядного отображения информации и удобной работы с ней. Также осуществляется проверка, если для текущей записи признак `On-`

Hand установлен в 1, значит, книга на руках у читателя, и ее нельзя удалять и менять информацию о ней. Поэтому в этом случае соответствующие кнопки блокируются.

Переходим на форму FormHandBook и для компонента DBGridBook устанавливаем свойство DefaultDrawing в False, тем самым мы указываем, что будем самостоятельно следить за выводом данных на экран. В событии OnDrawColumnCell, которое отвечает за прорисовку сетки компонента DBGrid и данных в ней, пишем следующий код:

```
procedure TFormHandBook.DBGridBookDrawColumnCell(Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
 {Если книга на руках, то}
 if DM.QAllBookONHAND.Value='1' then
    {выделяем запись синим цветом}
    with FormHandBook.DBGridBook.Canvas do
    begin
      Brush.Color:=clSkyBlue;
      FillRect(Rect);
      TextOut(Rect.Left,Rect.Top,Column.Field.Text);
    end
      else
       {иначе выводим в стандартном режиме}
       DBGridBook.DefaultDrawColumnCell(Rect,DataCol,Column,State);
end;
```

У DBGrid есть свойство Canvas, на котором можно рисовать. Процедура FillRect заполняет указанную в структуре Rect область выбранным цветом. Цвет мы указали clSkyBlue. С помощью процедуры TextOut выводится текст в заданных координатах. Все данные передаются в событие при его вызове, поэтому просто указываем переменные, остальное система сама подставит. Когда мы выводим информацию в стандартном режиме, то используем процедуру DefaultDrawColumnCell.

Для кнопки **Добавить** пишем код добавления новой записи в таблицу Book:

```
procedure TFormHandBook.ButtonAddClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {Номер записи - в данном случае значение null}
 DM.ProcModifyBook.ParamByName('bookid_par').Value:=null;
```

```
 {Название книги}
 DM.ProcModifyBook.ParamByName('nameofbook_par').Value:=
                          EditNameOfBook.Text;
 {Шифр книги}
 DM.ProcModifyBook.ParamByName('shifr_par').Value:=EditShifr.Text;
 {Содержание книги}
 DM.ProcModifyBook.ParamByName('bookabout_par').Value:=
                          MemoBookAbout.Lines.Text;
 {Признак нахождения}
 DM.ProcModifyBook.ParamByName('onhand_par').Value:='0';
 {Параметр, который укажет процедуре, что происходит
 добавление данных}
 DM.ProcModifyBook.ParamByName('choose_par').Value:=1;

 {пытаемся выполнить процедуру}
 try
  DM.ProcModifyBook.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Добавление не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllBook.Close;
 DM.QAllBook.Open;

end;
```

Здесь особых проблем возникнуть не должно, но все равно рассмотрим более подробно. Сначала заполняем входные параметры процедуры данными, взятыми из полей **Название книги**, **Шифр**, **Краткое содержание**, а также указываем, что будем добавлять данные (choose_par=1). Затем пытаемся выполнить добавление; если получилось, то подтверждаем транзакцию, если нет, то откатываем и информируем пользователя. Затем, если операцию по добавлению удалось провести, обновляем набор данных.

Для кнопки **Изменить** пишем код изменения записи в таблице Book:

```
procedure TFormHandBook.ButtonChangeClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {Номер выбранной записи}
 DM.ProcModifyBook.ParamByName('bookid_par').Value:=
                          DM.QAllBookBOOKID.Value;
 {Название книги}
 DM.ProcModifyBook.ParamByName('nameofbook_par').Value:=
                          EditNameOfBook.Text;
 {Шифр книги}
 DM.ProcModifyBook.ParamByName('shifr_par').Value:=
                          EditShifr.Text;
 {Содержание книги}
 DM.ProcModifyBook.ParamByName('bookabout_par').Value:=
                          MemoBookAbout.Lines.Text;
 {Признак нахождения}
 DM.ProcModifyBook.ParamByName('onhand_par').Value:='0';
 {Параметр, который укажет процедуре, что происходит
 изменение данных}
 DM.ProcModifyBook.ParamByName('choose_par').Value:=2;

 {пытаемся выполнить процедуру}
 try
  DM.ProcModifyBook.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Изменить запись не получилось!'+#13+
               'Повторите попытку');
```

```
    {откатываем транзакцию}
    DM.IBTransaction1.RollbackRetaining;
    {выходим из процедуры}
    exit;
  end;

  {если запрос выполнился, то подтверждаем транзакцию}
  DM.IBTransaction1.CommitRetaining;

  {обновляем набор данных}
  DM.QAllBook.Close;
  DM.QAllBook.Open;
end;
```

Для кнопки **Удалить** пишем код удаления записи из таблицы Book:

```
procedure TFormHandBook.ButtonDeleteClick(Sender: TObject);
begin
  {Передаем в параметры процедуры необходимые данные}
  {Номер выбранной записи}
  DM.ProcModifyBook.ParamByName('bookid_par').Value:=
                          DM.QAllBookBOOKID.Value;
  {Название книги, для удаления не используется}
  DM.ProcModifyBook.ParamByName('nameofbook_par').Value:=null;
  {Шифр книги, для удаления не используется}
  DM.ProcModifyBook.ParamByName('shifr_par').Value:=null;
  {Содержание книги, для удаления не используется}
  DM.ProcModifyBook.ParamByName('bookabout_par').Value:=null;
  {Признак нахождения, для удаления не используется}
  DM.ProcModifyBook.ParamByName('onhand_par').Value:=null;
  {Параметр, который укажет процедуре, что происходит
  удаление данных}
  DM.ProcModifyBook.ParamByName('choose_par').Value:=3;

  {пытаемся выполнить процедуру}
  try
   DM.ProcModifyBook.ExecProc;
```

```
 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Изменить запись не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllBook.Close;
 DM.QAllBook.Open;
end;
```

Теперь переходим на вкладку **Читатели** формы `FormHandBook`. Располагаем на ней `DBGrid`, который называем `DBGridReader`, и подключаем его к `DSAllReader`.

Теперь необходимо разместить `Label` с текстом "ФИО", под ним — разместить `Edit`, у которого нужно очистить свойство `Text` и свойство `Name` изменить на `EditFIO`.

Размещаем еще один `Label` с текстом "Паспорт", под ним располагаем `Edit`, у которого нужно очистить свойство `Text` и свойство `Name` изменить на `EditPasport`.

После этого располагаем три компонента `Button`, в свойство `Caption` которых пишем **Добавить**, **Изменить**, **Удалить**. А в свойство `Name`, соответственно, — `ButtonAddReader`, `ButtonChangeReader`, `ButtonDeleteReader`.

Вид формы с активной вкладкой **Читатели** после размещения всех компонентов представлен на рис. 5.70.

Переходим к модулю `UDM` и для компонента `QAllReader` в событии `AfterScroll` пишем код:

```
procedure TDM.QAllReaderAfterScroll(DataSet: TDataSet);
begin
 {Если форма Справочники создана, то}
 if ExistsHandBook then
```

```
 begin
   {В EditFIO помещаем фамилию читателя}
   FormHandBook.EditFIO.Text:=QAllReaderFIO.Value;
   {В EditPasport помещаем паспортные данные читателя}
   FormHandBook.EditPasport.Text:=QAllReaderPASPORT.Value;
 end;

end;
```

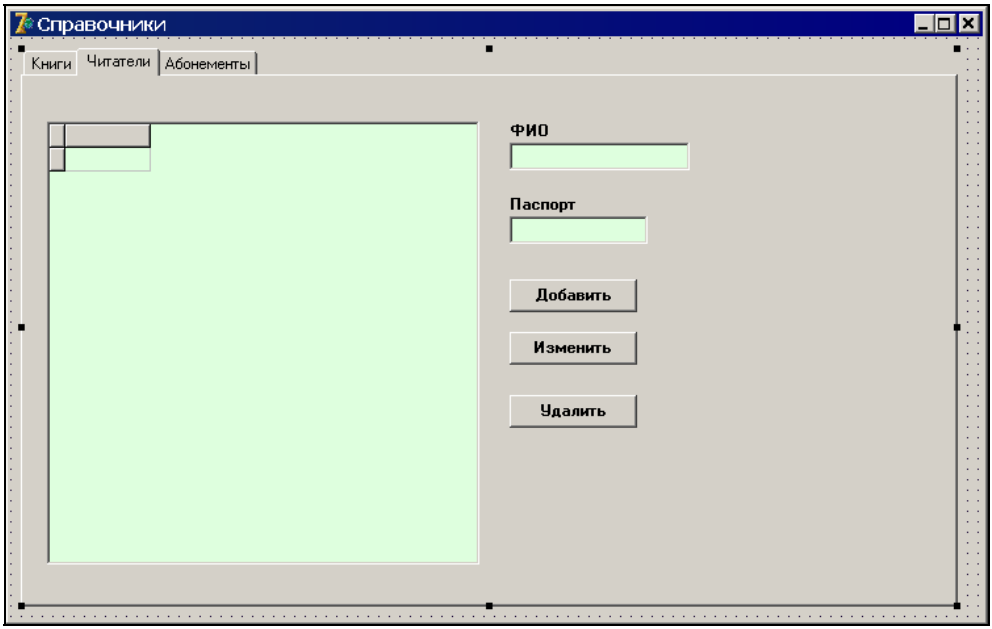


Рис. 5.70. Вид формы с активной вкладкой **Читатели**

В отличие от прошлого раза здесь мы не блокируем кнопки. Потому что проверка на возможность удаления читателя (проверяется, есть у него абонементы) реализована в самой процедуре. А изменять данные читателя можно без проблем, на целостности базы это не отразится.

Возвращаемся обратно на форму `FormHandBook` к вкладке **Читатели** и для кнопки **Добавить** пишем код добавления записи в таблицу Reader:

```
procedure TFormHandBook.ButtonAddReaderClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {Номер выбранной записи}
 DM.ProcModifyReader.ParamByName('readerid_par').Value:=null;
```

```
 {ФИО читателя}
 DM.ProcModifyReader.ParamByName('fio_par').Value:=EditFIO.Text;
 {Паспортные данные читателя}
 DM.ProcModifyReader.ParamByName('pasport_par').Value:=
                              EditPasport.Text;
 {Параметр, который укажет процедуре, что происходит
 добавление данных}
 DM.ProcModifyReader.ParamByName('choose_par').Value:=1;

 {пытаемся выполнить процедуру}
 try
  DM.ProcModifyReader.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Не получилось добавить запись!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllReader.Close;
 DM.QAllReader.Open;
end;
```

Для кнопки **Изменить** пишем код изменения записи в таблице Reader:

```
procedure TFormHandBook.ButtonChangeReaderClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {Номер выбранной записи}
 DM.ProcModifyReader.ParamByName('readerid_par').Value:=
                              DM.QAllReaderREADERID.Value;
```

```
 {ФИО читателя}
 DM.ProcModifyReader.ParamByName('fio_par').Value:=EditFIO.Text;
 {Паспортные данные читателя}
 DM.ProcModifyReader.ParamByName('pasport_par').Value:=
                              EditPasport.Text;
 {Параметр, который укажет процедуре, что происходит
 изменение данных}
 DM.ProcModifyReader.ParamByName('choose_par').Value:=2;

 {пытаемся выполнить процедуру}
 try
  DM.ProcModifyReader.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Не получилось изменить запись!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllReader.Close;
 DM.QAllReader.Open;
end;
```

Для кнопки **Удалить** пишем код удаления записи из таблицы Reader:

```
procedure TFormHandBook.ButtonDeleteReaderClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {Номер выбранной записи}
 DM.ProcModifyReader.ParamByName('readerid_par').Value:=
                              DM.QAllReaderREADERID.Value;
```

```
 DM.ProcModifyReader.ParamByName('fio_par').Value:=null;

 DM.ProcModifyReader.ParamByName('pasport_par').Value:=null;

 {Параметр, который укажет процедуре, что происходит
 удаление данных}
 DM.ProcModifyReader.ParamByName('choose_par').Value:=3;

{пытаемся выполнить процедуру}
 try
  DM.ProcModifyReader.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Изменить запись не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {Если процедура вернула 0, то удаления не было,
 информируем об этом пользователя}
 if DM.ProcModifyReader.ParamByName('result_output').Value=0 then
 begin
  ShowMessage('Данного читателя удалить невозможно, '+
              'т.к. у него есть абонемент');
  {подтверждать нечего, поэтому выходим}
  exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllReader.Close;
 DM.QAllReader.Open;
end;
```

В этой процедуре мы отслеживаем возвращаемый результат, и если он равен нулю, то значит, у читателя есть абонементы и он не удалился, поэтому мы проинформируем об этом пользователя программы.

Переходим на вкладку **Абонементы** формы `FormHandBook`. Располагаем на ней `DBGrid`, который называем `DBGridAbonement`, и подключаем его к `DSAllAbonement`.

Теперь необходимо разместить `Label` с текстом "ФИО — владельца абонемента", под ним нужно разместить `Edit`, у которого нужно очистить свойство `Text`, и свойство `Name` изменить на `EditReaderKod`. Свойство `ReadOnly` — установить в `True`.

Размещаем еще один `Label` с текстом "ФИО — для внесения изменения", под ним размещаем `Edit`, у которого нужно очистить свойство `Text` и свойство `Name` — изменить на `EditNewReaderKod`. Свойство `ReadOnly` — установить в `True`.

Размещаем еще одни `Label` с текстом "Дата окончания абонемента", под ним размещаем `DateTimePicker`.

После этого располагаем три компонента `Button`, в свойство `Caption` которых пишем "Добавить", "Изменить", "Закрыть абонемент". А в свойство `Name`, соответственно, — `ButtonAddAbonement`, `ButtonChangeAbonement`, `ButtonCloseAbonement`.

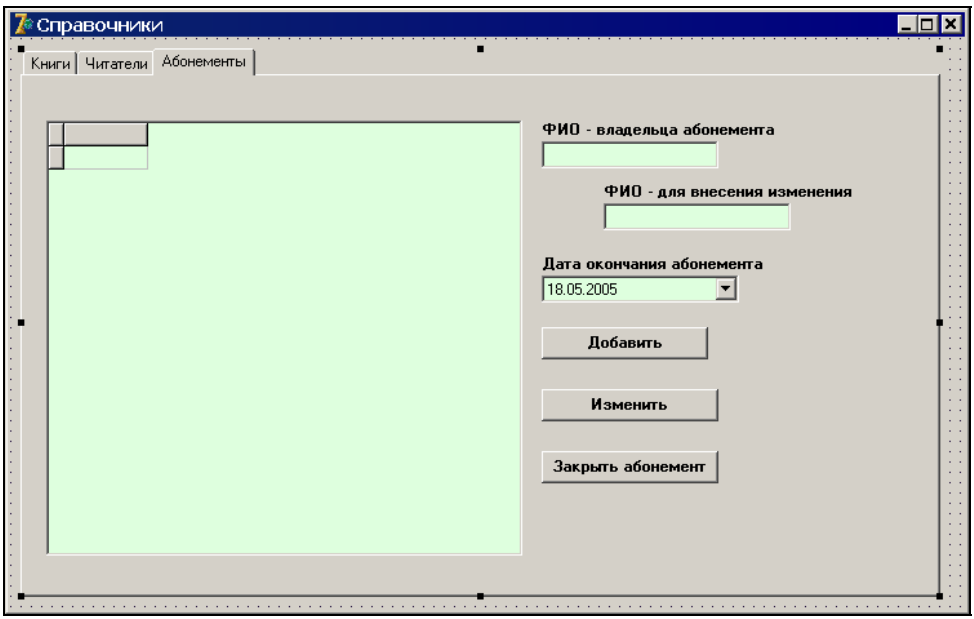


Рис. 5.71. Вид формы с активной вкладкой **Абонементы**

Вид формы с активной вкладкой **Абонементы** после размещения всех компонентов представлен на рис. 5.71.

Переходим к модулю `UDM` и для компонента `QAllAbonement` в событии `AfterScroll` пишем код:

```
procedure TDM.QAllAbonementAfterScroll(DataSet: TDataSet);
begin
 {если форма - Справочник создана, то}
 if ExistsHandBook then
 begin
  {Помещаем в EditReaderKod имя хозяина абонента}
  FormHandBook.EditReaderKod.Text:=QAllAbonementFIO.Value;
  {Помещаем в EditNewReaderKod выбранное имя читателя,
  на закладке Читатели}
  FormHandBook.EditNewReaderKod.Text:=QAllReaderFIO.Value;
  {Помещаем в свойство Tag компонента EditNewReaderKod
  код выбранного читателя}
  FormHandBook.EditNewReaderKod.Tag:=QAllReaderREADERID.Value;

  {Помещаем в DateTimePicker дату окончания абонемента}
  FormHandBook.DateTimePicker1.Date:=QAllAbonementDATEOFEND.Value;

  {Устанавливаем доступные кнопки}

  {Если абонемент просрочен, то
  данные можно только добавлять}
  if QAllAbonementPRIZNAKOFEND.Value='1' then
  {Блокируем кнопку Изменить и Закрыть абонемент}
  begin
   FormHandBook.ButtonChangeAbonement.Enabled:=false;
   FormHandBook.ButtonCloseAbonement.Enabled:=false;
  end
    else {Иначе все кнопки доступны}
  begin
   FormHandBook.ButtonChangeAbonement.Enabled:=true;
   FormHandBook.ButtonCloseAbonement.Enabled:=true;
  end;

 end;
end;
```

Переходим на форму FormHandBook и для компонента DBGridAbonement устанавливаем свойство DefaultDrawing в False, тем самым мы указываем, что будем самостоятельно следить за выводом данных на экране в компоненте DBGridAbonement. Создаем событие OnDrawColumnCell и пишем следующий код оформления DBGrid в зависимости от значения атрибута PriznakOfEnd:

```
procedure TFormHandBook.DBGridAbonementDrawColumnCell
  (Sender: TObject;
  const Rect: TRect; DataCol: Integer; Column: TColumn;
  State: TGridDrawState);
begin
 {Если абонемент закрыт, то}
 if DM.QAllAbonementPRIZNAKOFEND.Value='1' then
    {выделяем запись красным цветом}
    with FormHandBook.DBGridAbonement.Canvas do
    begin
      Brush.Color:=clRed;
      FillRect(Rect);
      TextOut(Rect.Left,Rect.Top,Column.Field.Text);
    end
      else
       {иначе выводим в стандартном режиме}
       DBGridAbonement.DefaultDrawColumnCell(Rect,DataCol,
                                          Column,State);
end;
```

Таким образом, если абонемент закрыт, то запись будет выделена красным цветом.

Теперь для кнопки **Добавить** пишем код добавления записи в таблицу Abonement:

```
procedure TFormHandBook.ButtonAddAbonementClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {для добавления не используется, поэтому null}
 DM.ProcModifyAbonement.ParamByName('abonementid_par').Value:=null;
 {код читателя, для которого
 создается абонемент}
 DM.ProcModifyAbonement.ParamByName('readerkod_par').Value:=
                                  EditNewReaderKod.Tag;
```

```
 {дата окончания абонемента}
 DM.ProcModifyAbonement.ParamByName('dateofend_par').Value:=
                                  DateTimePicker1.Date;
 {признак окончания абонемента}
 DM.ProcModifyAbonement.ParamByName('priznakofend_par').Value:='0';
 {Параметр, который укажет процедуре, что происходит
 добавление данных}
 DM.ProcModifyAbonement.ParamByName('choose_par').Value:=1;

 {пытаемся выполнить процедуру}
 try
  DM.ProcModifyAbonement.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Не получилось добавить запись!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllAbonement.Close;
 DM.QAllAbonement.Open;

end;
```

Для кнопки **Изменить** пишем код изменения записи в таблице Abonement:

```
procedure TFormHandBook.ButtonChangeAbonementClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {код абонемента}
 DM.ProcModifyAbonement.ParamByName('abonementid_par').Value:=
                          DM.QAllAbonementABONEMENTID.Value;
```

```
 {код читателя}
 DM.ProcModifyAbonement.ParamByName('readerkod_par').Value:=
                            EditNewReaderKod.Tag;
 {дата окончания абонемента}
 DM.ProcModifyAbonement.ParamByName('dateofend_par').Value:=
                            DateTimePicker1.Date;
 {признак окончания абонемента}
 DM.ProcModifyAbonement.ParamByName('priznakofend_par').Value:='0';
 {Параметр, который укажет процедуре, что происходит
 изменение данных}
 DM.ProcModifyAbonement.ParamByName('choose_par').Value:=2;

 {пытаемся выполнить процедуру}
 try
  DM.ProcModifyAbonement.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Не получилось изменить запись!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllAbonement.Close;
 DM.QAllAbonement.Open;
end;
```

Для кнопки **Закрыть абонемент** пишем код:

```
procedure TFormHandBook.ButtonCloseAbonementClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {код абонемента}
```

```
 DM.ProcCloseAbonement.ParamByName('abonementid_par').Value:=
                                  DM.QAllAbonementABONEMENTID.Value;
 {пытаемся выполнить процедуру}
 try
  DM.ProcCloseAbonement.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Закрыть абонемент не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {Если процедура вернула 0, то абонемент
 закрыть нельзя, т.к. на нем есть
 несданные книги}
 if DM.ProcCloseAbonement.ParamByName('result_output').Value=0 then
 begin
  ShowMessage('Абонемент закрыть нельзя, '+
              'т.к. на нем есть несданные книги');
  {подтверждать нечего, поэтому выходим}
  exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QAllAbonement.Close;
 DM.QAllAbonement.Open;
end;
```

В данной процедуре мы проверяем возвращенный результат: если он нулевой, то значит, на данном абонементе есть несданные книги и закрывать его нельзя. Поэтому мы информируем об этом пользователя.

Для события OnClose формы FormBookHand пишем код закрытия наборов данных:

```
procedure TFormHandBook.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
 DM.QAllBook.Close;
 DM.QAllReader.Close;
 DM.QAllAbonement.Close;
end;
```

Замечание

Закрывая форму, необходимо закрыть и наборы данных на случай, если вдруг мы захотим их использовать в других формах.

Работа с формой FormHandBook завершена. Переходим на форму MainForm, подключаем модули UHandBook, UDM и для пункта меню **Справочник** пишем код:

```
procedure TMainForm.N4Click(Sender: TObject);
begin
 {Создаем форму}
 FormHandBook:=TFormHandBook.Create(self);
 {Открываем наборы данных,
 для работы с таблицами}
 DM.QAllBook.Open;
 DM.QAllReader.Open;
 DM.QAllAbonement.Open;

 {Указываем, что форма создана}
 ExistsHandBook:=true;

 {Искусственно вызываем событие AfterScroll
 для наборов данных, чтобы
 заполнились данными необходимые элементы формы}
 DM.QAllBook.AfterScroll(nil);
 DM.QAllReader.AfterScroll(nil);
 DM.QAllAbonement.AfterScroll(nil);

 {Отображаем форму}
 FormHandBook.ShowModal;
end;
```

5.8.2. Разработка формы "Внимание"

Для этой формы нам потребуется еще два набора данных. Переходим на модуль UDM и располагаем на нем два компонента IBQuery. Один называем QBackAbonement, другой — QBackBook. Подключаем их к DBLibrary.

В свойстве SQL компонента QBackAbonement пишем запрос выбора просроченных абонементов:

```
SELECT * FROM ABONEMENT,READER
WHERE
(Abonement.DateOfEnd<:ParDateOfEnd  AND  Abonement.PriznakOfEnd='0')
AND
(Abonement.ReaderKod=Reader.ReaderID)

ORDER BY Abonement.AbonementID
```

В данном запросе выбираются просроченные абонементы и читатели, которым принадлежат эти абонементы. В запрос передается текущая дата. Сортировка производится по номеру абонемента.

В свойстве SQL компонента QBackBook пишем запрос выбора книг, которые не возвращены в срок:

```
SELECT * FROM MOVE,BOOK,ABONEMENT
WHERE
(DateOfReturn<:ParDateOfReturn  AND PriznakOfReturn='0')
AND
(Move.AbonementKod=Abonement.AbonementID)
AND
(Move.BookKod=Book.BookID)

ORDER BY BOOK.NameOfBook
```

В данном запросе выбираются книги, не возвращенные в срок, а также абонементы, на которые записаны данные книги. В запрос передается текущая дата. Сортировка производится по названию книги.

Для обоих компонентов IBQuery необходимо создать объекты-столбцы в редакторе столбцов и настроить их на свое усмотрение.

Теперь необходимо разместить на модуле данных два компонента DataSource и назвать их DSBackAbonement и DSBackBook. После этого связать их с одноименными наборами данных.

Создаем новую форму, называем `FormAttention`, в свойство `Caption` пишем "!!!Внимание!!!" Сохраняем под именем `UAttention`, удаляем ее из списка автоматически создаваемых форм. Подключаем модуль `UDM`.

Размещаем на ней `Label` с текстом "Просроченные абонементы", под ним — `DBGrid`, который связываем с `DSBackAbonement`. Рядом располагаем кнопку `Button` с текстом "Закрыть абонемент" и названием `ButtonCloseAbonement`.

Помещаем на форму еще один `Label` с текстом "Просроченные книги", под ним — `DBGrid`, настроеный на `DSBackBook`, под ним — кнопку с текстом "Вернуть книгу" и названием `ButtonReturnBook`. Вид формы "!!!Внимание!!!" показан на рис. 5.72.

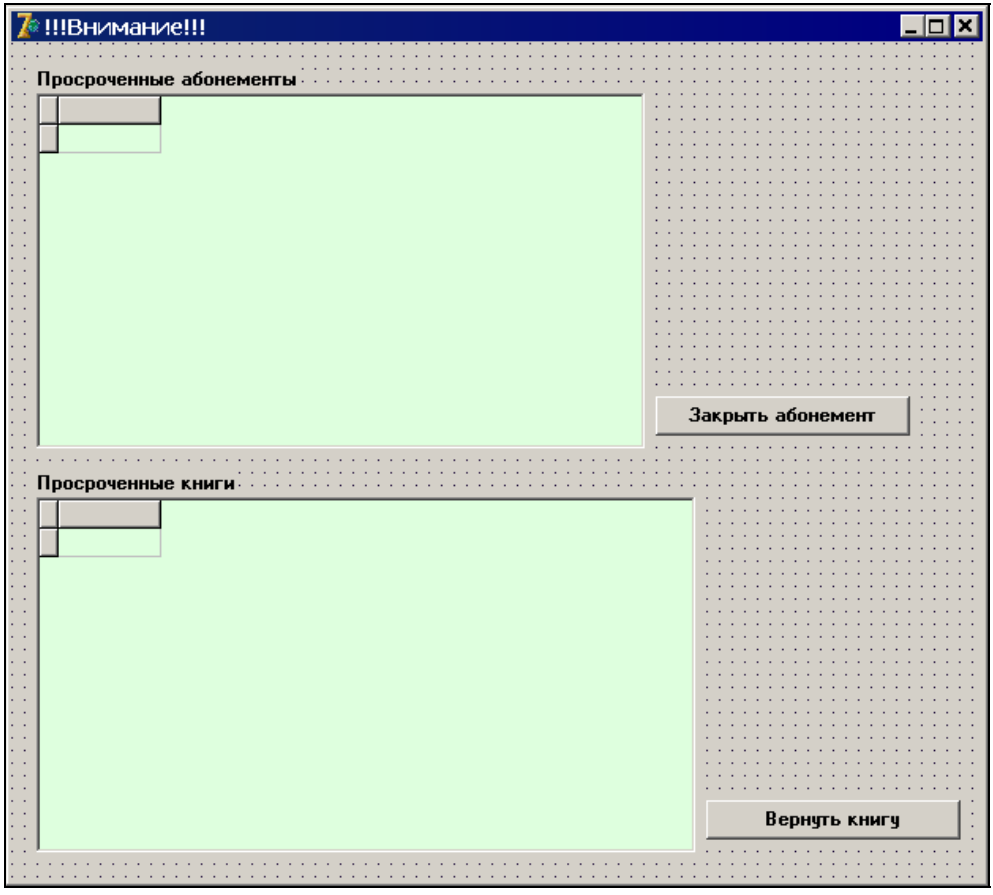


Рис. 5.72. Вид формы **!!!Внимание!!!**

Переходим на форму `MainForm`, подключаем к ней модуль `UAttention` и для пункта меню **Работа с данными \ !!!Внимание!!!** пишем код выбора книг, которые не возвращены в срок, а также просроченных абонементов:

```
procedure TMainForm.N2Click(Sender: TObject);
begin
 {Создание формы}
 FormAttention:=TFormAttention.Create(self);
 {Передаем в параметр для обоих запросов текущую дату
 и после этого открываем их}
 {1--запрос}
 DM.QBackAbonement.Params.ParamByName('ParDateOfEnd').Value:=Date;
 DM.QBackAbonement.Open;
 {2--запрос}
 DM.QBackBook.Params.ParamByName('ParDateOfReturn').Value:=Date;
 DM.QBackBook.Open;

 {Отображаем форму}
 FormAttention.ShowModal;

end;
```

Возвращаемся обратно на форму `FormAttention` и для события `OnClose` пишем код для закрытия наборов данных при закрытии формы:

```
procedure TFormAttention.FormClose(Sender: TObject;
  var Action: TCloseAction);
begin
 DM.QBackAbonement.Close;
 DM.QBackBook.Close;
end;
```

Теперь для кнопки **Закрыть абонемент** пишем код закрытия просроченного абонемента:

```
procedure TFormAttention.ButtonCloseAbonementClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {Номер абонемента}
 DM.ProcCloseAbonement.ParamByName('abonementid_par').Value:=
                                DM.QBackAbonementABONEMENTID.Value;
```

```
 {пытаемся выполнить процедуру}
 try
  DM.ProcCloseAbonement.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Выполнить действие не удалось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {Если на абонементе есть не возвращенные книги,
 то закрывать нельзя, поэтому информируем об этом пользователя}
 if DM.ProcCloseAbonement.ParamByName('result_output').Value=0 then
 begin
  ShowMessage('Абонемент нельзя закрыть, '+
              'так как на нем есть невозвращенные книги');
  exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QBackAbonement.Close;
 DM.QBackAbonement.Open;
end;
```

Для кнопки **Вернуть книгу** пишем код возвращения просроченной книги:

```
procedure TFormAttention.ButtonReturnBookClick(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {код книги}
 DM.ProcReturnBook.ParamByName('bookid_par').Value:=
                                    DM.QBackBookBOOKKOD.Value;
```

```
 {код абонемента}
 DM.ProcReturnBook.ParamByName('abonementkod__par').Value:=
                                    DM.QBackBookABONEMENTKOD.Value;

 {пытаемся выполнить процедуру}
 try
  DM.ProcReturnBook.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Операцию произвести не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем набор данных}
 DM.QBackBook.Close;
 DM.QBackBook.Open;
end;
```

5.8.3. Разработка формы "Выдача\Возврат книг"

Для этой формы нам потребуется еще два набора данных. Переходим на модуль `UDM` и располагаем на нем два компонента `IBQuery`. Один называем `QAllBookInLibrary`, другой — `QAllBookInReader`. Подключаем их к `DBLibrary`. В свойстве `SQL` компонента `QAllBookInLibrary` пишем запрос выбора книг, находящихся в библиотеке:

```
SELECT * FROM BOOK
WHERE  OnHand='0'
```

В свойстве SQL компонента QAllBookInReader пишем запрос выбора книг, которые находятся у читателей:

```
SELECT * FROM BOOK, MOVE
WHERE
    (OnHand='1')
          AND
    (Book.BookID=Move.BookKod)
         AND
    (Move.PriznakOfReturn='0')
```

Для обоих компонентов IBQuery необходимо создать объекты-столбцы в редакторе столбцов и настроить их на свое усмотрение.

Теперь необходимо разместить на модуле данных два компонента DataSource и назвать их DSAllBookInLibrary и DSAllBookInReader. После этого связать их с одноименными наборами данных.

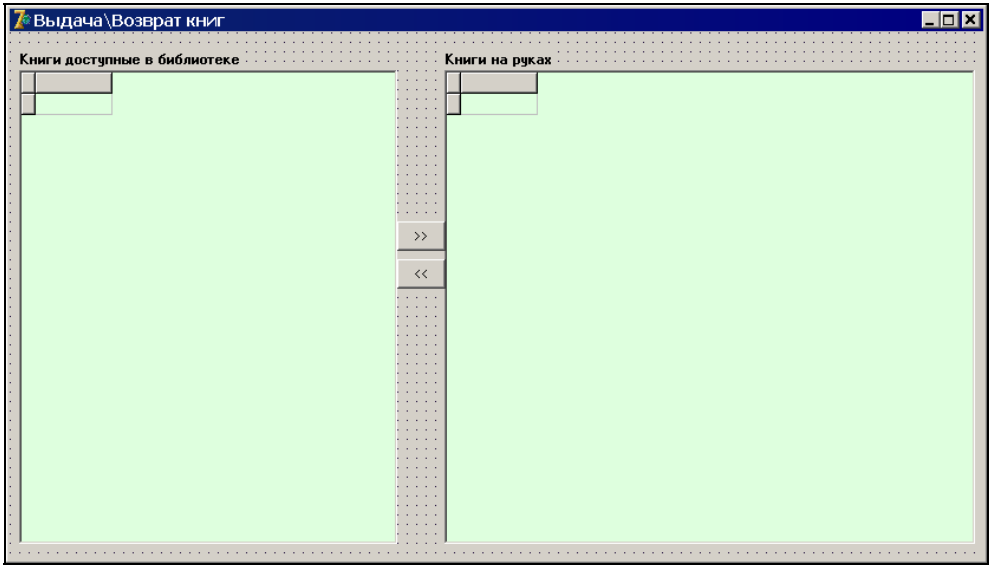


Рис. 5.73. Вид формы **Выдача\Возврат книг**

Создаем новую форму, называем ее FormMoveBook, в свойство Caption пишем "Выдача\Возврат книг". Сохраняем ее под именем UMoveForm, удаляем ее из списка автоматически создаваемых форм. Подключаем модуль UDM. Размещаем на ней Label с текстом "Книги, доступные в библиотеке", под ним — DBGrid, который связываем с DSAllBookInLibrary. Помещаем еще

один Label с текстом "Книги на руках", под ним — DBGrid, настроенный на DSBookInReader. Помещаем две кнопки, в свойство Caption первой пишем ">>", второй — "<<". На рис. 5.73 представлен рисунок формы.

Для кнопки >> пишем код выдачи книги читателю на руки:

```
procedure TFormMoveBook.Button1Click(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {код книги}
 DM.ProcGiveBook.ParamByName('bookkod_par').Value:=
                    DM.QAllBookInLibraryBOOKID.Value;

 {Используем стандартную функцию InputBox,
 которая выводит на экран окно с полем для
 ввода информации}
 {Таким образом, мы запрашиваем код абонемента
 и дату возврата книги
 и сразу присваиваем значения в параметры процедуры}
 DM.ProcGiveBook.ParamByName('abonementkod_par').Value:=
       InputBox('Введите данные','Введите код абонемента','');

 DM.ProcGiveBook.ParamByName('dateofreturn_par').Value:=
       InputBox('Введите данные',
                'Введите дату возврата в формате дд.мм.гг.','');

 {пытаемся выполнить процедуру}
 try
  DM.ProcGiveBook.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Провести операцию не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
   {выходим из процедуры}
   exit;
 end;
```

```
 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем оба набора данных}
 DM.QAllBookInLibrary.Close;
 DM.QAllBookInLibrary.Open;

 DM.QAllBookInReader.Close;
 DM.QAllBookInReader.Open;
end;
```

Замечание

Скажу сразу, что получать данные с помощью `InputBox` не совсем корректно, так как пользователь может ввести несуществующий номер абонемента, также можно ввести дату в неправильном формате. Но это учебный пример, поэтому ничего страшного нет, тем более нет ничего сложного в том, чтобы самостоятельно усовершенствовать механизм выдачи книг.

Для кнопки << пишем код возврата книги в библиотеку:

```
procedure TFormMoveBook.Button2Click(Sender: TObject);
begin
 {Передаем в параметры процедуры необходимые данные}
 {код книги}
 DM.ProcReturnBook.ParamByName('bookid_par').Value:=
                          DM.QAllBookInReaderBOOKID.Value;
 {код абонемента}
 DM.ProcReturnBook.ParamByName('abonementkod_par').Value:=
                          DM.QAllBookInReaderABONEMENTKOD.Value;

 {пытаемся выполнить процедуру}
 try
  DM.ProcReturnBook.ExecProc;

 {если не удалось, то сообщаем об ошибке}
 except
   ShowMessage('Операцию произвести не получилось!'+#13+
               'Повторите попытку');
   {откатываем транзакцию}
   DM.IBTransaction1.RollbackRetaining;
```

```
   {выходим из процедуры}
   exit;
 end;

 {если запрос выполнился, то подтверждаем транзакцию}
 DM.IBTransaction1.CommitRetaining;

 {обновляем оба набора данных}
 DM.QAllBookInReader.Close;
 DM.QAllBookInReader.Open;

 DM.QAllBookInLibrary.Close;
 DM.QAllBookInLibrary.Open;

end;
```

Для события `OnClose` пишем код:

```
DM.QAllBookInLibrary.Close;
DM.QAllBookInReader.Close;
```

Переходим на форму `MainForm`, подключаем модуль `UMoveBook` и для пункта меню **Работа с данными \ Выдача\Возврат книг** пишем код:

```
procedure TMainForm.N3Click(Sender: TObject);
begin
 FormMoveBook:=TFormMoveBook.Create(self);
 DM.QAllBookInLibrary.Open;
 DM.QAllBookInReader.Open;
 FormMoveBook.ShowModal;
end;
```

Необходимо создать обработчик события `OnShow` для вкладки **Абонементы**, иначе поле **ФИО — для внесения изменений** не будет заполняться автоматически при активизации вкладки. Код приведен ниже:

```
procedure TFormHandBook.TabSheet3Show(Sender: TObject);
begin
 DM.QAllAbonement.AfterScroll(nil)
end;
```

Замечание

Реализация пункта меню **Аналитика** и его подпунктов оставлено на самостоятельную реализацию читателем.

5.8.4. Разработка модуля читателя

Создаем новое приложение, форму называем `ReaderModuleForm`, в ее свойство `Caption` пишем "Модуль читателя". Сохраняем форму под именем `UReaderModule`, а проект под именем `ReaderModule_proj`.

Размещаем на форме три компонента `Button` с текстом "Информация", "Поиск в библиотеке" и "Выход" (рис. 5.74).

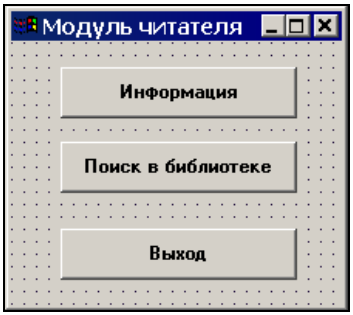


Рис. 5.74. Вид формы **Модуль читателя**

Для кнопки **Выход** пишем код выхода из приложения:

```
Application.Terminate;
```

Создаем `DataModule`, называем его `DMReaderModule`, сохраняем под именем `UDMReaderModule`. Располагаем на нем компонент `IBDatabase`, называем его `DBLibrary`, настраиваем на базу LIBRARY.FDB.

Далее располагаем компонент `IBTransaction`, который связываем с `DBLibrary` и 2 штуки `IBQuery`. Называем их `QBookHistory`, `QBookInLibrary`. Для `QBookHistory` в свойстве `SQL` пишем запрос для выбора всех книг определенного читателя, у которого они находятся на руках:

```
SELECT * FROM BOOK, MOVE
WHERE
          (Move.BookKod=Book.BookID)
                      AND
          (Move.AbonementKod=:Par)
                     AND
          (Move.PriznakOfReturn='0')
```

Для `QBookInLibrary` в свойстве `SQL` пишем запрос для выбора всех книг, которые находятся в библиотеке:

```
SELECT * FROM BOOK
WHERE OnHand='0'
```

Располагаем 2 компонента DataSource, называем их DSQBookHistory и DSBookInLibrary, связываем их с одноименными компонентами IBQuery (рис. 5.75).

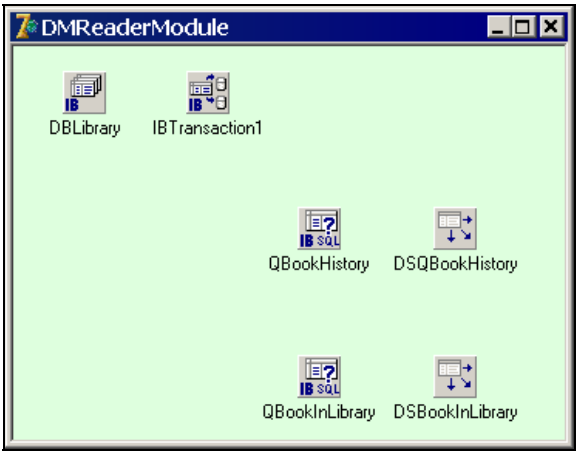


Рис. 5.75. Вид модуля данных

Нам надо, чтобы пользователь проходил аутентификацию перед входом в программу. Этим и займемся сейчас. Создаем новую форму, называем ее FormSecurity. Сохраняем под именем USecurity, удаляем ее из списка автоматически создаваемых форм. Подключаем модули UDMReaderModule и UReaderModule. Помещаем на форму 2 штуки Label, 2 штуки Edit, IBQuery, 2 Button (рис. 5.76).

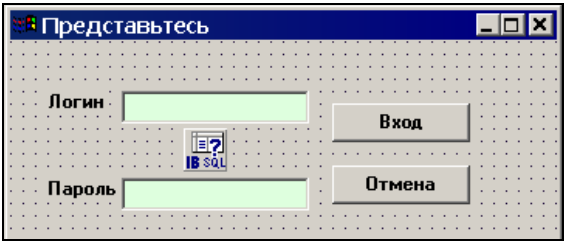


Рис. 5.76. Вид формы **Представьтесь**

IBQuery переименовываем в QSecurity, настраиваем его на DBLibrary. В свойстве SQL пишем запрос проверки данных, введенных пользователем, для аутентификации:

```
SELECT * FROM READER, ABONEMENT
WHERE
```

```
(Reader.FIO=:ParFio)
        AND
(Abonement.AbonementID=:ParAbonementID)
        AND
(Abonement.ReaderKod=Reader.ReaderID)
```

Данный запрос осуществляет проверку правильности введенных данных пользователем; если все верно, то запрос вернет запись, иначе — количество возращенных записей будет нулевым. Необходимо создать объекты-столбцы в редакторе столбцов.

Для кнопки **Вход** пишем код:

```
procedure TFormSecurity.Button1Click(Sender: TObject);
begin
 {Задаем значение параметров, для запроса,
 первые - логин, второй - пароль}
 with QSecurity.Params do
 begin
   ParamByName('ParFIO').Value:=Edit1.Text;
   ParamByName('ParAbonementID').Value:=StrToInt(Edit2.Text);
 end;

 {Выполняем запрос}
 QSecurity.Open;

 {Если количество возращенных данных запросом больше 0,
 значит, пользователь ввел правильные данные, количество
 возвращенных записей хранится в свойстве RecordCount}
 if  QSecurity.RecordCount>0 then
 begin
  {Запоминаем код пользователя}
  id_user:=QSecurityREADERID.Value;
  {Запоминаем фамилию пользователя}
  FIO_user:=QSecurityFIO.Value;
  {Запоминаем код пользователя}
  id_abonement:=QSecurityABONEMENTID.Value;

  {Устанавливаем признак прохождения аутентификации}
  CheckSecurity:=true;

  QSecurity.Close;
```

```
  {Закрываем окно аутентификации}
  FormSecurity.Close;
  {Выходим из процедуры, т.к. дальше обрабатывать нечего}
  exit;
 end;

 {Если пользователь ввел неправильные данные,
 то проинформировать его об этом}
 ShowMessage('В доступе отказано');
 {Закрываем набор данных}
 QSecurity.Close;

end;
```

Для кнопки **Отмена** пишем код закрытия формы `FormSecurity`:

```
FormSecurity.Close;
```

В модуле `UModuleReader` объявляем глобальные переменные:

```
var
  ReaderModuleForm: TReaderModuleForm;

  CheckSecurity:boolean=false;
  {Данные пользователя,
  код, ФИО, код абонемента}
  id_user:integer=0;
  FIO_user:string='';
  id_abonement:integer=0;
```

Переменная `CheckSecurity` будет отвечать за то, прошел ли пользователь аутентификацию или нет. Для формы `ReaderModuleForm` в событии `OnShow` пишем код вызова окна аутентификации:

```
procedure TReaderModuleForm.FormShow(Sender: TObject);
begin
 {Открываем связь с базой данных}
 DMReaderModule.DBLibrary.Open;

 {Если пользователь не прошел
 аутентификацию, то}
 if CheckSecurity=false then
```

```
 begin
   {Создаем форму аутентификации}
   FormSecurity:=TFormSecurity.Create(self);
   FormSecurity.ShowModal;
   {Если пользователь отказался от проверки, то
   закрываем приложение}
   if CheckSecurity=false then Application.Terminate
 end;
end;
```

Создаем новую форму, называем ее `InformationForm`. В свойство `Caption` пишем "Информация". Сохраняем ее под именем `UInformationForm`, удаляем ее из списка автоматически создаваемых форм. Подключаем к ней модуль `UDMReaderModule`. Размещаем на ней компоненты, как показано на рис. 5.77.

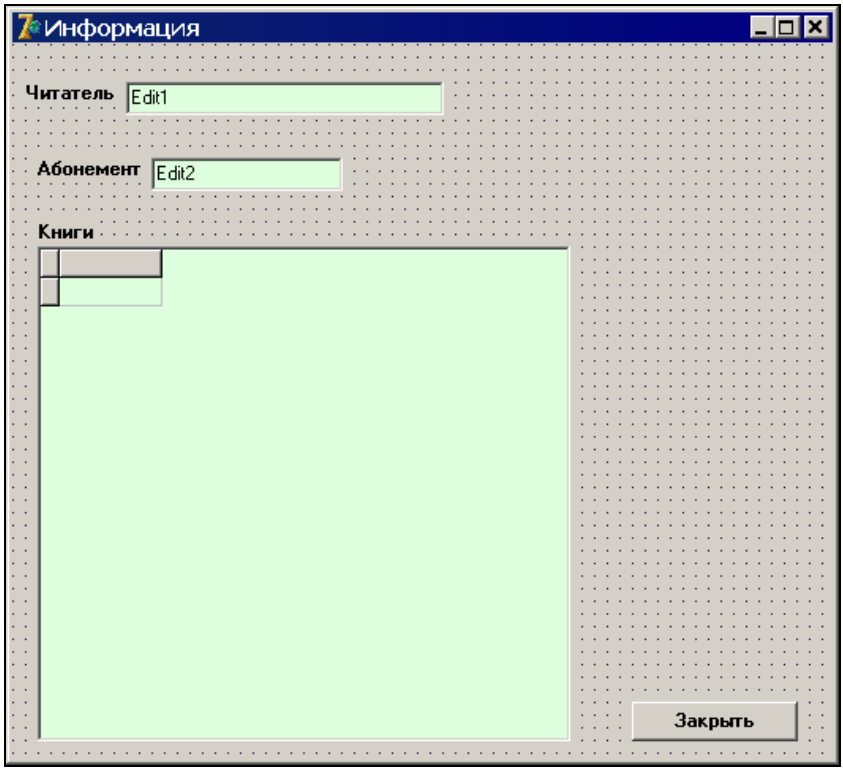


Рис. 5.77. Вид формы **Информация**

DBGrid связываем с DSBookHistory. Для кнопки **Закрыть** пишем код закрытия формы InformationForm:

```
InformationForm.Close;
```

Переходим к форме ReaderModuleForm и для кнопки **Информация** пишем код вызова окна **Информация**:

```
procedure TReaderModuleForm.Button1Click(Sender: TObject);
begin
 InformationForm:=TInformationForm.Create(self);
 {Задаем абонемент, относительно
 которого будут отображаться книги}
 DMReaderModule.QBookHistory.Params.
     ParamByName('Par').Value:=id_abonement;
 DMReaderModule.QBookHistory.Open;

 {Выводим пользовательские данные}
 InformationForm.Edit1.Text:=FIO_user;
 InformationForm.Edit2.Text:=IntToStr(id_abonement);

 InformationForm.ShowModal;
end;
```

Создаем новую форму, называем ее LibraryBookForm, в свойство Caption пишем "Поиск книг в библиотеке". Сохраняем ее под именем ULibraryBook, удаляем ее из списка автоматически создаваемых форм. Подключаем к ней модуль UDMReaderModule. Размещаем на ней компоненты, как показано на рис. 5.78.

DBGrid связываем с DSBookInLibrary. Для кнопки **Закрыть** пишем код закрытия формы LibraryBookForm:

```
LibraryBookForm.Close;
```

Для компонента Edit в событии OnChange пишем следующий код поиска книги по первым введенным символам:

```
DMReaderModule.QBookInLibrary.

Locate('NameOfBook',Edit1.Text,[loCaseInsensitive,loPartialKey]);
```

Но чтобы Delphi не ругалась на этот код, нужно подключить модуль DB, потому что константы loCaseInsensitive, loPartialKey описаны именно в нем. Теперь при вводе в Edit данных с клавиатуры автоматически будет происходить поиск названия книги в наборе данных QBookInLibrary.

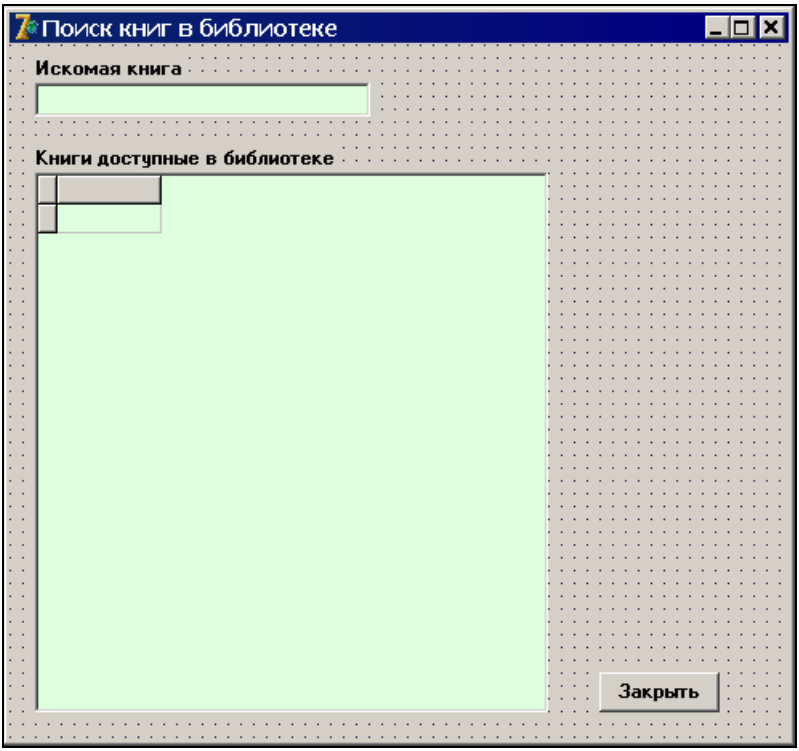


Рис. 5.78. Вид формы **Поиск книг в библиотеке**

Переходим к форме `ReaderModuleForm` и для кнопки **Поиск в библиотеке** пишем код вызова окна **Поиск в библиотеке**:

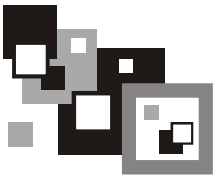
```
LibraryBookForm:=TLibraryBookForm.Create(self);
DMReaderModule.QBookInLibrary.Open;
LibraryBookForm.ShowModal;
```

На компакт-диске в каталоге ГЛАВА 5\SOURCE можно увидеть пример данной программы.

В каталоге ГЛАВА 5\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 5\VIDEO можно посмотреть видеоролики, посвященные созданию разрабатываемого приложения.

Глава 6



QReport

6.1. Вступление

QReport — это самая простейшая система построения отчетов, представленная отдельными компонентами вкладки **QReport**. Она обладает минимальными возможностями и удобствами, но в то же время представляет неплохой инструмент для быстрого построения несложных отчетов. К тому же эта система включена в Delphi, начиная с первой версии.

6.2. Установка QReport в Delphi 7

Начиная с 7 версии в Delphi по умолчанию отсутствует вкладка **QReport**, на которой размещены компоненты для построения отчетов, поддерживаемые этой системой. Для установки компонентов QReport необходимо запустить Delphi, выбрать пункт меню **Component\Install Packages** — появится окно **Project Options** (рис. 6.1).

Нажимаем кнопку **Add**. В появившемся окне **Add Design Package** необходимо выбрать файл для установки компонентов, данный файл находится в папке, в которую была проинсталлирована Delphi, по умолчанию — C:\Program Files\Borland\Delphi7\Bin\DCLQRT70.BPL, и нажать кнопку **Открыть**. В группе элементов **Design packages** окна **Project Options** появится новая запись — **QuickReport Components** (рис. 6.2).

Нажимаем **OK**. Теперь в палитре компонентов должна появиться новая вкладка под названием **QReport** (рис. 6.3).

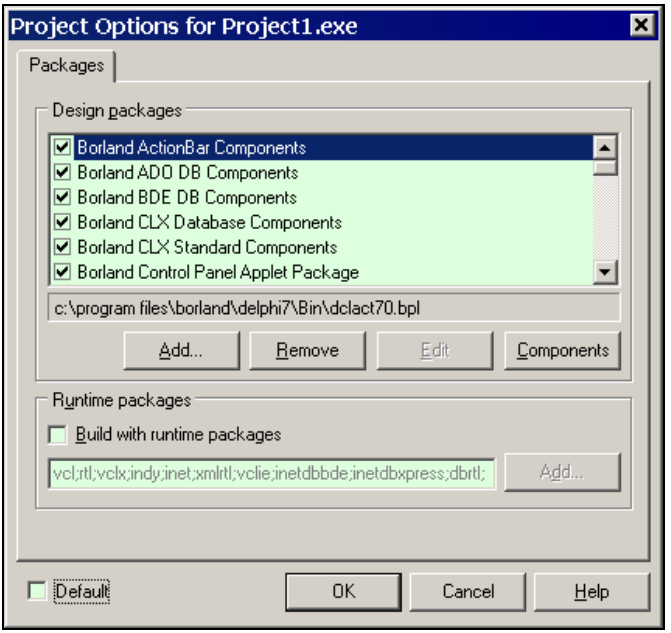


Рис. 6.1. Окно **Project Options**

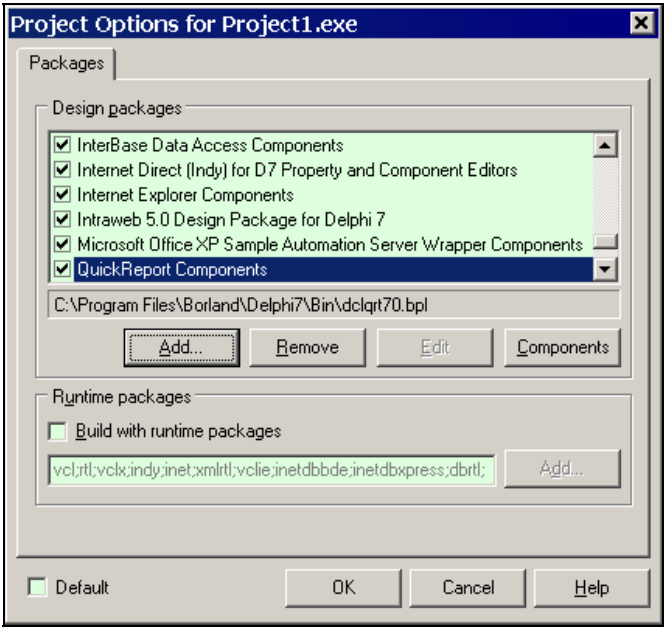


Рис. 6.2. Окно **Project Options** с добавленными компонентами QuickReport

Рис. 6.3. Вкладка **QReport** в палитре компонентов Delphi 7

6.3. Установка QReport в Delphi 2005

Для установки QReport в Delphi 2005 необходимо скачать дополнительный дистрибутив данной системы. Файл называется QReportPro_v4[1].05_Delphi2005.rar, скачать его можно с сайта **http://rapidshare.de/files/1930007/QReportPro_v4.05_Delphi2005.rar.html** — там немного запутанный интерфейс, но разобраться можно.

Информацию о QReport можно почерпнуть с официального сайта: **http://www.qusoft.com/**, там также можно скачать данную систему (файл QR405PDX.EXE), но для этого необходимо быть зарегистрированным пользователем и знать пароль.

Необходимо запустить файл qr405pdx.exe, после чего появится мастер установки, в котором следует выполнить несколько нажатий на кнопку **Next**, все предложенные параметры можно оставлять по умолчанию. После чего QReport будет установлен в папку C:\Program Files\QuickeportsX (если, конечно, предложенный по умолчанию путь не был изменен).

Теперь необходимо добавить QReport в палитру компонентов. Для этого в Delphi 2005 выбираем пункт меню **Component\Install Packages**. В появившемся окне нажимаем кнопку **Add**. Появится окно **Add Design Package**, в котором нам надо выбрать файл QR4DesignDX.bpl.

Если при установке QReport путь был оставлен по умолчанию, то данный файл будет находиться по адресу C:\Program Files\QuickeportsX\bpl. После того как файл указан, необходимо нажать **OK**, теперь в Delphi должна появиться вкладка **QReport** (она будет видна, только если тип создаваемого приложения **for Win32**) — рис. 6.4.

Сразу можно заметить, что по сравнению с QReport в 7 версии Delphi, в Delphi 2005 появились новые фильтры (фильтры — компоненты для преобразования отчета в другие форматы — более подробное описание можно найти в *разд. 6.5* данной главы).

Замечание

Также появилась приличная справка, которая содержится в файле QuickReport_Help.chm, который располагается по адресу C:\Program Files\QuickeportsX\help. Справка на английском языке, но при наличии элементарных знаний можно легко разобраться.

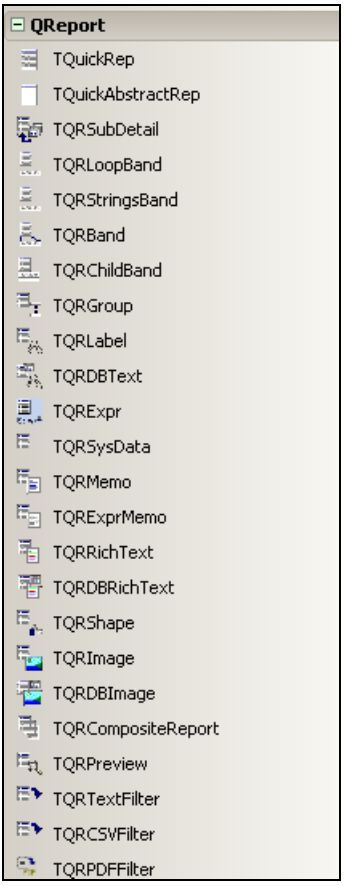


Рис. 6.4. Вкладка **QReport** в палитре компонентов Delphi 2005

Для того чтобы Delphi смог компилировать программы, в которых используются компоненты QReport, необходимо настроить рабочие директории, то есть указать каталоги, в которых располагается дистрибутив QReport. Выбираем пункт **Project\Options** главного меню. На экране появится окно, в котором выбираем пункт **Directories/Conditionals** (рис. 6.5).

Нажимаем кнопку с тремя точками, расположенную напротив поля **Search path**, — появится окно **Directories**, в которое надо добавить следующие пути (рис. 6.6):

- C:\Program Files\QuickeportsX
- C:\Program Files\QuickeportsX\lib
- C:\Program Files\QuickeportsX\quickrpt

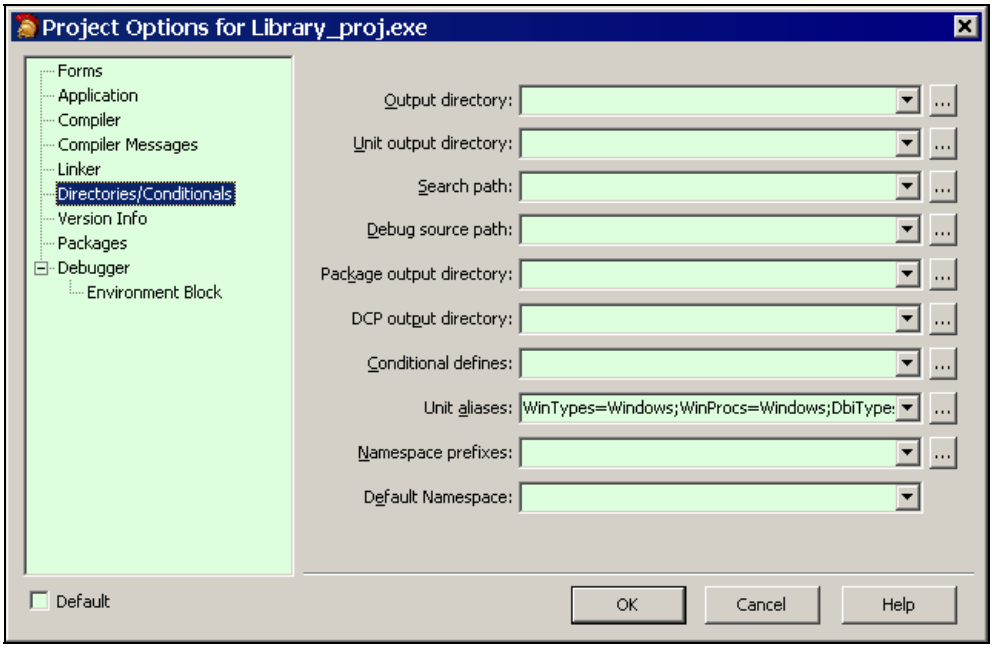


Рис. 6.5. Окно для настроек опций Delphi 2005

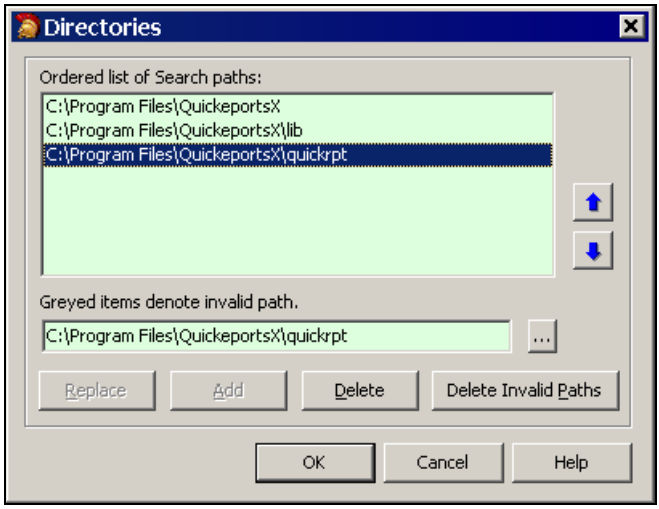


Рис. 6.6. Окно настроек рабочих директорий

Также можно заметить различия в окне просмотра отчета, в QReport для Delphi 2005 оно более удобное, чем в Delphi 7 (рис. 6.7 и 6.8).

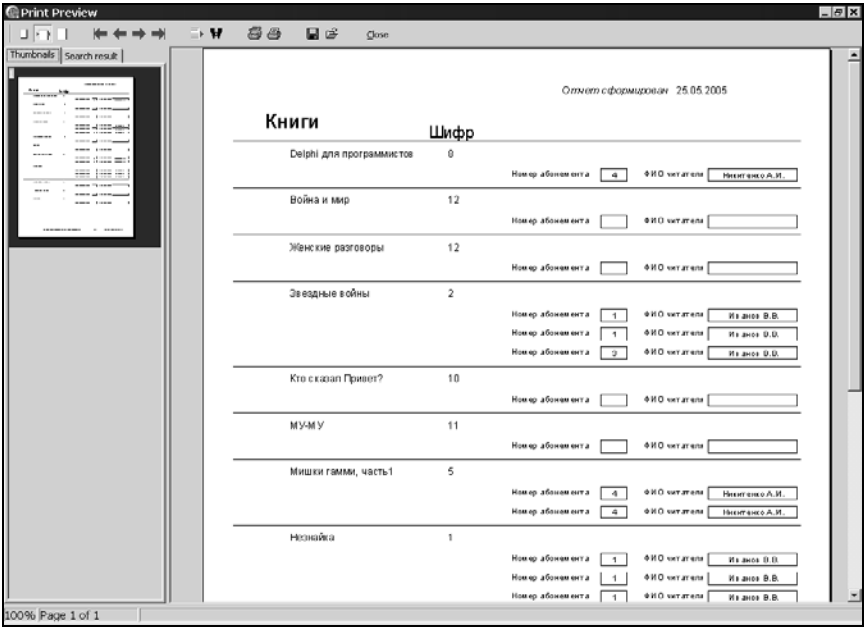


Рис. 6.7. Окно просмотра отчета **Print Preview** в Delphi 2005

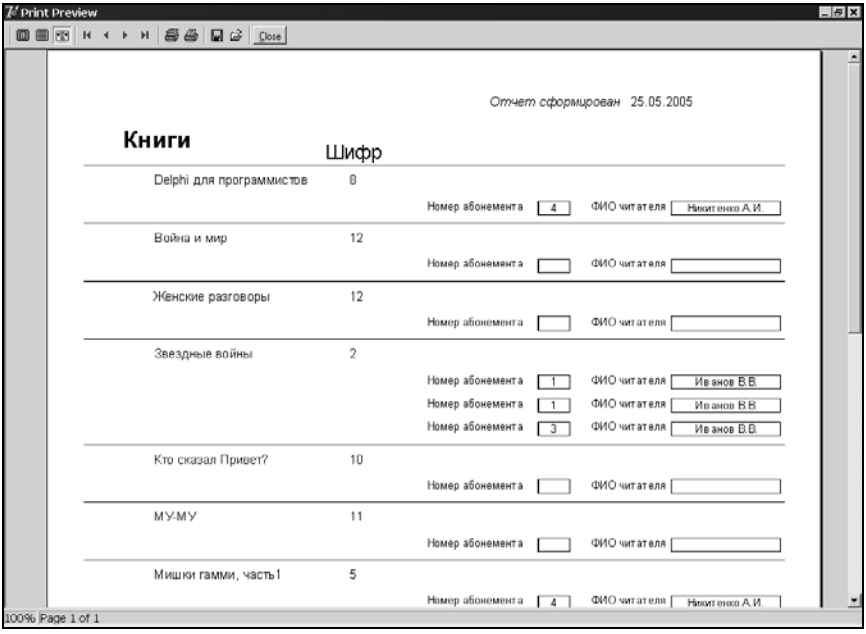


Рис. 6.8. Окно просмотра отчета **Print Preview** в Delphi 7

6.4. Использование QReport

Попробуем немного развить проект Библиотекарь, рассмотренный в прошлой главе, добавим к нему отчет. Начнем с самого простейшего: выведем всех читателей библиотеки, упорядоченных по имени.

Открываем проект `Library_proj`, разработанный в *главе 5*. Создаем новую форму, называем ее `FormSimpleRep`, сохраняем под именем `USimpleRep`. Удаляем форму из списка автоматически создаваемых форм, подключаем модуль `UDM`.

Помещаем на нее компонент `IBQuery`, называем его `QRepBook`, подключаем его к `DBLibrary`. В свойстве `SQL` пишем запрос выбора всех книг из таблицы BOOK:

```
SELECT NameOfBook FROM BOOK
ORDER BY NameOfBook
```

Запрос выбирает все названия книг из таблицы BOOK и упорядочивает их в алфавитном порядке.

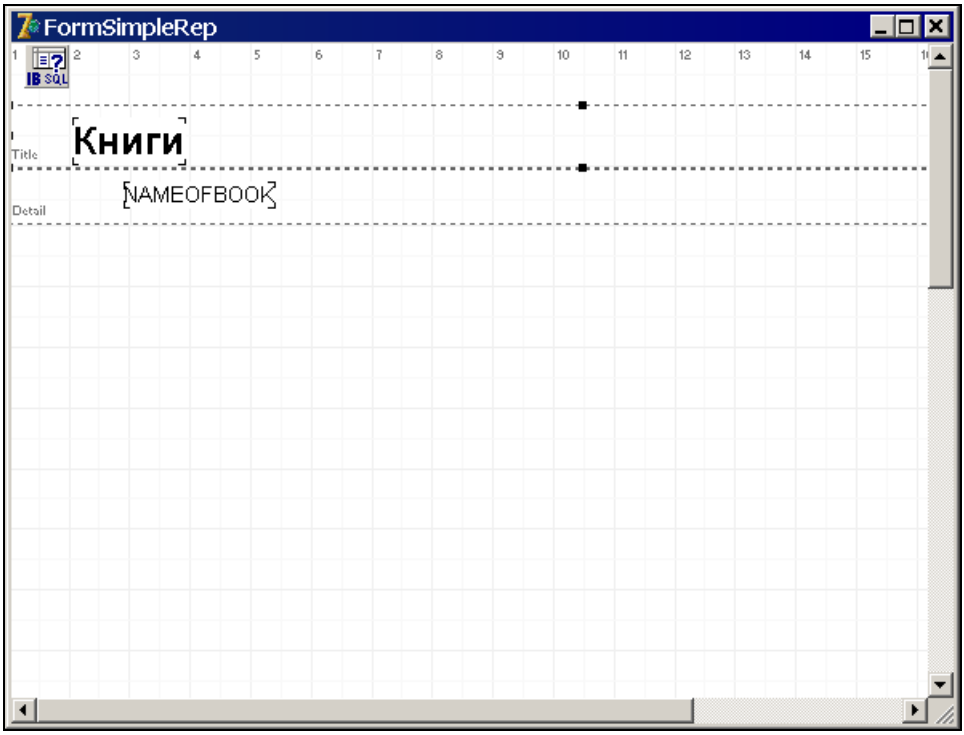


Рис. 6.9. Простой отчет

Далее располагаем QuckRep, его свойство DataSet настраиваем на QRepBook. Раскрываем его свойство Bands и в подсвойство HasDetail устанавливаем True. В компоненте QuickRep появится дополнительный элемент — прямоугольник, который называется секцией или *бендом*. Помещаем на прямоугольник компонент QRDBText, в свойстве DataSet указываем QRepBook (связываем с набором данных), в свойстве DataField указываем NameOfBook.

Устанавливаем в True подсвойство HasTitle свойства Bands компонента QuickRep. В компоненте QuickRep появится еще один бенд. Помещаем на него компонент QRLabel, в свойстве Caption пишем "Книги". Устанавливаем с помощью свойства Font жирный шрифт размера 18 (рис. 6.9).

Переходим на форму MainForm, производим двойной щелчок левой кнопкой мыши по компоненту MainMenu и добавляем новый пункт меню **Отчет\ Простой отчет** (рис. 6.10).

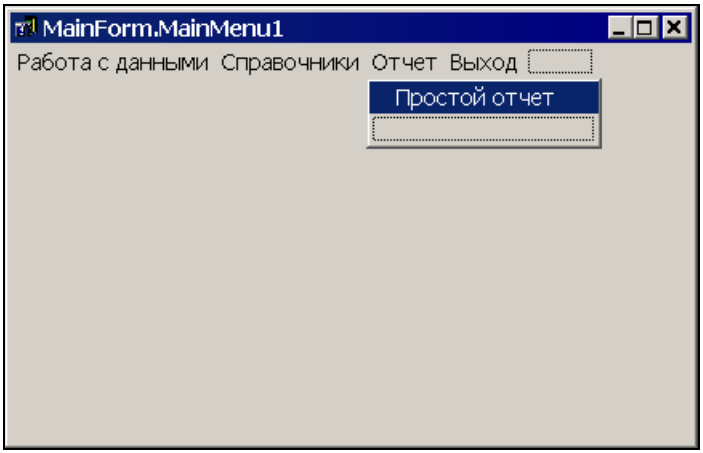


Рис. 6.10. Добавление нового пункта меню

Для пункта меню **Простой отчет** пишем код вывода отчета на экран (предварительно нужно будет подключить модуль USimpleRep):

```
{Создаем форму, содержащую отчет}
FormSimpleRep:=TFormSimpleRep.Create(self);
{Открываем набор данных}
FormSimpleRep.QRepBook.Open;

{Выводим отчет на экран}
FormSimpleRep.QuickRep1.Preview;
```

```
{После того, как пользователь закроет
отчет, начнется выполняться этот участок кода}
{Закрываем набор данных}
FormSimpleRep.QRepBook.Close;
{Закрываем форму}
FormSimpleRep.Close;
```

Компилируем программу, выбираем пункт **Отчет\Простой отчет** и на экране появляется окно **Print Preview**, содержащее готовый отчет (рис. 6.11).

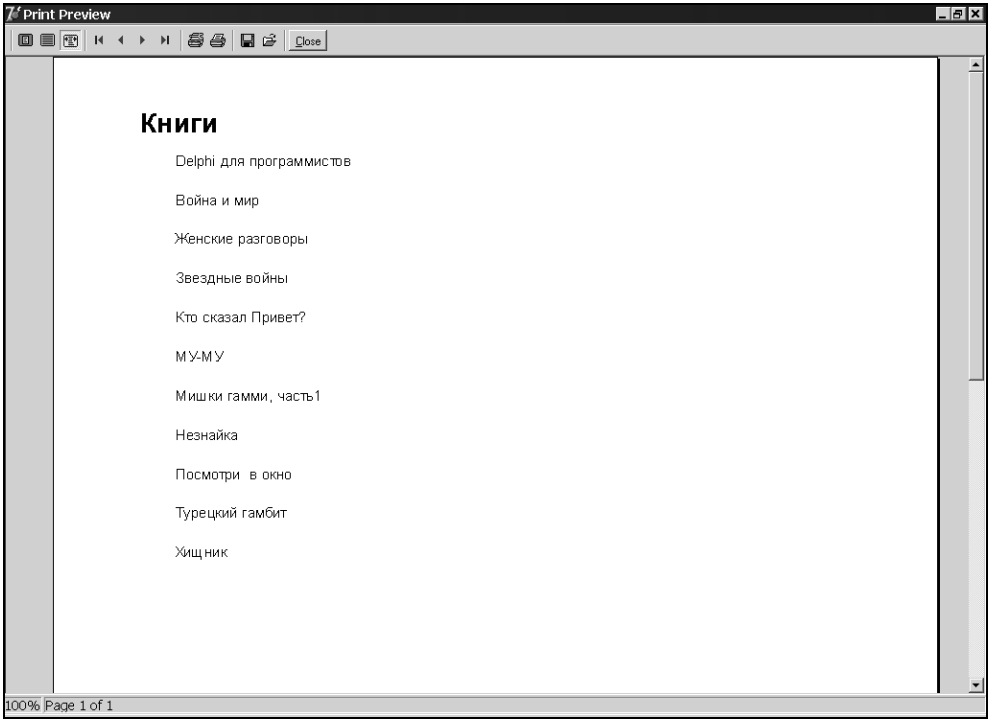


Рис. 6.11. Готовый отчет

В верхней части окна **Print Preview** расположена панель инструментов, предназначенная для смены масштаба, перелистывания страниц, сохранения отчета и др. Остальную часть занимает сам отчет (в Delphi 2005 окно **Print Preview** будет выглядеть немного по-другому).

6.5. Принцип построения отчета

Принцип построения отчета заключается в следующем. Сначала размещается компонент `QuickReport` — это главный компонент отчета. Как форма является контейнером для объектов, так же и `QuickReport` является контейнером для элементов отчета. У этого компонента задается свойство `DataSet`, которое должно указывать на основной источник данных. Например, если строится отчет типа master-detail, то для вывода информации из главной таблицы и подчиненной ей необходимо свойство `DataSet` настраивать на главную таблицу, речь идет о компоненте `Table` или `Query`, который возвращает набор данных из главной таблицы).

Далее задается структура будущего отчета с помощью бендов или секций. У каждого бенда есть свой тип, который указывает, за какой элемент отчета он отвечает, например, бенд типа `Title` отвечает за титульную часть отчета; данные, которые он отображает, размещаются один раз в начале отчета.

В свою очередь на каждом бенде располагаются компоненты для отображения данных, так как бенд — это просто площадка определенного типа. Как правило, в роли такого компонента выступает `QRDBText`, у которого указывается источник данных — свойство `DataSet` — и атрибут, данные которого будут отображаться, — свойство `DataField`. Кроме того, для отображения данных могут использоваться специальные компоненты, которые могут производить операции над данными и, соответственно, у них могут быть заданы параметры отображения: цвет, шрифт, наличие рамки и т. д.

Для доступа к данным базы выступают точно те же компоненты, что и используются при построении приложений: `Query`, `Table` и т. д., то есть они являются посредником между отчетом и базой данных.

Бенды могут быть следующих типов:

- `Column Header` — заголовок атрибутов. Когда в отчете печатается таблица, то данная секция выступает в роли шапки, где, как правило, содержатся название атрибутов таблицы. Вместо этого бенда можно использовать бенд `Title`;
- `Detail` — подробности. В данном бенде отображаются строки таблицы или набора данных, это достигается тем, что на бенде располагаются компоненты `QRDBText` или аналогичные ему, у которых настраиваются свойства `DataSet` и `DataField`. И далее эта секция в отчете отображается столько раз, сколько записей в используемом наборе данных;
- `Sub Detail` — дополнительные подробности. Данный бенд можно добавить только с помощью палитры компонентов, а именно размещением на `QuickRep` компонента `QRSubDetail`. Данная секция используется, когда требуется отображать данные из связанной таблицы, то есть на секции `Detail` отображаются данные из главной таблицы — `Master`, а на секции

`Sub Detail` — данные из таблицы, связанной с главной. Таким образом строится отчет типа master-detail;

- `Page Footer` — нижний колонтитул;
- `Page Header` — верхний колонтитул;
- `Summary` — информация данной секции печатается один раз на последней странице отчета. Бенд используется для отображения аналитической информации, такой как, количество записей, сумма по какому-либо атрибуту и т. д.;
- `Title` — заголовок. Секция предназначена для отображения заголовка отчета.

Бенды можно размещать двумя способами:

- воспользоваться палитрой компонентов. Разместить компонент `QRBand` на форме, затем настроить его свойство `BandType`, выбрав из выпадающего списка то значение, которое необходимо;

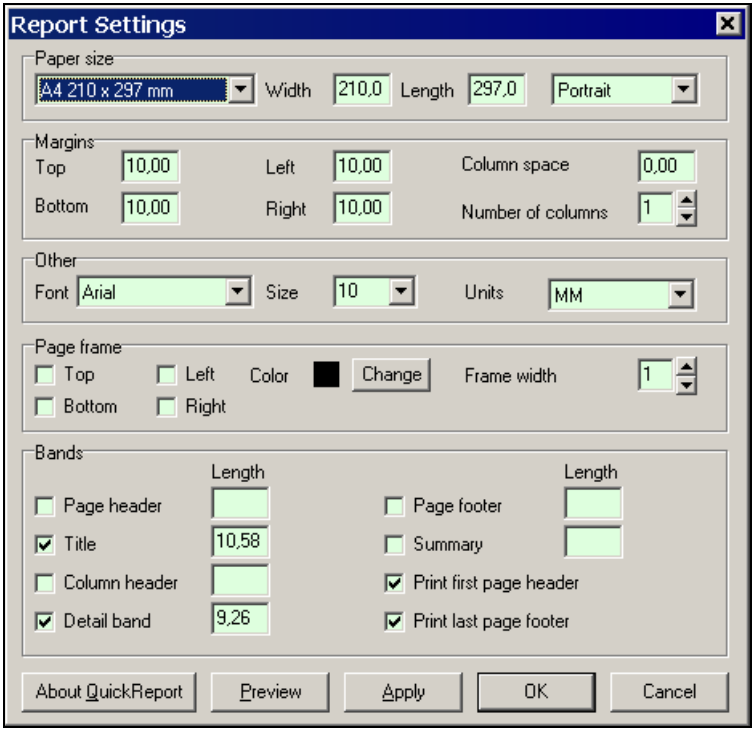


Рис. 6.12. Окно **Report Settings**

- у компонента `QuickRep` раскрыть свойство `Bands` и установить `True` напротив тех бендов, которые должны присутствовать в отчете (имя бенда имеет приставку `Has` в инспекторе объектов). Нужно учесть, что бенд `Sub Detail` таким образом разместить не удастся.

Если произвести двойной щелчок левой кнопкой мыши по компоненту `QuickRep`, то появится окно **Report Settings** для настроек параметров отчета (рис. 6.12).

Для того чтобы вывести отчет на экран, необходимо вызвать метод `Preview`. В общем случае код выглядит следующим образом:

```
QuickRep.Preview;
```

6.6. Усложняем пример

Добавляем бенд `PageHeader` к уже существующему отчету, на нем располагаем компонент `QRLabel` с текстом "Отчет сформирован" и установленным типом шрифта курсив.

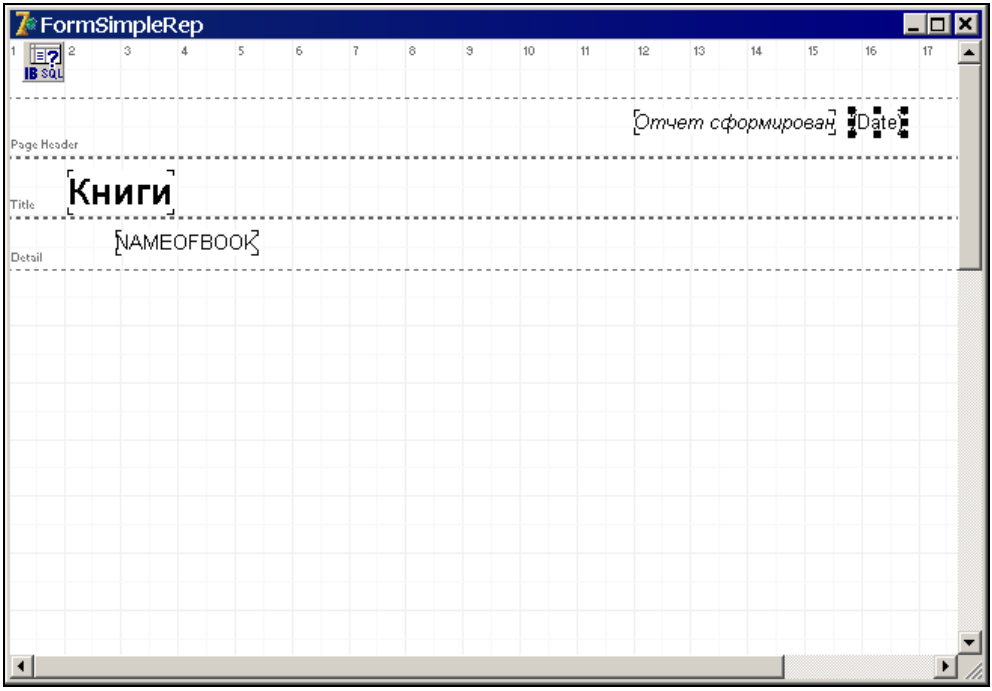


Рис. 6.13. Форма проектирования отчета

Рядом с ним располагаем компонент QRSysData, с помощью которого можно выводить дополнительную системную информацию и информацию по отчету. В свойстве Data устанавливаем qrsDate, тем самым указывая, что в этом месте отчета будет выводиться системная дата (рис. 6.13).

Идем дальше. Теперь будем выводить под названием книги номера абонементов, на которые она была хоть раз записана. Для этого, в начале, изменим запрос в QRepBook. Необходимо, чтобы он был следующим:

```
SELECT BookID, NameOfBook, Shifr FROM BOOK
ORDER BY NameOfBook
```

Необходимо создать объекты-столбцы в редакторе столбцов. На бенд Title добавляем компонент QRLabel, в свойстве Caption пишем "Шифр". На бенд Detail добавляем компонент QRDBText, который связываем с набором данных QRepBook и атрибутом SHIFR.

Теперь необходимо разместить компонент IBTable, называем его TRepMove, связываем с базой данных DBLibrary. В свойстве TableName выбираем Move. Необходимо создать для него объекты-столбцы в редакторе столбцов.

Следующим действием нам надо будет связать QRepBook и TRepMove по коду книги. Для этого располагаем компонент DataSource рядом с QRepBook, называем его DSRepBook, в свойство DataSet устанавливаем QRepBook. Далее в свойство MasterSource компонента TRepMove устанавливаем DSRepBook. Открываем свойство MasterFields и в появившемся окне **Field Link Designer** в списке **Detail Fields** выбираем атрибут BOOKKOD, а в списке **Master Fields** выбираем элемент BOOKID (рис. 6.14).

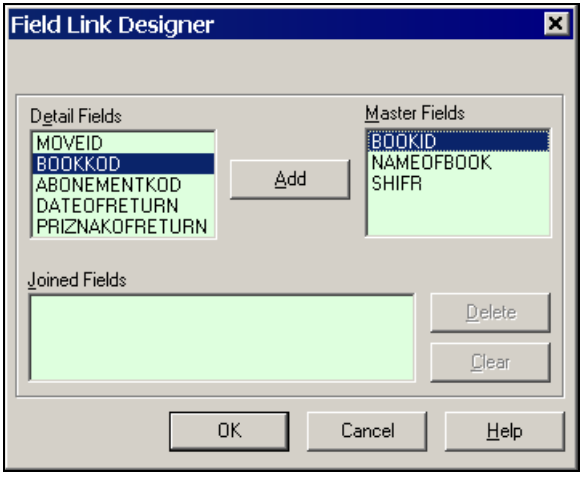


Рис. 6.14. Окно **Field Link Designer**

Нажимаем кнопку **Add**, в поле **Joined Fields** появится запись вида BOOKKOD —> BOOKID, которая указывает на то, что мы связали два набора данных. Нажимаем **OK**.

Добавляем еще один бенд с помощью компонента `QRSubDetail`. Размещаем на нем `QRLabel` с текстом "Номер абонемента", а также `QRDBText`, который связываем с набором данных `TRepMove` и атрибутом `AbonementKod`. В свойство `DataSet` бенда `Sub Detail` устанавливаем `TRepMove`.

Теперь в свойстве `Frame` компонента `QRDBText`, который располагается на бенде `Sub Detail`, устанавливаем `True` для подсвойств `DrawBotton`, `DrawLeft`, `DrawRight`, `DrawTop`, тем самым указывая, что данный компонент будет окружен рамкой. Свойство `AutoSize` устанавливаем в `False`, свойство `Aligment` в `taCenter`. Высоту бенда `Sub Detal` надо сделать чуть больше высоты компонента `QRDBText`, размещенного на нем, чтобы бенд занимал чуть больше одной строчки. Это нужно для экономии места в отчете (рис. 6.15).

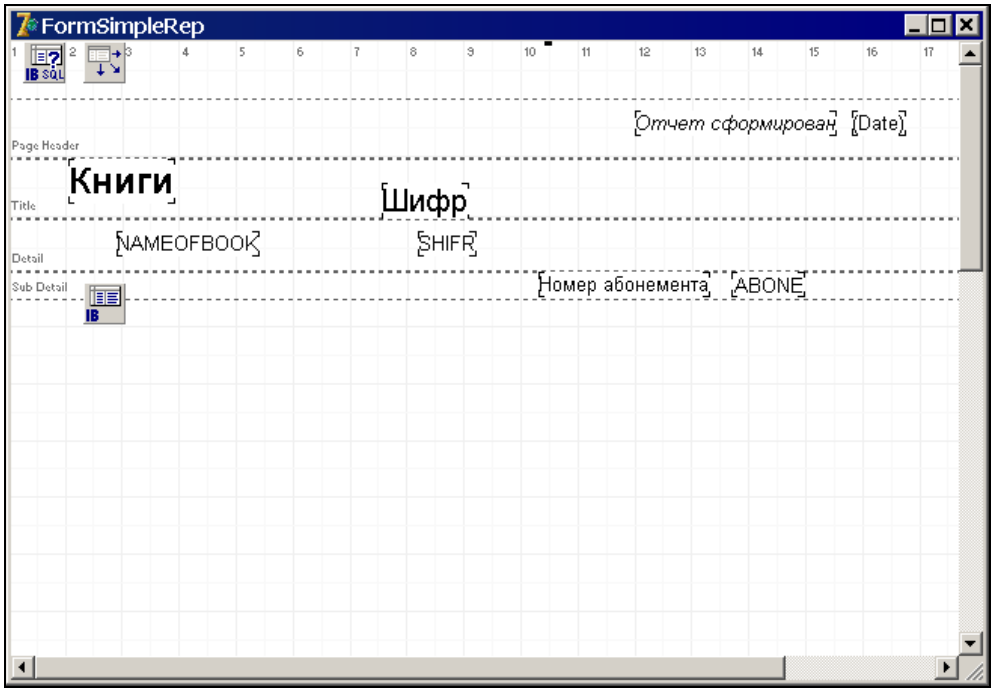


Рис. 6.15. Установка высоты бенда в отчете

Еще несколько штрихов. Добавляем бенд `Page Footer`, у которого устанавливаем в `True` подсвойство `DrawTop` свойства `Frame`. Теперь размещаем на нем `QRLabel` с текстом "Общее количество книг в библиотеке", а также

QRSysData в свойстве Data, в котором выбираем qrsDetailCount, тем самым мы будем отображать количество записей в главной таблице.

Размещаем на том же бенде еще один компонент QRLabel с текстом "на", а также компонент QRSysData в свойстве Data, у которого выбираем qrsDateTime, тем самым мы будем отображать дату и время формирования отчета.

У бенда Detail устанавливаем в True подсвойство DrawTop свойства Frame. Окончательный вариант формы отчета можно увидеть на рис. 6.16.

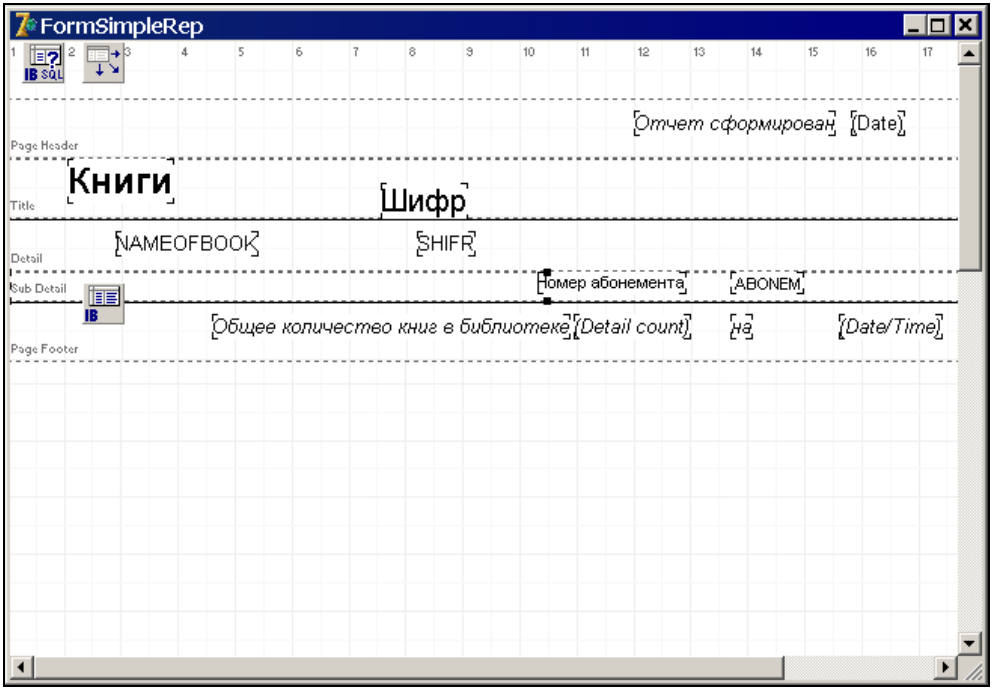


Рис. 6.16. Окончательный вариант формы отчета

Теперь немного изменим процедуру для пункта меню **Отчет\Простой отчет**:

```
{Создаем форму, содержащую отчет}
FormSimpleRep:=TFormSimpleRep.Create(self);

{Открываем наборы данных}
FormSimpleRep.QRepBook.Open;
FormSimpleRep.TRepMove.Open;
```

```
{Выводим отчет на экран}
FormSimpleRep.QuickRep1.Preview;

{После того как пользователь закроет
отчет, начнется выполняться этот участок кода}
{Закрываем набор данных}
FormSimpleRep.TRepMove.Close;
FormSimpleRep.QRepBook.Close;
{Закрываем форму}
FormSimpleRep.Close;
```

Запускаем приложение, выбираем пункт меню **Отчет\Простой отчет** и наслаждаемся полученным результатом (рис. 6.17).

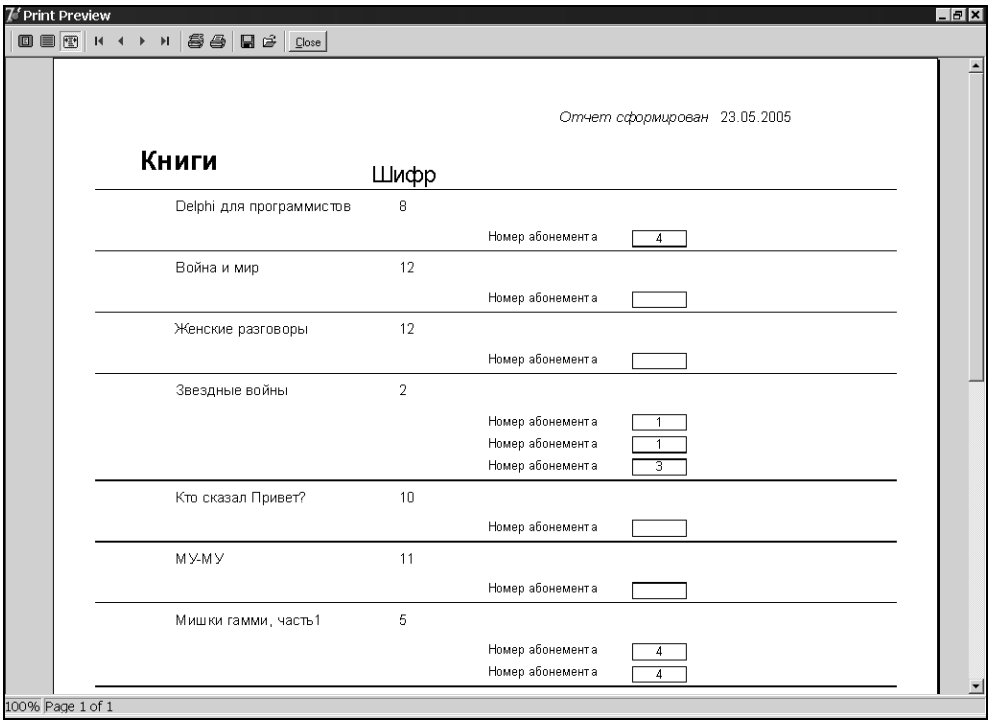


Рис. 6.17. Готовый отчет

6.7. Фильтры

Если во время выполнения программы в сформированном отчете в панели инструментов нажать кнопку сохранения с пиктограммой дискеты, то появится диалоговое окно **Save report**. Если попытаться выбрать формат для сохранения в выпадающем списке **Тип файла**, то можно увидеть, что сохранить отчет можно только в формате QReport — QRP (рис. 6.18).

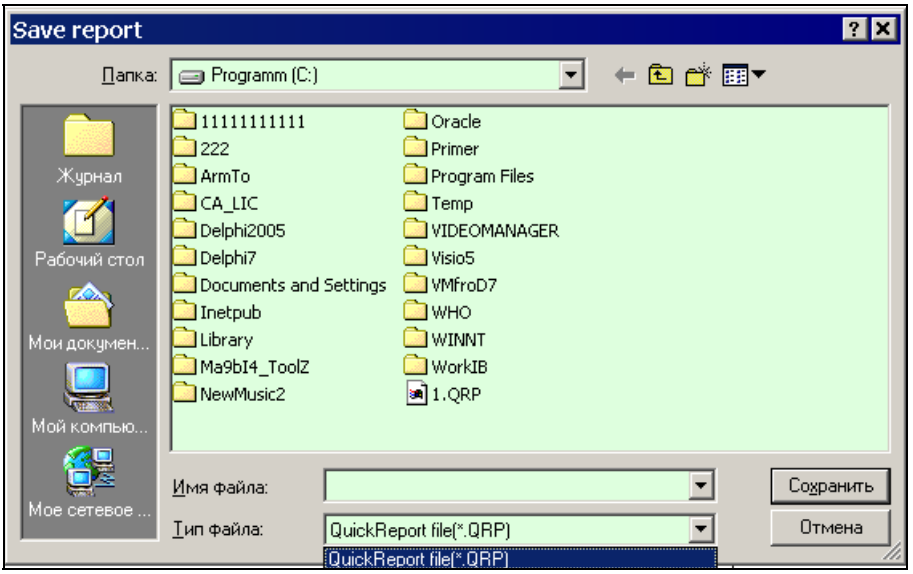


Рис. 6.18. Окно **Save report**

Замечание

Если вам захочется просмотреть сохраненный отчет, то можно нажать кнопку **Open** с пиктограммой открывающейся папки в панели инструментов в сформированном отчете.

Но часто бывают случаи, когда отчет надо сохранить в другом формате, который можно было бы использовать в других программах, например, Microsoft Word или Excel, и желательно, чтобы этот процесс происходил при минимуме написания кода. Для этого существуют фильтры — это компоненты, которые позволяют сохранять отчет в различных форматах. Для этого их просто надо разместить на компоненте `QuickRep` и новые возможности станут доступны.

К этим компонентам относятся:

- `QRTextFilter` — сохранение отчета в текстовом формате;
- `QRCSVFilter` — сохранение отчета в текстовом формате с разделителями, в роли разделителя по умолчанию выступает запятая, но это можно изменить с помощью свойства `Separator` данного компонента. Этот формат, как правило, используется для конвертации данных из отчета в базу данных;
- `QRHTMLFilter` — сохранение отчета в формате HTML.

Размещаем все эти три компонента на компоненте `QuickRep`, который расположен на форме `FormSimpleRep` (рис. 6.19).

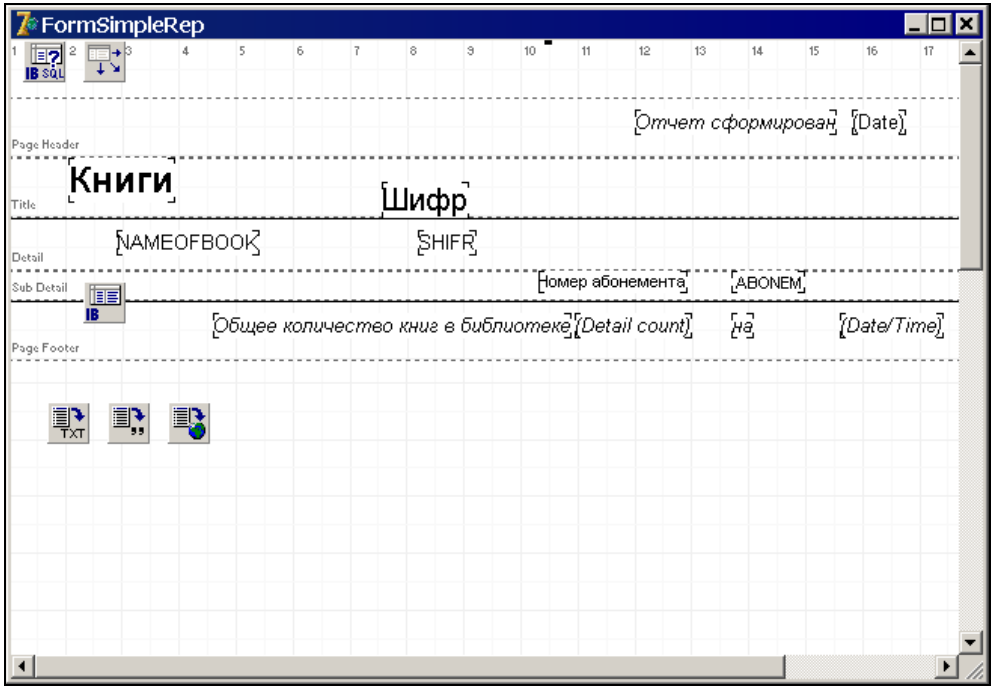


Рис. 6.19. Отчет с размещенными на нем фильтрами

Теперь запускаем приложение, формируем отчет и нажимает кнопку **Save** с изображением дискеты, открываем выпадающий список **Тип файла** и видим, что количество возможных форматов увеличилось (рис. 6.20).

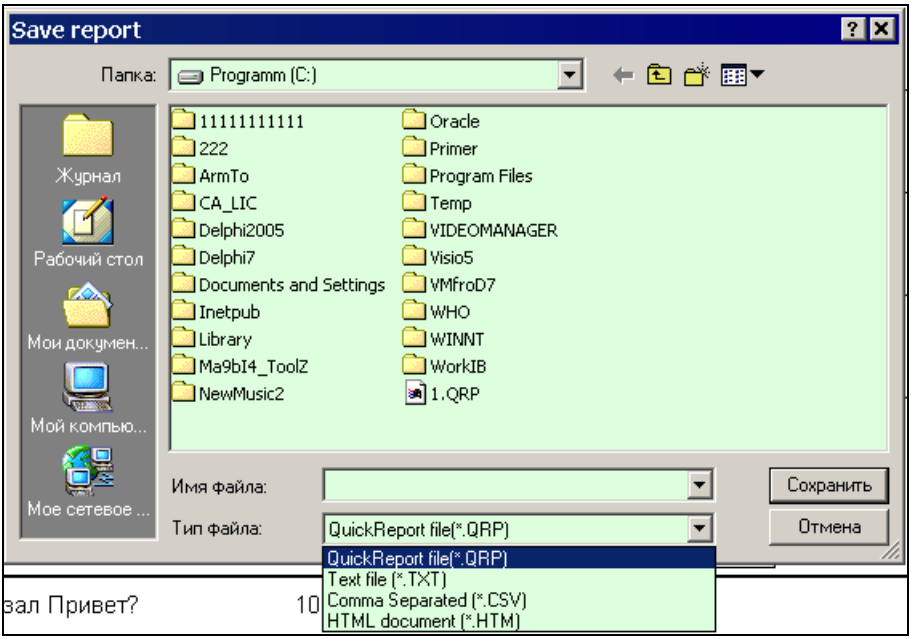


Рис. 6.20. Окно **Save report** при выборе формата сохранения отчете

6.8. Подстановочные (Lookup) поля

Продолжим усложнять отчет. Предположим, что нам надо вывести рядом с номером абонемента еще и фамилию читателя, которому этот абонемент принадлежит. Как быть в этом случае?

Как известно, фамилии читателей у нас хранятся в таблице Reader, принадлежность абонементов указывается в таблице Abonement, а информация о том, на какой абонемент бралась та или иная книга, хранится в таблице Move. Получается, что надо добавить не один бенд, чтобы дойти, наконец, до имени читателя, да к тому же не факт, что все это у нас заработает.

Мы пойдем более простым путем — воспользуемся подстановочными полями. Будем использовать их таким образом, чтобы в наборе данных `TRepMove` появился атрибут `FIOLookup`, который и будет содержать имя владельца атрибута.

Для этого располагаем на форме `FormSimpleRep` два компонента `IBTable`, связываем их с `DBLibrary`, один компонент называем `TRepAbonement` и связываем его с таблицей Abonement, другой называем `TRepReader` и связываем с таблицей Reader. Необходимо создать объекты-столбцы в редакторе столбцов для обоих наборов данных.

Теперь описывается важный момент, поэтому будьте внимательны. Производим двойной щелчок левой кнопкой мыши на компоненте `TRepAbonement`, в появившемся окне щелкаем правой кнопкой мыши и в выпадающем меню выбираем пункт **New Field** (рис. 6.21).

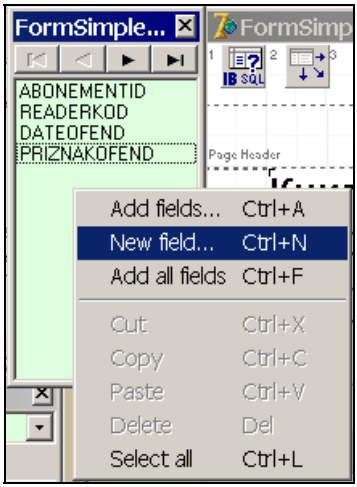


Рис. 6.21. Создание нового атрибута в наборе данных

Появится окно **New Field**. В поле **Name** указываем имя будущего атрибута — `FIOLookup`, причем надо учесть, что данный атрибут будет существовать только в наборе данных, в таблице его не будет.

В выпадающем списке **Type** выбираем тип атрибута — **String**, в поле **Size** — количество символов — **100**. В группе элементов **Field type** устанавливаем переключатель **Lookup**, тем самым указывая, что будет создаваться подстановочное поле. В группе элементов **Lookup definition** определяется, из какого набора данных берутся значения и через какие атрибуты организуется связь между наборами данных:

- **Key Fields** (ключевое поле) — `ReaderKod`.
- **Dataset** (набор данных, откуда будут браться значения) — так как нам нужна фамилия читателя, то соответственно нас интересует `TRepReader`.
- **Lookup Keys** (дополнительный ключ) — здесь указывается атрибут для набора данных, выбранного в выпадающем списке **Dataset**, через который будет осуществляться связь. Указываем здесь `ReaderID`.
- **Result Fiels** — атрибут, значения которого будут отображаться в результате. Указываем `FIO`.

В результате заполнения всех полей окно **New Field** должно принять вид, как на рис. 6.22.

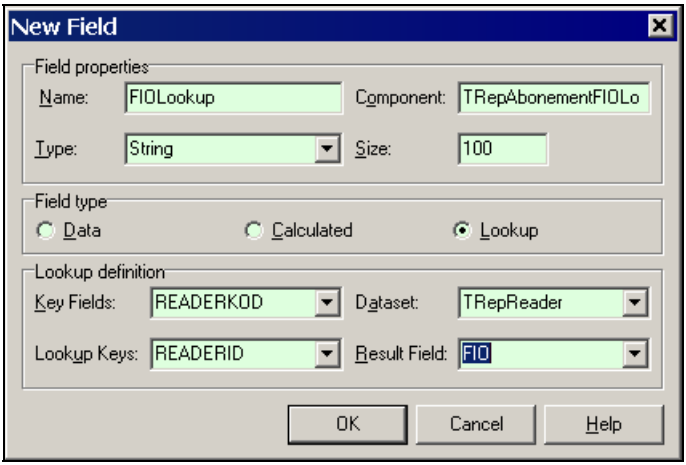


Рис. 6.22. Окно **New Field** с заполненными полями для набора `TRepAbonement`

Теперь нам надо создать точно таким же способом подстановочное поле для набора данных `TRepMove`. Окно **New Field** с заполненными полями представлен на рис. 6.23.

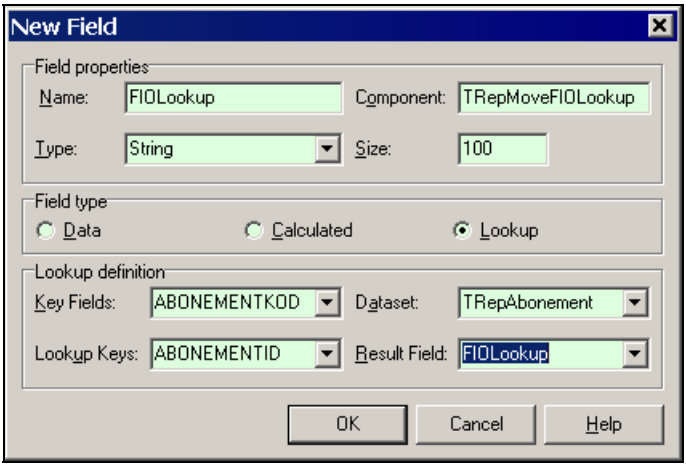


Рис. 6.23. Окно **New Field** с заполненными полями для набора `TRepMove`

Здесь мы организуем связь с набором данных TRepAbonement по коду абонемента, и в результате в подстановочном поле у нас отображается атрибут FIOLookup набора данных TRepAbonement.

Теперь на бенде Sub Detail размещаем QRLabel с текстом "ФИО читателя", а также QRDBText, который связываем с набором данных TRepMove и атрибутом FIOLookup, и настраиваем свойство Frame таким образом, чтобы текст был в рамке. Свойство AutoSize устанавливаем в False, свойство Aligment — в taCenter. Форма FormSimpleRep после размещения на ней дополнительных компонентов представлена на рис. 6.24.

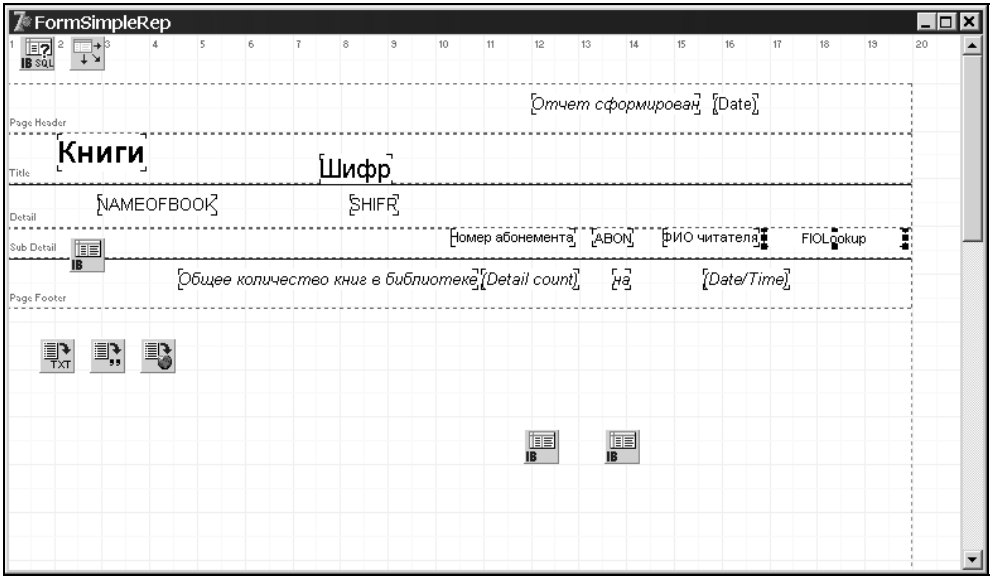


Рис. 6.24. Форма FormSimpleRep после размещения на ней дополнительных компонентов

Теперь немного изменим процедуру для пункта меню **Отчет\Простой отчет**:

```
procedure TMainForm.N6Click(Sender: TObject);
begin
 {Создаем форму, содержащую отчет}
 FormSimpleRep:=TFormSimpleRep.Create(self);

 {Открываем наборы данных}
 FormSimpleRep.QRepBook.Open;
 FormSimpleRep.TRepMove.Open;
```

```
 FormSimpleRep.TRepAbonement.Open;
 FormSimpleRep.TRepReader.Open;

 {Выводим отчет на экран}
 FormSimpleRep.QuickRep1.Preview;

 {После того, как пользователь закроет
 отчет, начнется выполняться этот участок кода}
 {Закрываем набор данных}
 FormSimpleRep.TRepReader.Close;
 FormSimpleRep.TRepAbonement.Close;
 FormSimpleRep.TRepMove.Close;
 FormSimpleRep.QRepBook.Close;
 {Закрываем форму}
 FormSimpleRep.Close;
end;
```

Можно запускать приложение и наслаждаться созданным отчетом.

На компакт-диске в каталоге ГЛАВА 6\SOURCE можно увидеть пример данной программы.

В каталоге ГЛАВА 6\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 6\VIDEO можно посмотреть видеоролики, посвященные теме, рассмотренной в данной главе.

Глава 7

RaveReports

7.1. Вступление

Начиная с 7 версии Delphi, вместе с ней поставляется система построения отчетов RaveReports, разработанная фирмой Nevrona, официальный сайт — **http://www.nevrona.com**.

Замечание

Вы можете воспользоваться готовыми примерами отчетов, для этого необходимо запустить систему RaveReports выбором пункта меню **Tools\Rave Designer**, в появившемся дизайнере отчетов выбрать **File\Open** и выбрать файл RaveDemo.res.

У меня на компьютере для Delphi 7 этот файл находится по следующему пути: C:\Program Files\Borland\Delphi7\Rave5\Demos\RaveDemo.res. Для Delphi 2005 этот файл находится по следующему пути: C:\Program Files\Borland\2005_1\ RaveReports\ Demos\Visual.

К недостаткам этой системы можно отнести запутанный интерфейс, что поначалу приводит к путанице.

7.2. Установка RaveReports

Система построения отчетов RaveReports устанавливается автоматически вместе с Delphi, начиная с 7 версии. Компоненты для формирования отчетов располагаются на вкладке **Rave** (рис. 7.1 и 7.2).

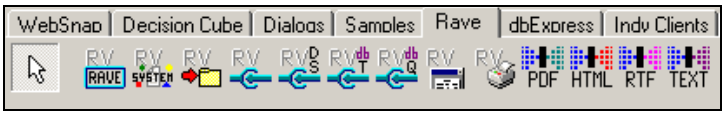


Рис. 7.1. Вкладка **Rave** в палитре компонентов Delphi 7

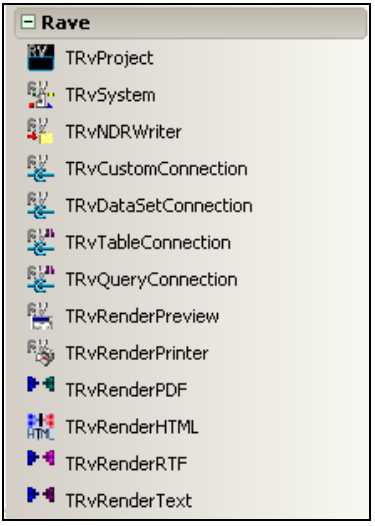


Рис. 7.2. Вкладка **Rave**
в палитре компонентов Delphi 2005

7.3. Использование RaveReports

Сразу рассмотрим пример. В этой главе будем работать с разработанным ранее приложением "Менеджер видеофильмов", рассмотренным в *главе 4*. Для начала сформируем простой отчет по всем фильмам, которые у нас есть в базе.

Замечание

Предлагаю вернуться к *главе 4*, в которой описывается разработка приложения под Microsoft SQL Server, и освежить воспоминания о структуре базы данных, используемой в приложении "Менеджер видеофильмов".

Открываем проект `VideoManager`. Создаем новую форму, называем ее `FormForReport`, сохраняем под именем `UForReport`. Удаляем форму из списка автоматически создаваемых форм, подключаем модуль `UDM`.

Помещаем на форму компонент `ADOQuery`, называем его `QRepMain`, подключаем его к `ADOConnection1`. В свойстве `SQL` пишем запрос выборки всех фильмов из таблицы Main:

```
SELECT NameOfFilm FROM MAIN
ORDER BY NameOfFilm
```

Запрос выбирает названия фильмов из таблицы Mian и упорядочивает их в алфавитном порядке.

Далее располагаем компонент `RVDataSetConnection`, в свойство `DataSet` устанавливаем `QRepMain`. Помещаем компоненты `RvProject`, `RvSystem`. У `RVProject` в свойство `Engine` устанавливаем `RvSystem1` (рис. 7.3).

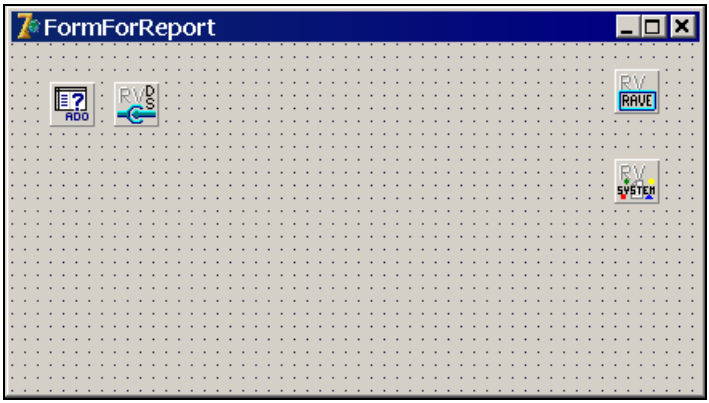


Рис. 7.3. Форма `FormForReport` с расположенными на ней компонентами для построения отчета

Щелкаем правой кнопкой мыши по компоненту `RvProject` и в выпадающем меню выбираем пункт **Rave Visual Designer**, тем самым мы запустим систему RaveReport. На экране появится заставка. После загрузки RaveReport можно будет увидеть следующее окно (рис. 7.4).

Замечание

Систему RaveReport можно вызвать также с помощью пункта главного меню **Tools\Rave Designer.**

Данное окно представляет собой визуальную среду RaveReports для создания отчетов, которая похожа на среду разработки Delphi. Вверху располагается панель инструментов, в которой представлены как кнопки быстрого доступа к функциям системы, так и компоненты, расположенные на нескольких вкладках для построения отчета. Слева можно увидеть инспектор свойств, который отображает свойства выбранного компонента. Справа находится дерево проекта отчета или инспектор отчета, который представлен на рис. 7.5.

Для разработки отчета доступны следующие вкладки:

- **Drawing** — графические элементы оформления отчета;
- **Bar Code** — различные варианты штрихкодов;
- **Standard** — содержит компоненты для размещения статического текста и изображения в отчете;

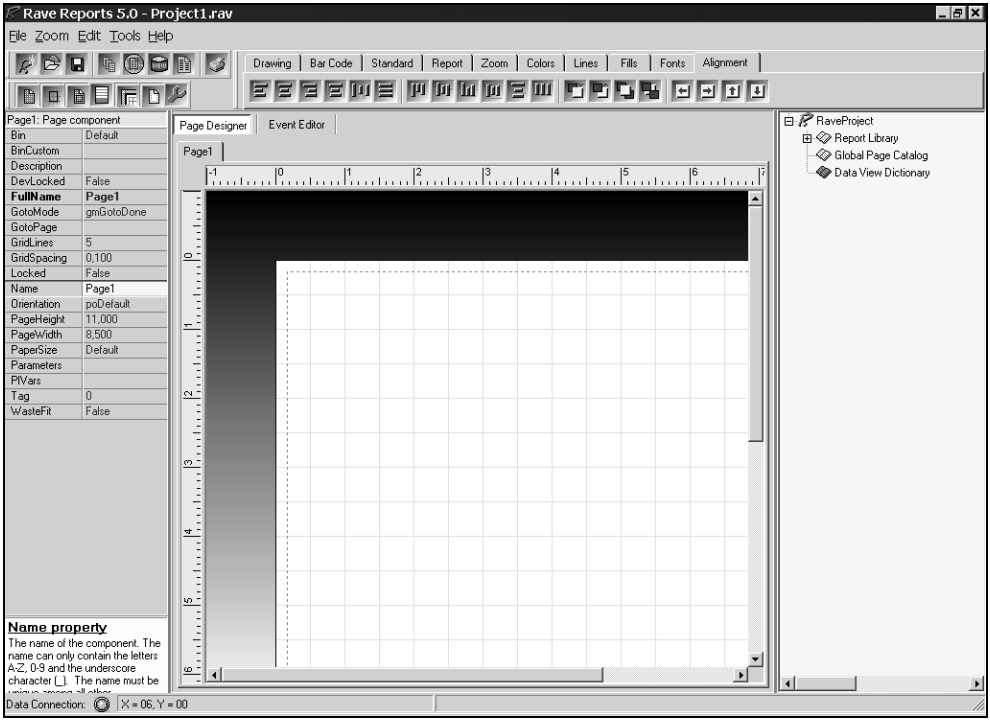


Рис. 7.4. Визуальная среда построения отчетов RaveReports

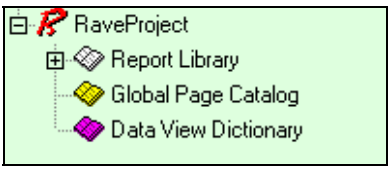


Рис. 7.5. Инспектор отчета RaveReports

- **Report** — содержит компоненты для размещения информации, полученной из внешних источников данных, таких как база данных;
- **Zoom** — управление масштабом текущей страницы отчета;
- **Colors** — задание цвета для выбранного элемента, расположенного на холсте отчета;
- **Line** — задание толщины линии и ее стиля для выбранного элемента, расположенного на холсте отчета;
- **Fills** — задает способ заполнения выбранного элемента, расположенного на холсте отчета;

- **Font** — задает параметры шрифта для текста;
- **Alignment** — задает выравнивание элементов относительно страницы отчета.

Вернемся к примеру. Теперь нам необходимо указать источник данных для формирования отчета (источников может быть несколько). Выбираем пункт меню **File\New Data Object** (данное действие можно сделать с помощью кнопки, расположенной в панели инструментов, ее пиктограмма совпадает с той, что можно увидеть при выборе пункта меню). На экране появится окно **Data Connections** (рис. 7.6).

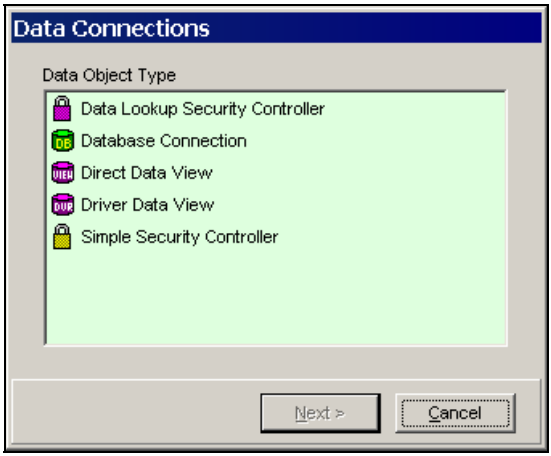


Рис. 7.6. Окно **Data Connections**

Выбираем **Direct Data View** — это будет означать, что мы хотим использовать данные, поставляемые компонентом `RVDataSetConnection`. Нажимаем **Next**, на экране появится окно, как показано на рис. 7.7.

В нем отображаются доступные источники данных, щелкаем на единственном доступном нам и нажимаем кнопку **Finish**. Теперь, если раскрыть пункт **Data View Dictionary** в инспекторе отчета, то можно увидеть пункт **DataView**, который отвечает за один источник данных. Если раскрыть его, то мы увидим все атрибуты, которые возвращает запрос `QRepMain`, тип атрибута символизирует пиктограмма, расположенная слева от его имени (рис. 7.8).

У всех элементов, расположенных в инспекторе отчета, есть свойства, которые можно менять в инспекторе свойств. Например, поменяем свойство `Name` для `DataView1` на `MyDataView`. Теперь выбираем пункт меню **Tools\Report Wizards\Simple Table**, данным действием мы вызовем мастер построения простых отчетов. На экране появится окно, в котором необходимо выбрать доступный **Data View**, выбираем `MyDataView` (рис. 7.9).

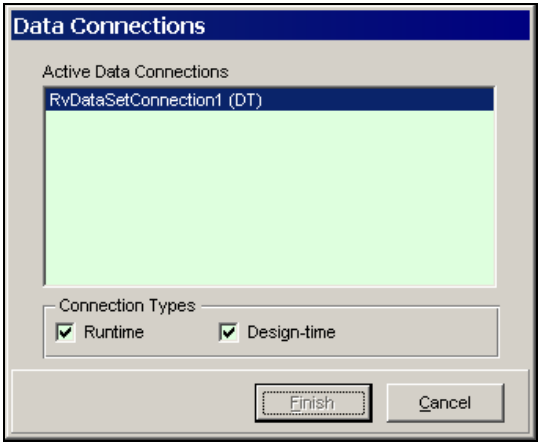


Рис. 7.7. Окно выбора **Data Connections**

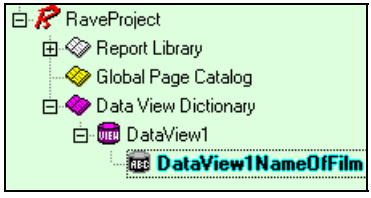


Рис. 7.8. Инспектор отчета с подключенным источником данных

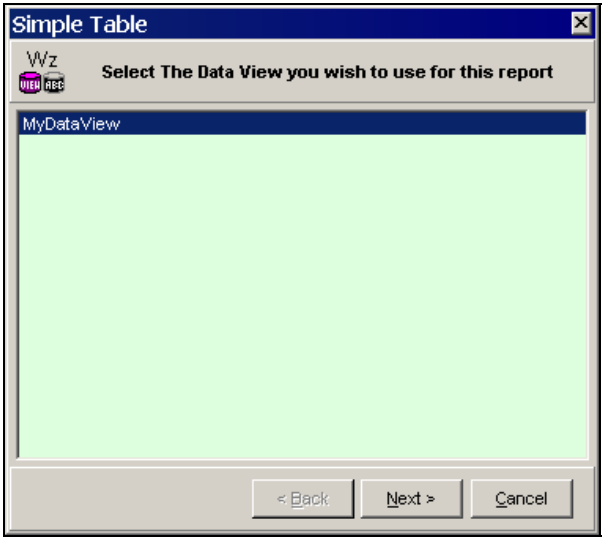


Рис. 7.9. Выбор источника данных в окне мастера простых отчетов

Нажимаем **Next**. Появится окно, в котором необходимо выбрать атрибуты, которые будут отображаться в отчете. Так как у нас один атрибут (`NameOfFilm`), то устанавливаем флаг напротив него (рис. 7.10).

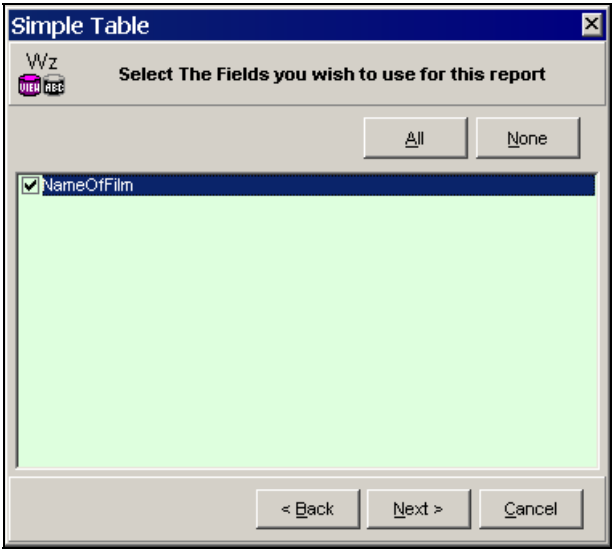


Рис. 7.10. Выбор атрибутов, отображаемых в отчете

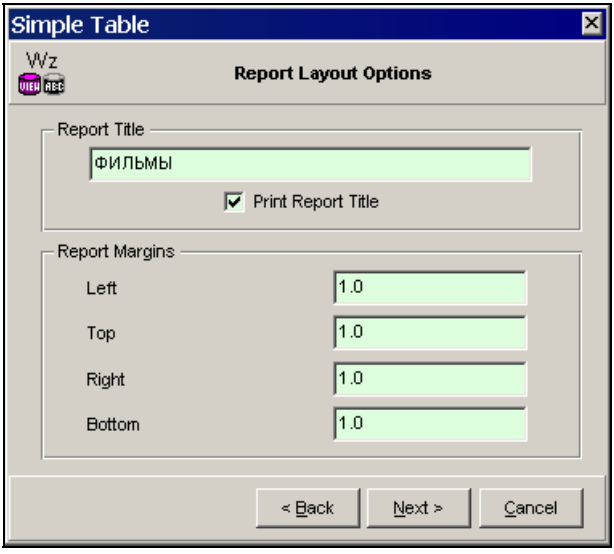


Рис. 7.11. Установка границ отступа

Нажимаем **Next**. Если бы у нас было несколько атрибутов, то появилось бы окно для выбора порядка следования атрибутов в отчете. Так как атрибут у нас один, то мы не увидим это окно. Вместо этого у нас сразу появится окно, где необходимо задать заголовок отчета (у меня — "ФИЛЬМЫ"), также можно указать границы отступа — групп элементов **Report Margin**, советую оставить значения, предложенные по умолчанию (рис. 7.11).

Нажимаем **Next**. Появляется последнее окно мастера, где задаются параметры шрифта для различных частей отчета, например, в секции **Title Font** задаются параметры шрифта для заголовка отчета (рис. 7.12).

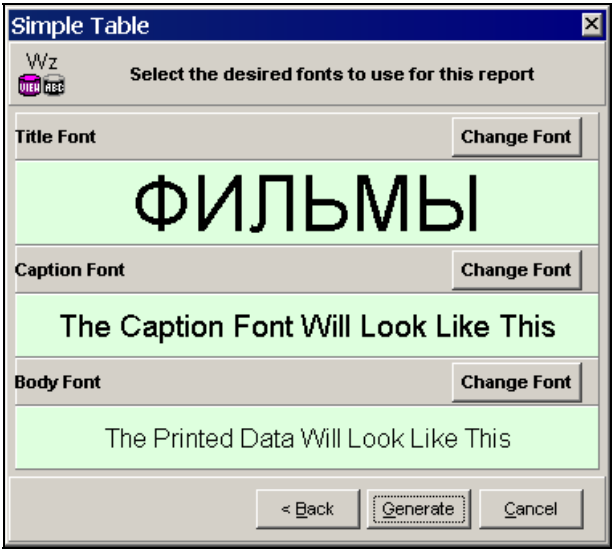


Рис. 7.12. Установка параметров шрифта

Нажимаем **Generate**. После этого у нас появится готовый отчет (рис. 7.13).

Обратите внимание, что в инспекторе отчета появились дополнительные элементы.

Выбираем пункт меню **File\Save**, на экране появится окно, в котором необходимо указать путь и имя файла для будущего отчета. Предлагаю назвать его `FirstReport` и сохранить в папке проекта VideoManager.

Замечание

Файлы RaveReports имеют расширение rav.

Для того чтобы просмотреть отчет, необходимо выбрать пункт меню **File\Execute Report**, после чего на экране появится окно **Output Options** (рис. 7.14).

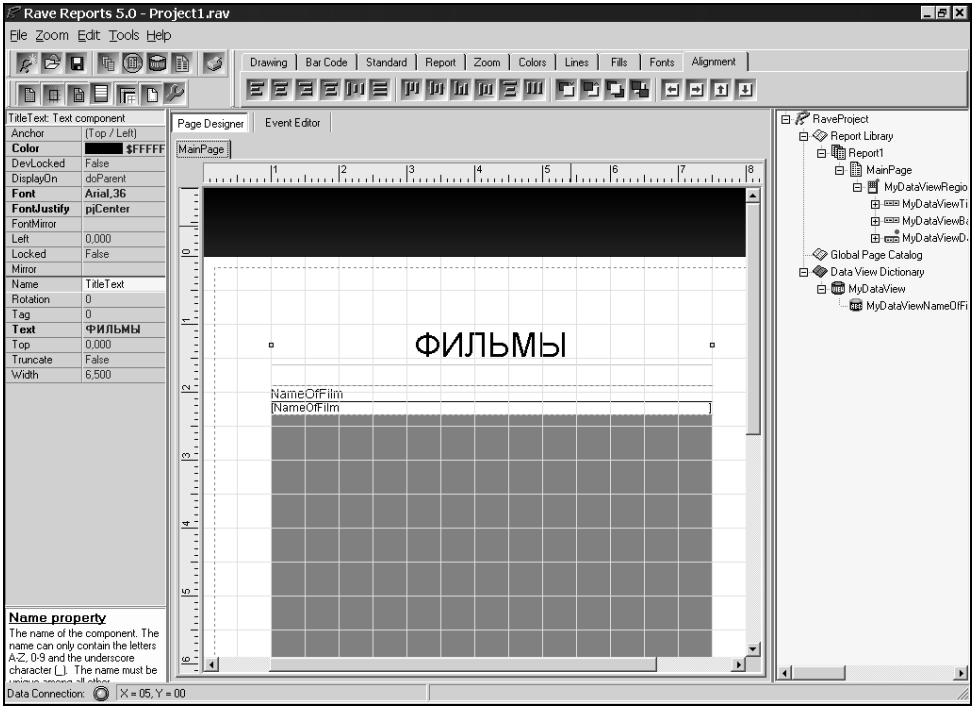


Рис. 7.13. Отчет, созданный с помощью мастера

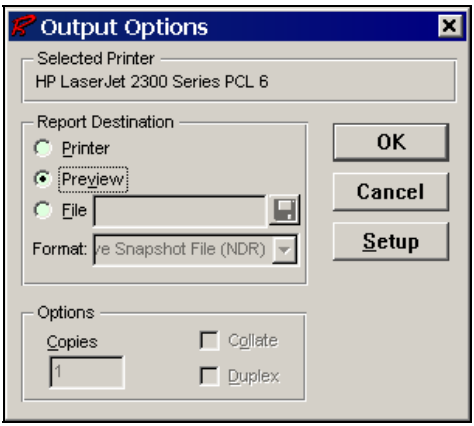


Рис. 7.14. Окно **Output Options**

Здесь можно выбрать, куда будет выводиться отчет (на принтер, на экран — по умолчанию, или в файл). Нажимаем **OK**, после этого можно будет увидеть готовый отчет (рис. 7.15).

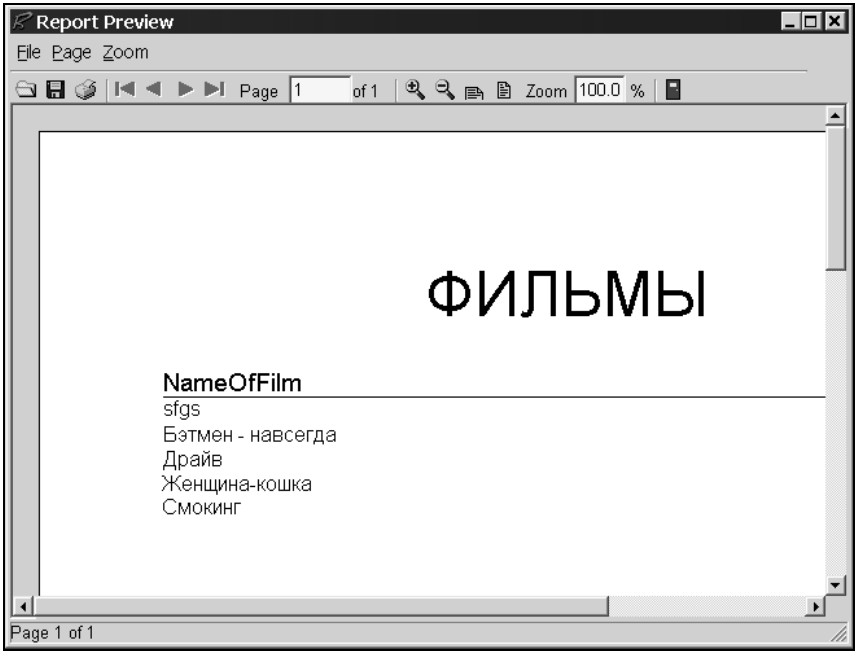


Рис. 7.15. Просмотр готового отчета

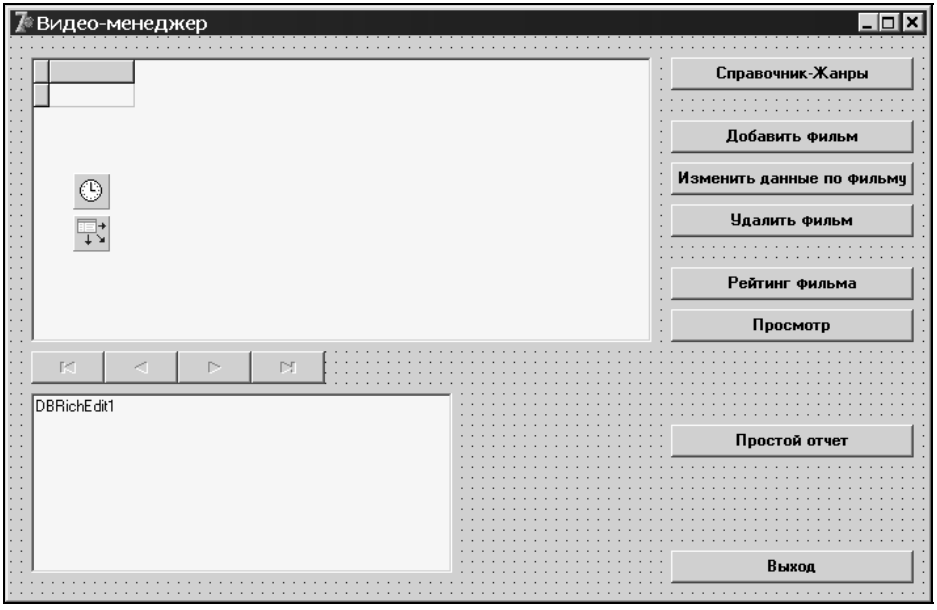


Рис. 7.16. Форма `MainForm` с кнопкой **Простой отчет**

Закрываем окно просмотра отчета, закрываем RaveReports. Поработаем теперь в Delphi. Для компонента `RVProject` в свойстве `ProjectFile` указываем имя файла отчета — FirstReport.rav.

Замечание

Путь может быть как относительным, так и абсолютным. Если путь относительный, то отчет должен располагаться в той же папке, что и проект программы.

Переходим на форму `MainForm`, добавляем кнопку `Button` с текстом "Простой отчет" (рис. 7.16).

Для данной кнопки пишем следующий код вывода отчета на экран (необходимо будет подключить модуль `UForReport`):

```
FormForReport:=TFormForReport.Create(self);

{Формируем отчет}
FormForReport.RvProject1.Execute;

FormForReport.Close;
```

Замечание

Для того чтобы сформировать отчет в системе RaveReports, не обязательно открывать перед этим набор данных, который используется в этом отчете.

7.4. Принцип построения отчета

Принцип построения отчета в RaveReports точно такой же, как и в QReport, за исключением того, что в RaveReports есть собственный дизайнер, в котором и выполняются все действия по построению отчета. Точно так же необходимо задать бенды, на которых отображаются дополнительные элементы для отображения информации. Главное отличие от QReport в том, что один отчет может состоять из нескольких страниц, в каждой из которых формируется самостоятельный отчет; этот момент мы рассмотрим на примере чуть позже.

Структуру отчета всегда можно посмотреть в инспекторе отчета, что является очень удобным. Так как всегда можно увидеть, из каких компонентов состоит отчет и какие источники данных используются.

Основными компонентами для работы с отчетом в Delphi являются `RvSystem` и `RvProject`.

- `RvSystem` — содержит ядро системы RaveReports, включает в себя возможность предварительного просмотра, а также вывода отчета на печать. Компонент является обязательным для работы с отчетом.

- `RvProject` — через данный компонент производится соединение с файлом отчета, который должен быть предварительно создан в дизайнере отчетов RaveReports. В свойстве `Engine` данного компонента необходимо сослаться на компонент `RvSystem`.

Для соединения с наборами данных используется компонент `RvDataSetConnection`, в свойстве `DataSet` которого указывается необходимый набор данных. Для того чтобы вывести отчет на экран, необходимо вызвать метод `Execute` компонента `RvProject`. В общем случае код выглядит следующим образом:

```
RvProject.Execute
```

7.5. Усложняем пример

Сейчас попробуем создать точно такой же отчет, только не прибегая к помощи мастера. Располагаем на форме `FormForReport` еще один компонент `RvProject` (оставим ему имя `RvProject2`, данное по умолчанию). В свойство `Engine` устанавливаем `RvSystem1`. Вызываем дизайнер RaveReports.

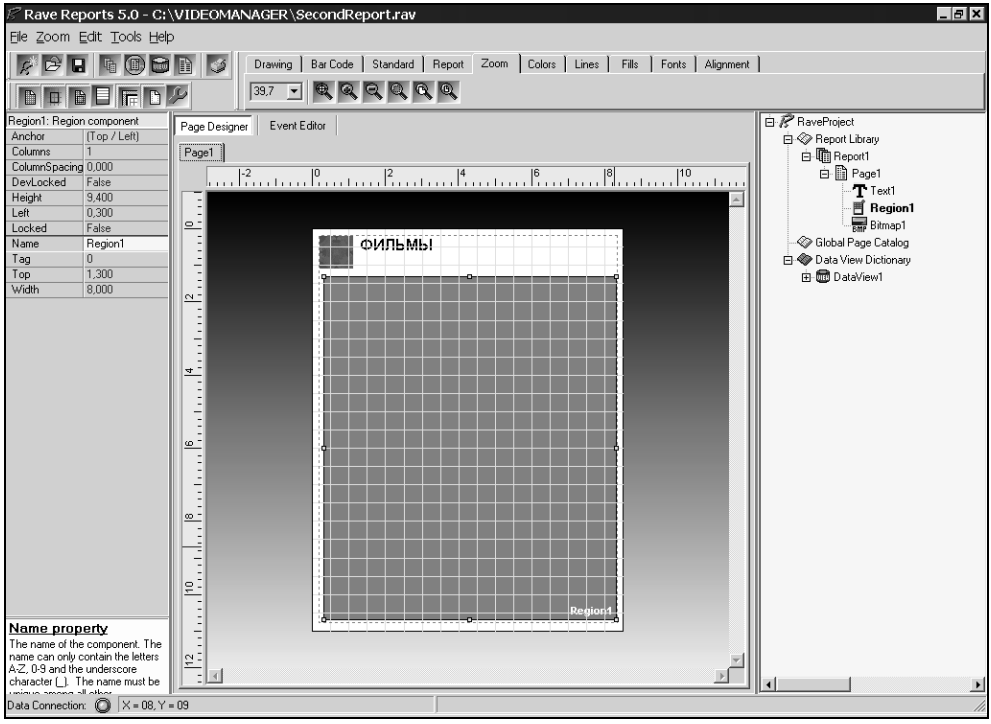


Рис. 7.17. Размещение компонента `Text component`

Создаем новый `DataView` (так же как описано выше), в качестве источника указываем `RvDataSetConnection1`. Располагаем в верхней левой части холста отчета `Bitmap component` (вкладка **Standart**), в свойстве `Image` выбираем любую картинку, которая нравится (я выбрал ПАРКЕТ.BMP из папки Windows). Справа от картинки размещаем компонент `Text component` (вкладка **Standard**), в свойство `Text` пишем "Фильмы", настраиваем свойство `Font`, чтобы шрифт был 30 кегля и полужирный. Под компонентом `Text component` размещаем компонент `Region` (вкладка **Report**), растягиваем его на всю оставшуюся свободную часть отчета (рис. 7.17).

Замечание

Так как весь отчет не помещается на экране, то можно воспользоваться вкладкой **Zoom** и кнопкой **Zoom Out** (изображение лупы с минусом).

Теперь все остальные компоненты необходимо будет размещать непосредственно на компоненте `Region`. Размещаем `Band component` (вкладка **Report**), на него — `Text component` с надписью "Название". Под ним `HLine component` (вкладка **Drawing**), растягиваем его по всей ширине отчета (рис. 7.18).

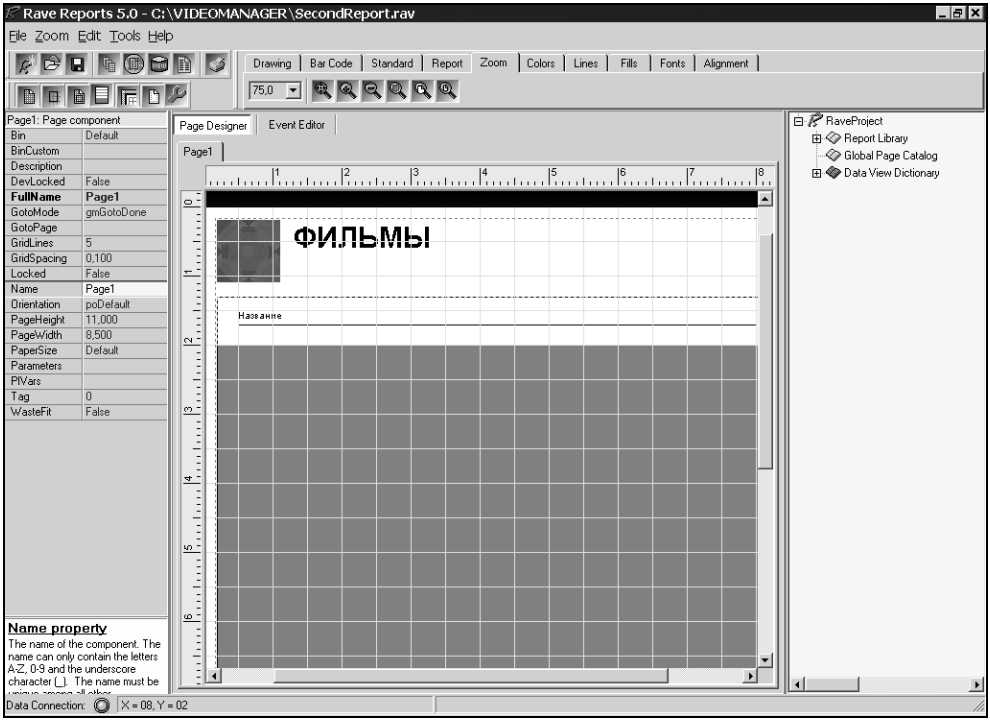


Рис. 7.18. Размещение компонента `HLine component`

Идем дальше. Располагаем компонент DataBand component (вкладка **Report**), в его свойство DataView устанавливаем DataView1. Располагаем на нем DataText component, свойство DataView которого устанавливаем в DataView1, а свойство DataField — в NameOfFilm, таким образом, мы связались с конкретным атрибутом. В свойство Width необходимо установить 2 (рис. 7.19).

Выбираем пункт меню **File\Save**, на экране появится окно, в котором необходимо указать путь и имя файла для будущего отчета. Предлагаю назвать его SecondReport и сохранить в папке проекта VideoManager. Закрываем дизайнер отчетов. Для компонента RVProject2 в свойстве ProjectFile указываем имя файла отчета — SecondReport.rav.

Переходим на форму MainForm, добавляем кнопку Button с текстом "Простой отчет№2" и пишем для нее код вывода отчета на экран:

```
FormForReport:=TFormForReport.Create(self);

{Формируем отчет}
FormForReport.RvProject2.Execute;

FormForReport.Close;
```

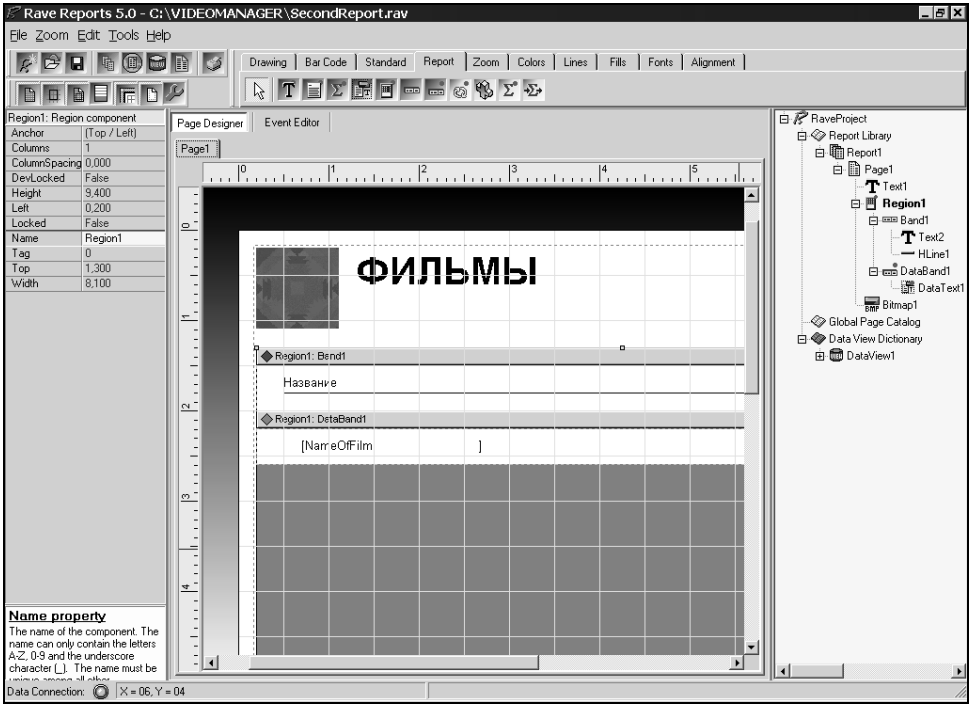


Рис. 7.19. Размещение компонента DataBand component

Если сейчас запустить приложение и нажать кнопку **Простой отчет№2**, то можно увидеть только что созданный отчет.

Усовершенствуем отчет, попробуем сделать так, чтобы напротив каждого фильма отображался его штрихкод, который будет формироваться исходя из кода фильма. Для начала изменим запрос в свойстве SQL компонента QRepMain для выборки всех фильмов из таблицы Main:

```
SELECT * FROM MAIN
ORDER BY NameOfFilm
```

После этого производим двойной щелчок левой кнопкой мыши на компоненте RvProject2, загрузится дизайнер вместе с отчетом. Размещаем рядом с компонентом DataText component компонент PostNetBarCode (вкладка **Bar Code**). В свойство DataView устанавливаем DataView1, а в свойство DataField — FilmID, таким образом мы указываем, из какого атрибута будет браться значение для формирования штрихкода.

Замечание

Все штрихкоды реализуют различные стандарты, некоторые могут работать только с числовыми последовательностями, а некоторые с буквенно-цифровыми. Например, штрихкод PostNet (компонент PostNetBarCode) используется почтовой службой США.

Теперь можно сформировать отчет и увидеть полученный результат (рис. 7.20).

Теперь сделаем, чтобы под названием фильма выводились оценки, поставленные пользователями. Для этого необходимо добавить еще один набор данных. Помещаем на форму FormSimpleRep компонент ADOTable, называем его TRepReiting, связываем с ADOConnection1, в свойстве Table выбираем Reiting — таблица, с которой будет работать данный компонент. Рядом помещаем компонент RvDataSetConnection, в свойство DataSet которого устанавливаем TRepReiting.

Производим двойной щелчок левой кнопкой мыши на компоненте RvProject2. После загрузки дизайнера отчетов RaveReports выбираем пункт меню **File\New Data Object** и подключаем новый источник данных, в инспекторе отчета он будет называться DataView2.

Размещаем в отчете очередной DataBand component. В его свойство DataView устанавливаем DataView2. Теперь необходимо настроить этот бенд, чтобы он отображал оценки для каждого фильма:

- в свойство MasterDataView устанавливаем DataView1 (главный DataView);
- в свойство MasterKey устанавливаем FilmID (атрибут, по которому будет организована связь);
- в свойство DetalKey устанавливаем FilmKod (атрибут, по которому организуется связь в зависимом DataView).

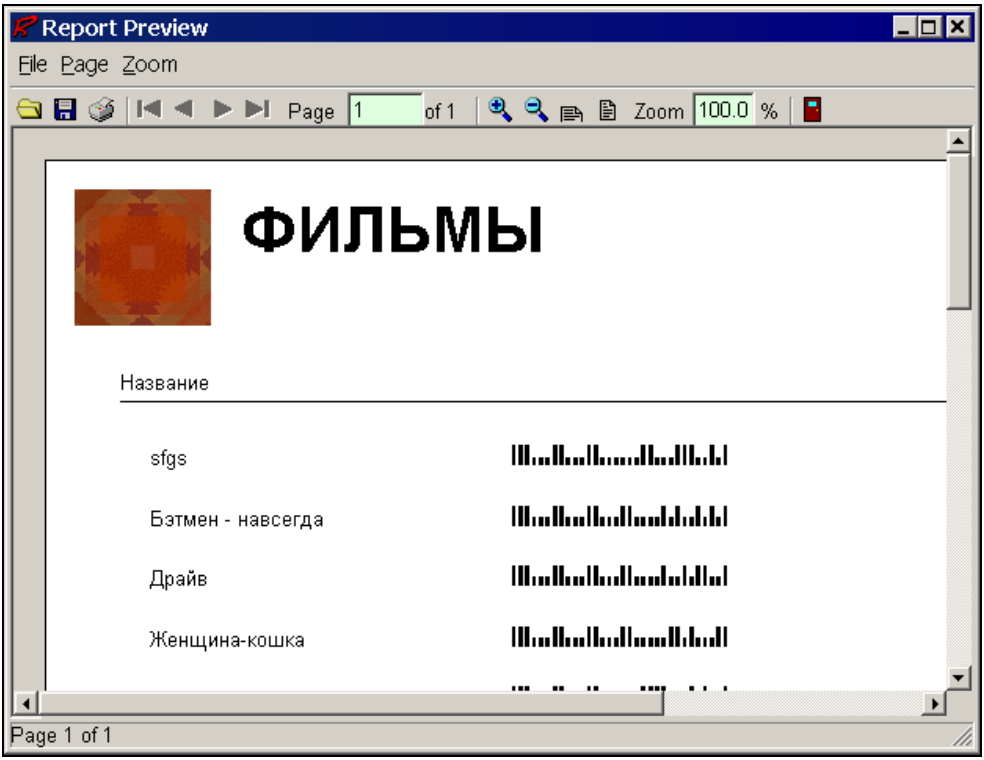


Рис. 7.20. Отчет с использованием штрихкода в режиме просмотра

В свойстве `ControllerBand` указываем `DataBand1`, таким образом мы устанавливаем между бендами связь master-detail. Теперь открываем свойство `BandStyle`, появится окно **Band Style Editor** (рис. 7.21).

Замечание

В данном окне бенд `DataBand1` отражен по три раза. Это не ошибка, это взгляд разработчиков системы RaveReports; таким образом они хотели показать, что бенд в отчете формируется столько раз, сколько записей в таблице, которые он отображает.

Здесь необходимо установить переключатель **Detail**, это очень важно, иначе отчет будет работать неправильно. Вид окна изменится (рис. 7.22).

Нажимаем **OK**.

Остался последний штрих: размещаем `DataText component` на `DataBand2`, в свойство `DataView` которого устанавливаем `DataView2`, а в свойство `DataField` — `Ball`. Можно запускать программу и наслаждаться отчетом.

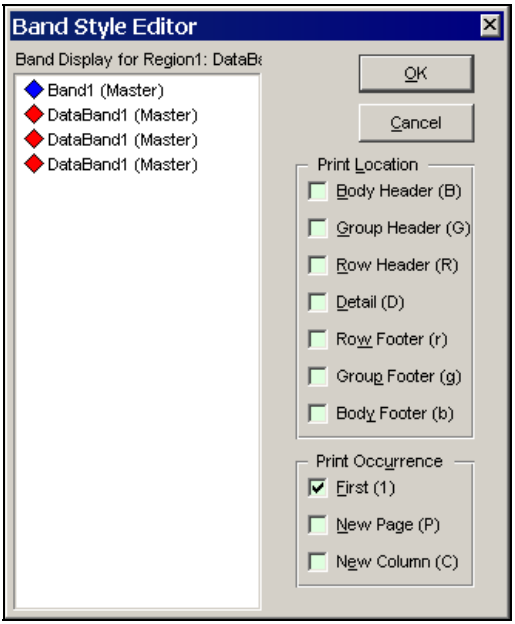


Рис. 7.21. Окно **Band Style Editor**

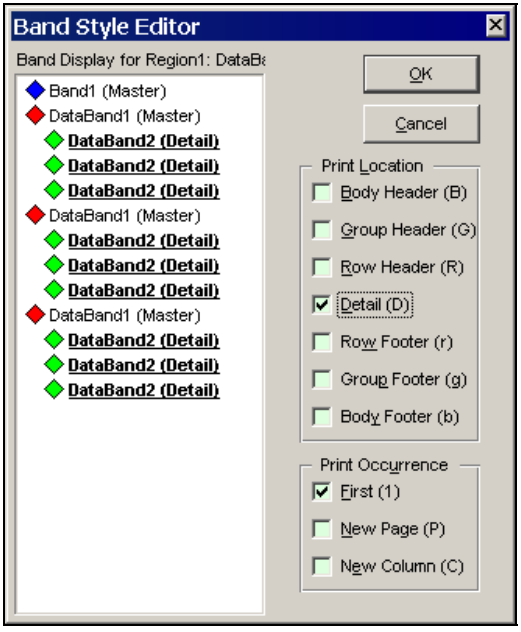


Рис. 7.22. Окно **Band Style Editor** с установленным переключателем **Detail**

7.6. Подстановочные (Lookup) поля

Замечание

В этом разделе будет дано краткое описание создания подстановочных полей, если потребуется больше информации, то можно вернуться к *главе 6*, *разд. 6.6*.

Усовершенствуем наш отчет таким образом, чтобы рядом с оценкой по фильму отображалось имя пользователя, который поставил эту оценку. На форму `FormForReport` помещаем компонент `ADOTable`, переименовываем его в `TRepSecurityTable`, связываем с `ADOConnection1`, в свойстве `Table` выбираем `SecurityTable`. Необходимо создать объекты-столбцы в редакторе столбцов для всех наборов данных, расположенных на форме `FormForReport`. Создаем новое поле в наборе данных `TRepReiting` в соответствии с рис. 7.23.

Загружаем дизайнер отчетов с проектом `SecondReport`. В инспекторе отчета щелкаем правой кнопкой мыши на пункте `DataView2` и в выпадающем меню выбираем пункт **Refresh** (рис. 7.24).

Данное действие необходимо, чтобы RaveReports "узнал", что у нас есть подстановочное поле, так как в тот момент, когда мы создавали **DataView**, его еще не было. А информация в RaveReports автоматически не обновляется.

Рядом с компонентом `DataText`, который отображает оценки, располагаем еще один `DataText component`, настраиваем его:

- в свойство `DataView` устанавливаем `DataView2`;
- в свойство `DataField` — `LoginLookup`.

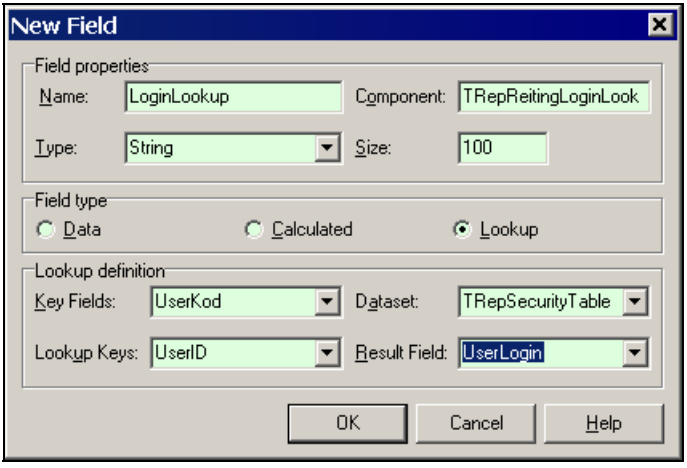


Рис. 7.23. Создание Lookup-поля

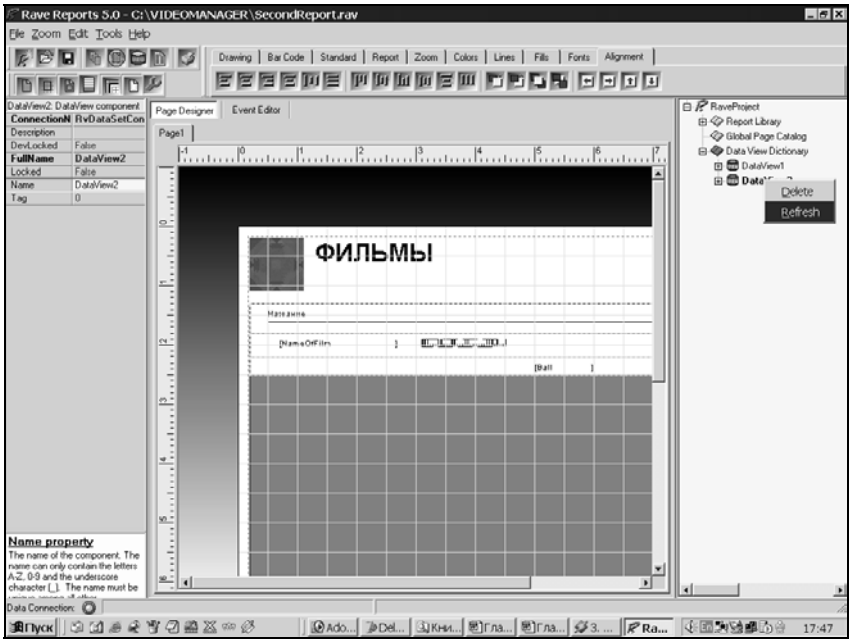


Рис. 7.24. Обновление информации в инспекторе отчета

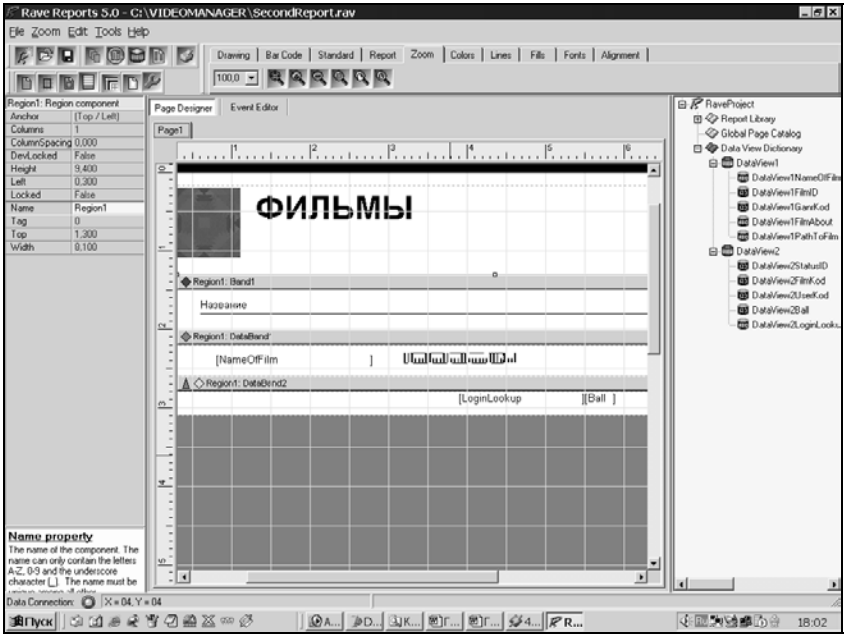


Рис. 7.25. Вид инспектора отчета после обновления

7.7. Многостраничный отчет

Предположим, нам надо в одном отчете выводить кроме списка фильмов и оценок по ним еще и список всех жанров, которые присутствуют в базе. Загружаем дизайнер отчетов с проектом `SecondReport`. Выбираем пункт меню **File\New Report Page**, тем самым мы создадим еще одну страницу в отчете (рис. 7.26).

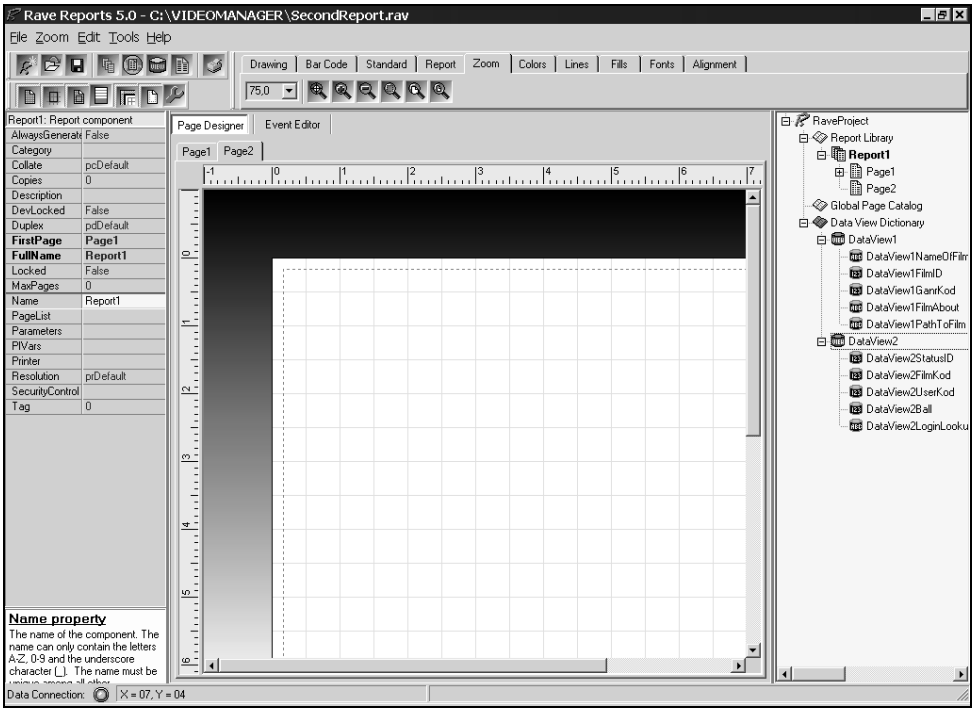


Рис. 7.26. Новая страница отчета

Выбираем пункт меню **File\New Data Object**, в появившемся окне выбираем пункт **Database Connection**. Тем самым мы указываем, что будем подключаться к базе данных непосредственно из отчета и никаких дополнительных компонентов в приложении нам не потребуется (рис. 7.27).

Нажимаем **Next**, появляется окно, в котором предлагается выбрать технологию доступа к данным (рис. 7.28).

Выбираем **ADO** и нажимаем **Finish**. Появится стандартное окно для настройки параметров соединения (рис. 7.29).

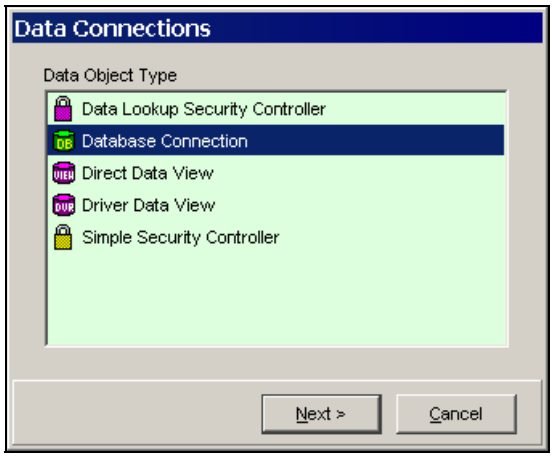


Рис. 7.27. Установка подключения к базе непосредственно из отчета

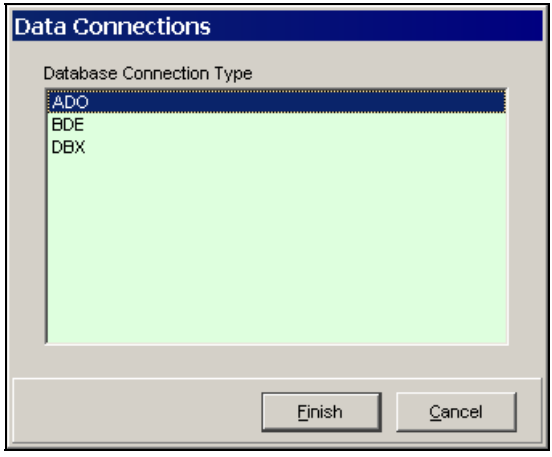


Рис. 7.28. Выбор технологии доступа к данным

Необходимо настроить соединение так же, как это было сделано в *главе 4*, посвященной работе с Microsoft SQL Server. После этого нажать кнопку **OK**. В инспекторе отчета появится новый объект — `Database` (рис. 7.30).

Идем далее. Выбираем пункт меню **File\New Data Object**, в появившемся окне выбираем пункт **Driver Data View**, тем самым мы указываем, что будет получать данные без помощи дополнительных компонентов (рис. 7.31).

Нажимаем **Next**. Появится окно, где необходимо выбрать независимое соединение с базой данных (рис. 7.32).

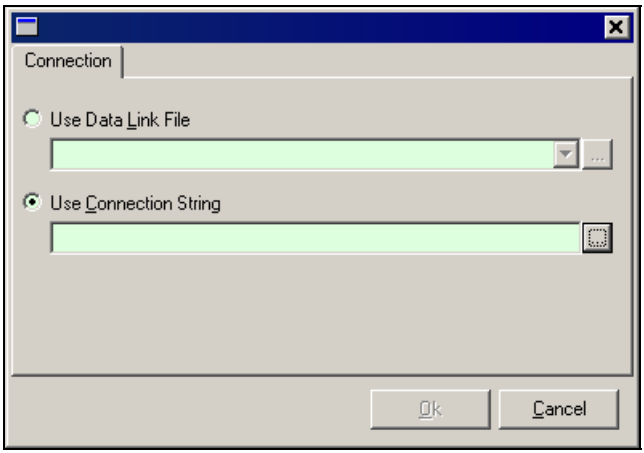


Рис. 7.29. Окно для настройки параметров соединения

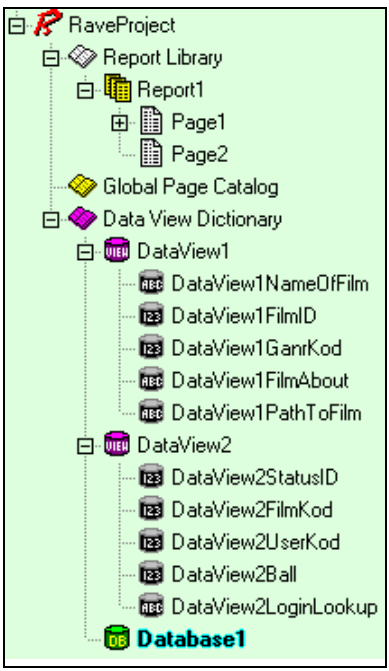


Рис. 7.30. Инспектор отчета и доступные источники данных

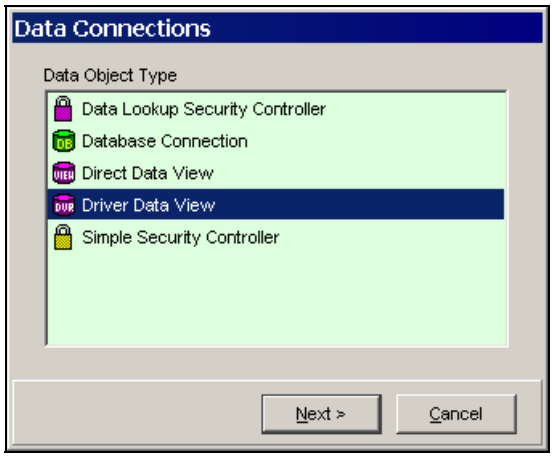


Рис. 7.31. Установка подключения к базе без помощи дополнительных компонентов

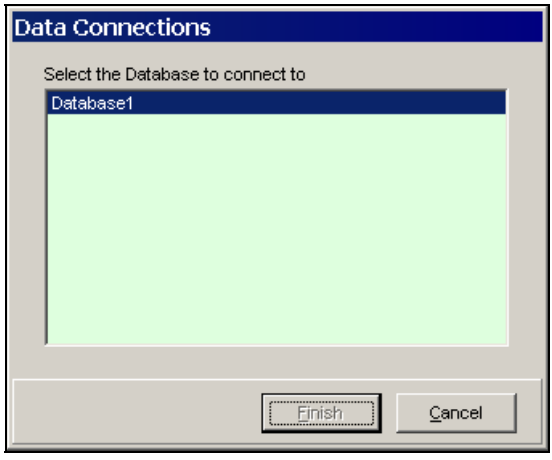


Рис. 7.32. Выбор независимого соединения с базой данных

Выбираем `Database1` и нажимаем **Finish**. Появится окно **Query Advanced Designer**, состоящее из двух вкладок:

- **Layout** — здесь выбираются таблицы, которые должны присутствовать в отчете. Делается это с помощью перетаскивания имен таблиц из левой половины окна и установкой переключателя тем атрибутам, которые должны быть доступны. Если переключатель установлен атрибуту с именем *, то значит, все атрибуты таблицы будут доступны. Нам необходимо добавить в отчет таблицу Ganri (рис. 7.33);

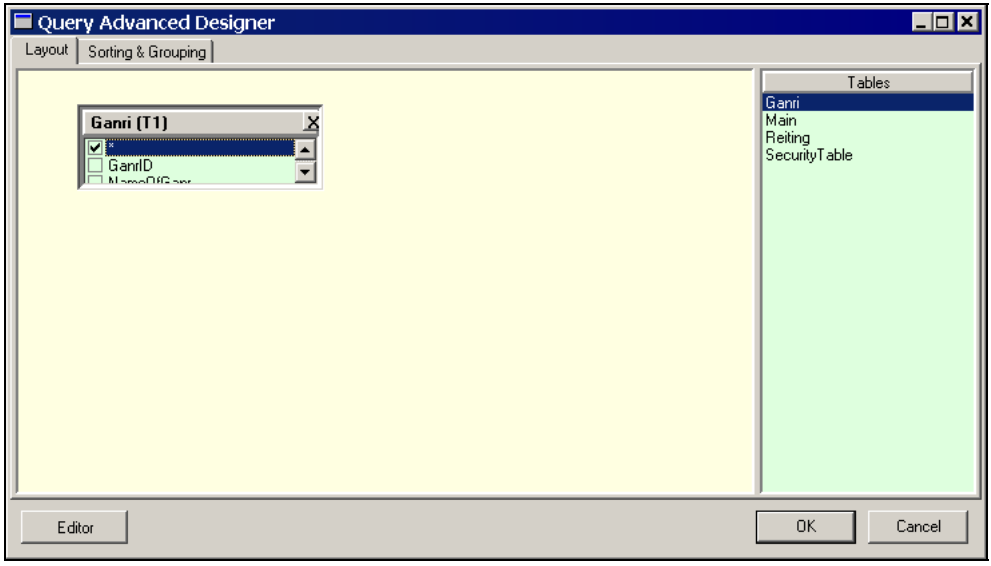


Рис. 7.33. Окно **Query Advanced Designer**

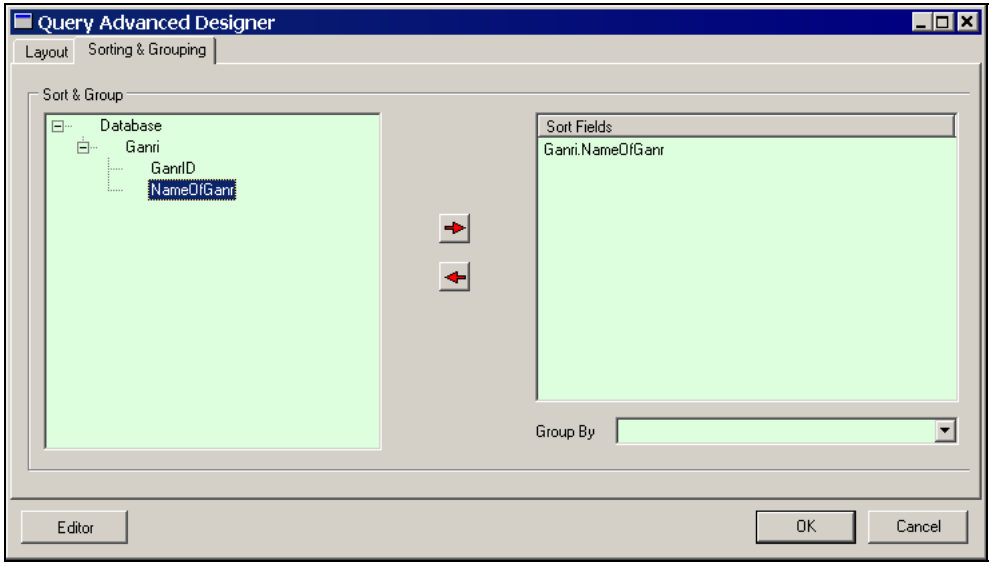


Рис. 7.34. Окно **Query Advanced Designer** с выбранной вкладкой **Sorting & Grouping**

- **Sorting & Grouping** — при активизации этой вкладки окно делится на две части, в левой выбирается атрибут, который с помощью стрелок перено-

сится в правую часть окна, таким образом, по нему организуется сортировка (рис. 7.34).

Нажимаем кнопку **OK**. Теперь у нас есть полноценный источник данных под названием `DriverDataView1`. Можно использовать его для построения отчетов.

Теперь располагаем компонент `Text component` (вкладка **Standard**), в свойство `Text` пишем "Жанры". Настраиваем свойство `Font`, чтобы шрифт был 30 кегля и полужирный. Под `Text component` размещаем компонент `Region` (вкладка **Report**), растягиваем его на всю оставшуюся свободную часть отчета. Далее все остальные компоненты размещаем непосредственно на компоненте `Region`.

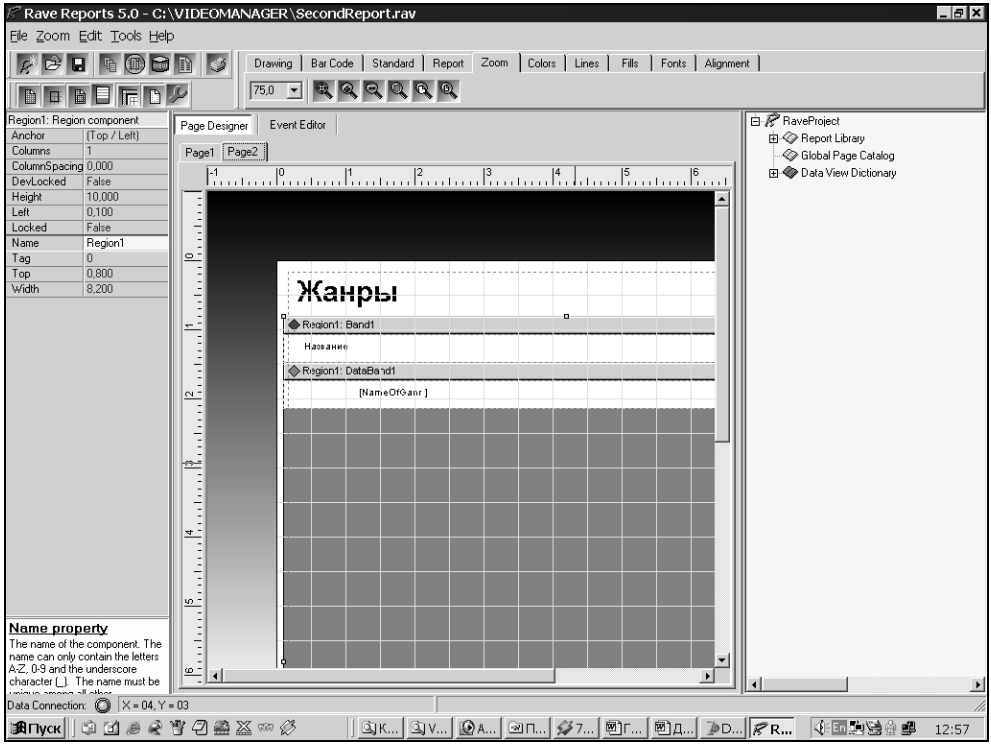


Рис. 7.35. Размещение компонента `DataText component`

Располагаем `Band component` (вкладка **Report**), на него `Text component` с надписью "Название". Далее располагаем `DataBand component`, в его свойстве `DataView` устанавливаем `DriverDataView1`, таким образом связывая бенд с источником данных. Располагаем на нем `DataText component`, в

свойство DataView которого также устанавливаем DriverDataView1, а свойство DataField — в NameOfGanr, таким образом мы связались с конкретным атрибутом (рис. 7.35).

Теперь в инспекторе отчета раскрываем объект ReportLibrary, делаем активным пункт Report1 (рис. 7.36).

Далее в инспекторе свойств открываем свойство PageList, появляется окно **Page List Editor**, где задаются страницы, которые будут отображаться в отчете. В выпадающем списке **Report Pages** выбираем Page1 и нажимаем **Add**, затем выбираем Page2 и также нажимаем **Add** (рис. 7.37).

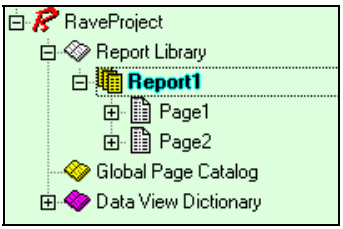


Рис. 7.36. Объект ReportLibrary в инспекторе отчета с активным пунктом Report1

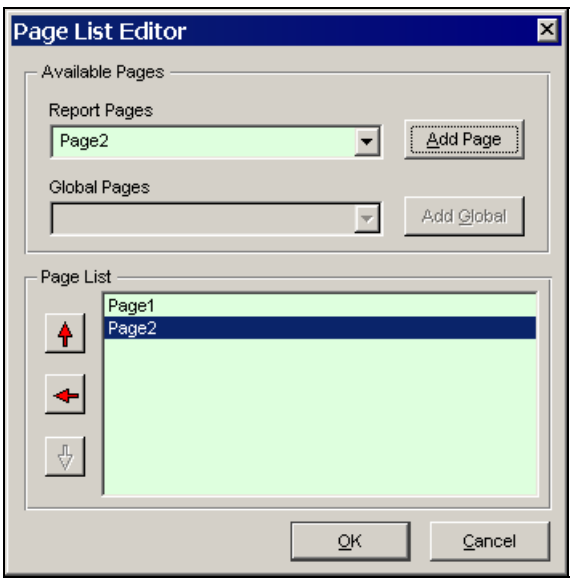


Рис. 7.37. Добавление страниц в окне **Page List Editor**

Нажимаем **OK**, теперь можно запускать отчет и наслаждаться результатом.

Замечание

Данный отчет не удастся запустить на просмотр методом Execute, так как он содержит независимый источник данных, встроенный непосредственно в файл отчета. Такие отчеты предназначены только для запуска в визуальной среде разработки RaveReports.

7.8. Фильтры

Если во время выполнения программы в сформированном отчете в панели инструментов нажать кнопку **Сохранить** с пиктограммой дискеты, то появится диалоговой окно **Save report**. Если попытаться выбрать формат для сохранения в выпадающем списке **Тип файла**, то можно увидеть, что сохранить отчет можно только в формате RaveReports — NDR (рис. 7.38).

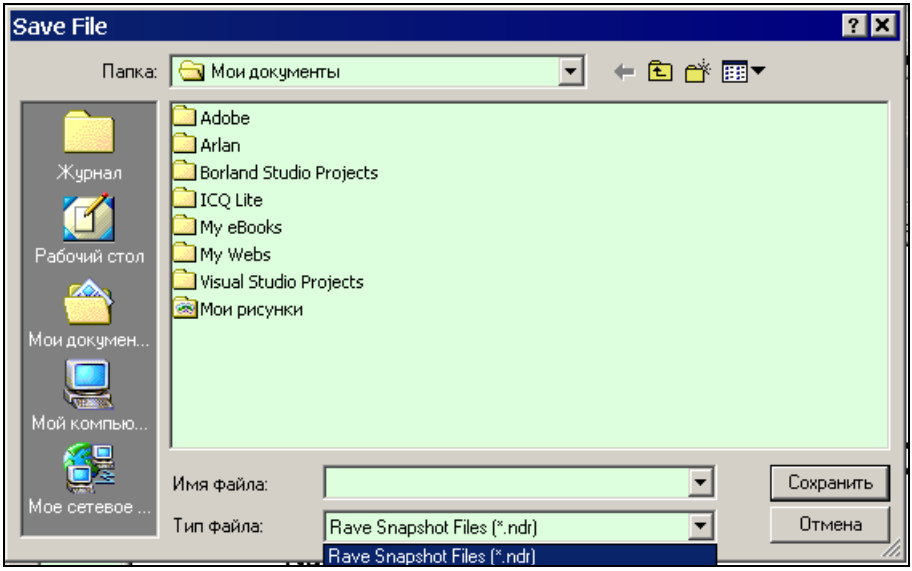


Рис. 7.38. Окно **Save File**

Замечание

Если вам захочется просмотреть сохраненный отчет, то можно нажать кнопку **Open** с пиктограммой открывающейся папки в панели инструментов в сформированном отчете.

Но часто бывают случаи, когда отчет надо сохранить в другом формате, который можно было бы использовать в других программах, например, Micro-

soft Word или Excel и желательно, чтобы этот процесс происходил при минимуме написания кода. Для этого существуют фильтры в RaveReports — это компоненты, которые позволяют сохранять отчет в различных форматах.

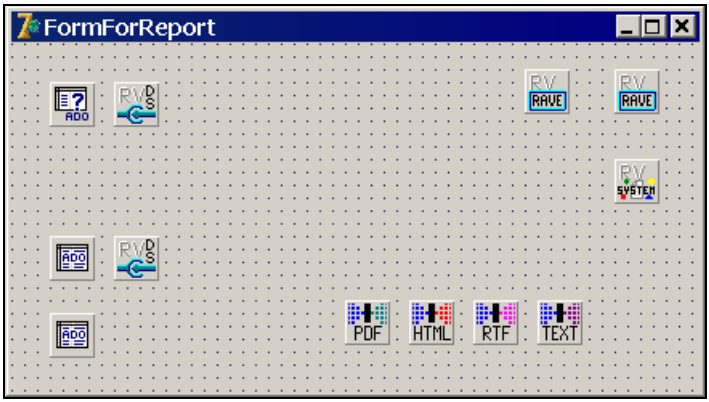


Рис. 7.39. Форма `FormForReport` с размещенными на ней фильтрами

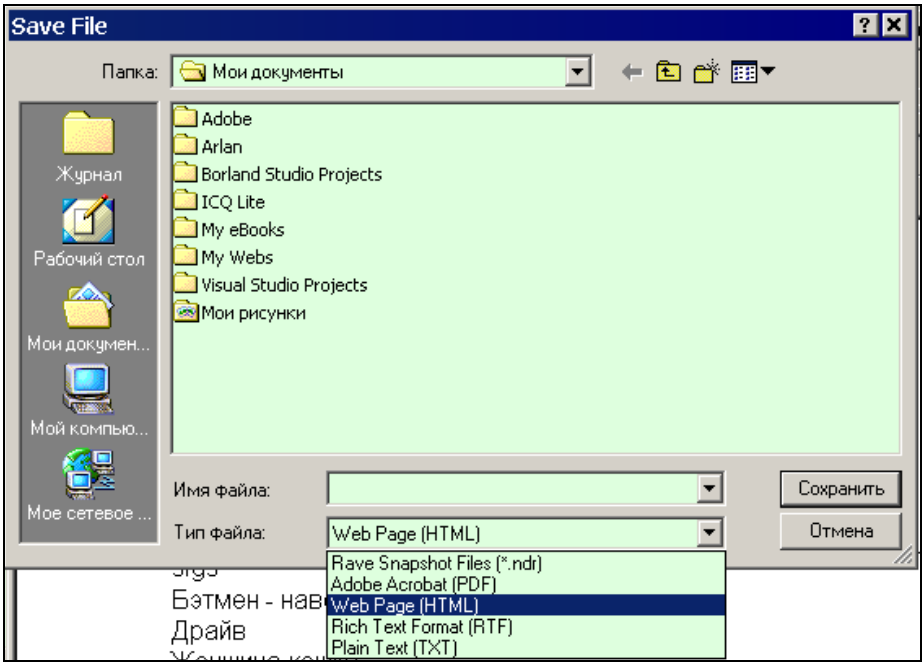


Рис. 7.40. Окно **Save File** при выборе формата сохранения отчете

Для этого их просто надо разместить на форме, которая содержит компоненты для работы с отчетом RaveReports, и новые возможности станут доступны. К этим компонентам относятся:

- `RvRenderPDF` — сохранение отчета в формате PDF;
- `RvRenderHTML` — сохранение отчета в формате HTML;
- `RvRenderRTF` — сохранение отчета в формате RTF;
- `RvRenderText` — сохранение отчета в текстовом формате.

Размещаем все эти компоненты на форме `FormForReport` (рис. 7.39).

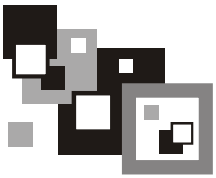
Теперь запускаем приложение, формируем отчет и нажимает кнопку **Save** с изображением дискеты, открываем выпадающий список **Тип файла** и видим, что количество возможных форматов увеличилось (рис. 7.40).

На компакт-диске в каталоге ГЛАВА 7\SOURCE можно увидеть пример данной программы.

В каталоге ГЛАВА 7\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 7\VIDEO можно посмотреть видеоролики, посвященные теме, рассмотренной в данной главе.

Глава 8



FastReport

8.1. Вступление

Данная система предоставляет широкий спектр возможностей по созданию отчетов и обладает, в то же время, интуитивно понятным интерфейсом, что, несомненно, позволяет ей занимать лидирующие позиции среди систем построения отчетов.

Замечание

С FastReport поставляется отличная справка, она расположена в папке C:\Program Files\FastReports\FastReport\help\russian (если при инсталляции FastReport путь установки был оставлен по умолчанию), файлы: Генератор отчетов.chm и Fruser.hlp.

FastReport можно скачать с официального сайта **http://www.fastreport.ru**. Этот генератор отчетов платный, но его можно использовать в своем обучении, только отчеты будут печататься с надписью **No Register** в левом верхнем углу и некоторые функции будут ограничены.

8.2. Установка FastReport

Для установки необходимо сначала скачать дистрибутив с официального сайта.

Замечание

В данной главе рассматривается работа с FastReport версии 2.5, в настоящее время на официальном сайте доступна более новая, третья версия. Работа с ней практически ничем не отличается, хотя появились некоторые нововведения. Появилась новая вкладка с компонентами (рис. 8.1). В дизайнере отчетов появился инспектор отчетов, такой же, как в RaveReports — он находится в правом верхнем углу (рис. 8.2). Также появился фильтр экспорта в XML.

Запускаем исполняемый файл из скачанного дистрибутива. На экране появится окно выбора языка, на котором будет происходить общение с масте-

ром установки, необходимо выбрать тот язык, который вам нужен, я выбрал русский (рис. 8.3).

Рис. 8.1. Новая вкладка в FastReport 3 (Delphi 2005)

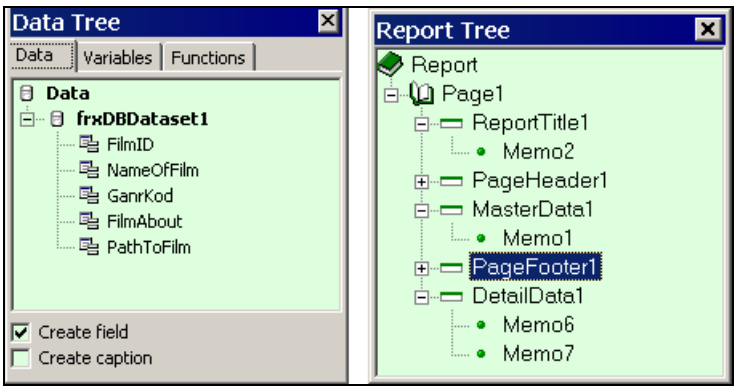


Рис. 8.2. Инспектор отчета

Нажимаем **Далее**. Появляется окно **Добро пожаловать**, в котором дается общая информация по установке, нажимаем **Далее**. Появляется окно **Информация**, в котором сообщаются данные по FastReport. Нажимаем **Далее**, появляется окно **Лицензионное соглашение**, в котором необходимо установить флаг **Да я согласен со всеми условиями данного лицензионного соглашения**. Нажимаем **Далее**. Появится окно **Тип установки**, в котором необходимо выбрать тип установки (рис. 8.4).

- **Custom** — выборочная;
- **Full** — полная.

Предлагаю выбрать пункт **Full Install** и нажать **Next**, появится окно **Выбор каталога установки**, предлагаю оставить предложенный по умолчанию каталог и нажать **Далее**. Появится окно **Установка...**, в котором необходимо нажать **Далее**, тем самым запустив процесс установки (рис. 8.5).

После того как установка завершится, в появившемся окне необходимо нажать **OK**.

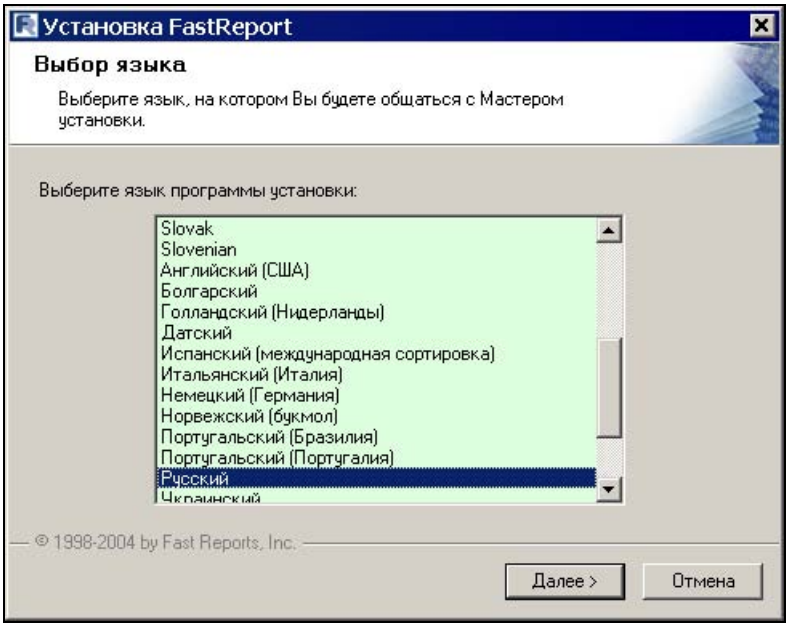


Рис. 8.3. Выбор языка установки FastReport

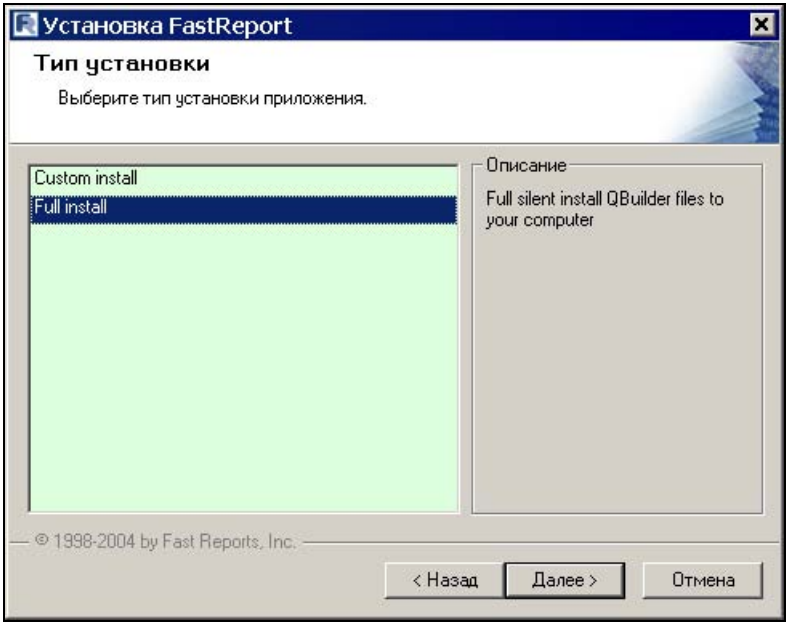


Рис. 8.4. Выбор типа установки

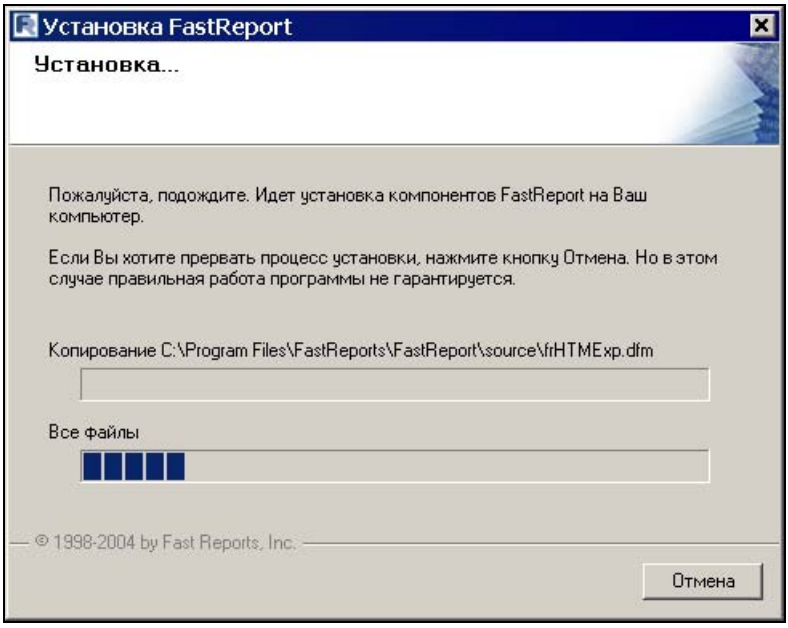


Рис. 8.5. Окно, отображающее процесс установки FastReport

8.3. Использование FastReport

После запуска Delphi можно обнаружить несколько новых вкладок, это:

- **FastReport** — здесь расположены основные компоненты для построения отчета;
- **FastReport Exports** — здесь расположены фильтры для экспорта отчета в различные форматы;
- **FR Tools** — здесь расположены элементы управления для использования в приложении (например, компонент `frSpeedButton` аналогичен обычному `SpeedButton` с вкладки **Additional**).

Рис. 8.6. Вкладка **FastReport** палитры компонентов Delphi 7

Рассмотрим небольшой пример. Открываем проект "Менеджер видеофильмов" разработанный в *главе 4* (можно также открыть проект из главы 7). Создаем новую форму, называем ее `FormForFastReport`, сохраняем под именем `UForFastReport`. Удаляем форму из списка автоматически создаваемых форм, подключаем модуль `UDM`.

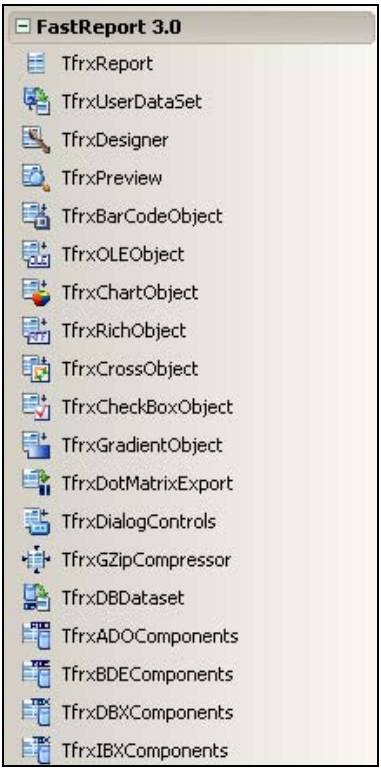


Рис. 8.7. Вкладка **FastReport** палитры компонентов Delphi 2005

Замечание

Не обязательно создавать для отчета новую форму, в данном случае это делается для наглядности и для более точного понимания читателем излагаемого материала.

Помещаем на форму компонент `ADOQuery`, называем его `QFastRepSecurityTable`, подключаем его к `ADOConnection1`.

В свойстве `SQL` пишем запрос выборки всех фильмов из таблицы SecurityTable:

```
SELECT * FROM SECURITYTABLE
ORDER BY UserLogin
```

Необходимо создать объекты-столбцы в редакторе столбцов для `QFastRepSecurityTable`. Помещаем на форму компоненты `frDBDataSet` и `frReport` (рис. 8.8).

Замечание

В FastReport 2.5 название всех компонентов начинается с `fr`, в FastReport 3.0 название всех компонентов начинается `frx`. Нужно учитывать этот момент при работе с данной системой построения отчетов.

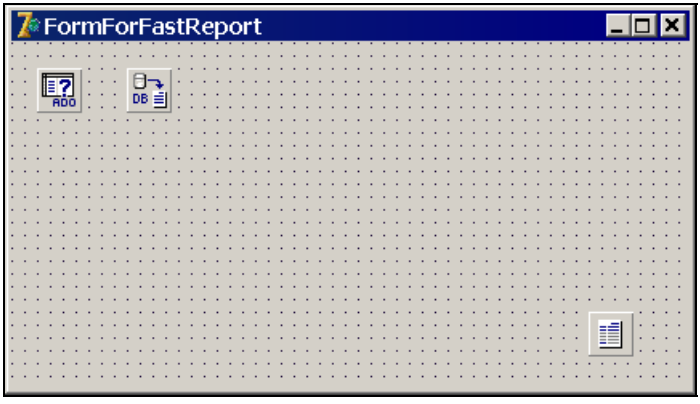


Рис. 8.8. Форма `FormForFastReport` с размещенными на ней компонентами построения отчета

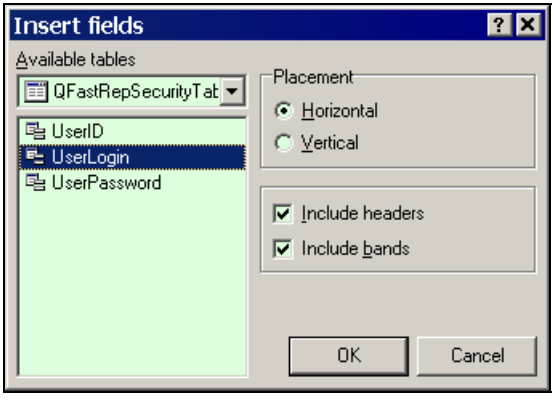


Рис. 8.9. Добавление атрибута в отчет

В свойство `DataSet` компонента `frDBDataSet` устанавливаем `QFastRepSecurityTable`. Устанавливаем в `True` свойство `StoredInDfm` компонента `frReport`. Производим двойной щелчок левой кнопкой мыши на компоненте `frReport`, что приведет к загрузке дизайнера отчетов. Необходимо нажать кнопку **Insert DB Fields**, расположенную в панели инструментов (последняя кнопка, если считать слева направо). Появится окно **Insert fields**, в выпадающем списке **Available tables** необходимо выбрать `QfastRepSecurityTable`, затем выбрать атрибут `UserLogin`. Нажимаем **OK**.

После этого окно дизайнера примет вид как на рис. 8.10.

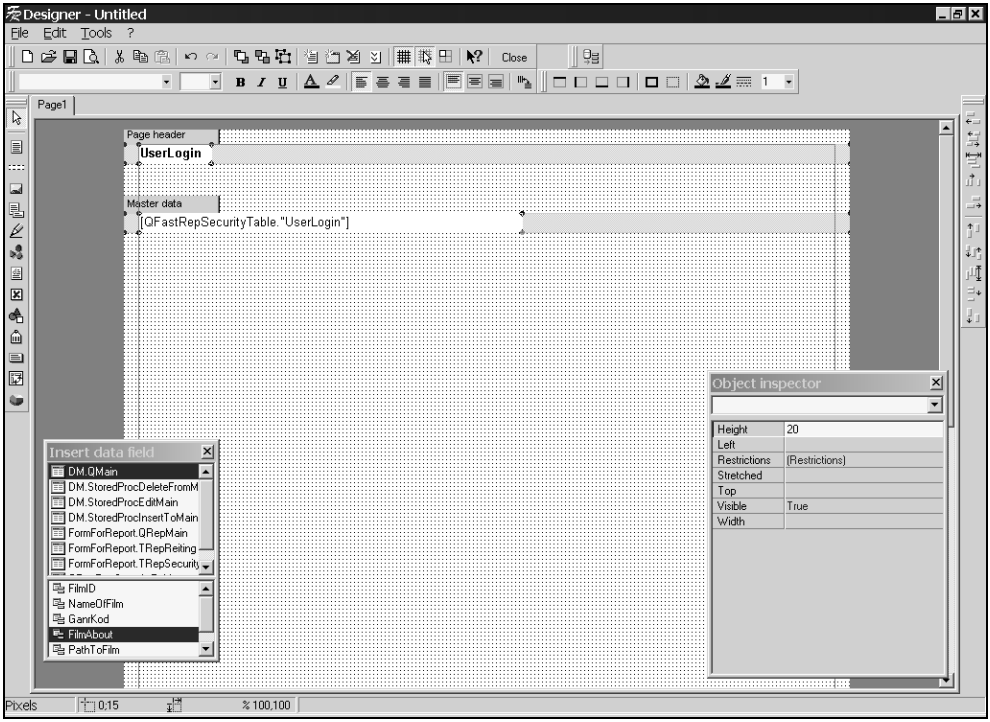


Рис. 8.10. Окно дизайнера отчета

Закрываем дизайнер. Переходим на форму `MainForm`, добавляем кнопку `Button` с текстом "Отчет в FastReport" (рис. 8.11).

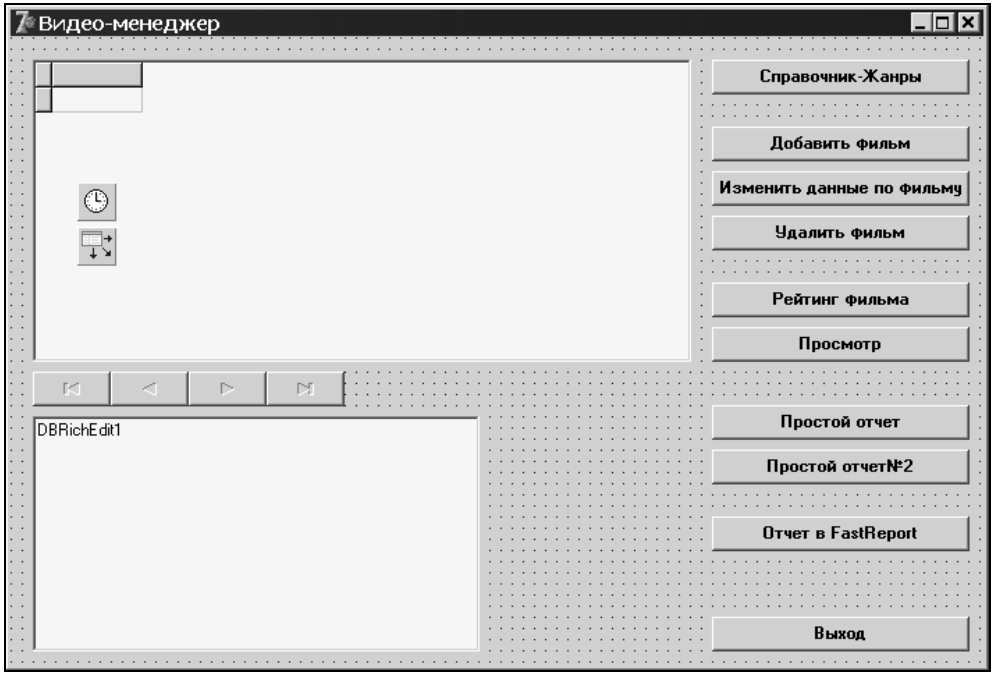


Рис. 8.11. Добавление кнопки вызова отчета на форму `MainForm`

Для данной кнопки пишем следующий код вывода отчета на экран (необходимо будет подключить модуль `UForFastReport`):

```
FormForFastReport:=TFormForFastReport.Create(self);

 {Открываем набор данных}
 FormForFastReport.QFastRepSecurityTable.Open;
 {Формируем отчет}
 FormForFastReport.frReport1.ShowReport;
 {Закрываем набор данных}
 FormForFastReport.QFastRepSecurityTable.Close;

 FormForFastReport.Close;
```

Компилируем программу, нажимаем кнопку **Отчет в FastReport** и наслаждаемся полученным результатом (рис. 8.12).

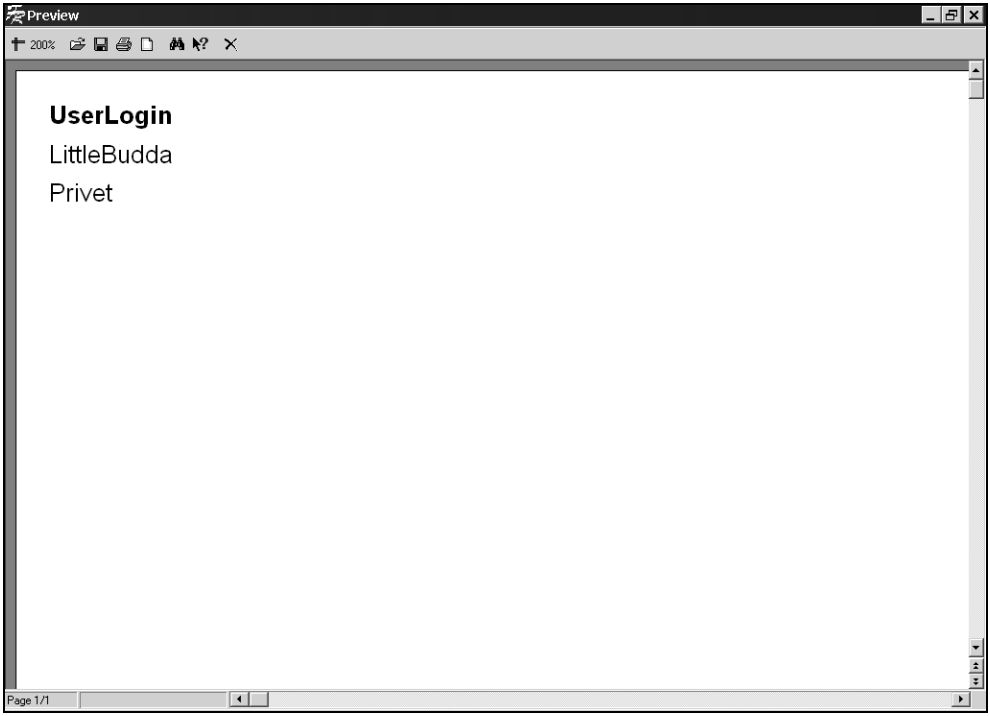


Рис. 8.12. Отчет, сформированный средствами FastReport

8.4. Принцип построения отчета

Основным компонентом при построении отчета является `frReport`. Именно с помощью него осуществляется формирование отчетов.

Отчеты могут поставляться как в виде отдельных файлов с расширением frp, так и включаться в исполняемый файл. За это отвечает свойство `StoreInDFM`; если оно установлено в `True`, то отчет будет включаться в исполняемый файл.

Создание отчета производится в дизайнере отчетов, который вызывается двойным щелчком левой кнопкой мыши на компоненте `frReport` или одинарным щелчком правой и выбором пункта **Design Report** в выпадающем меню. Вид дизайнера отчетов представлен на рис. 8.13.

В верхней части расположена панель инструментов, где используются стандартные пиктограммы для таких функций, как: создать новый отчет, открыть, сохранить и т. д. При нажатии на кнопку **Preview report** (лист бумаги с лупой на нем) можно посмотреть разработанный отчет.

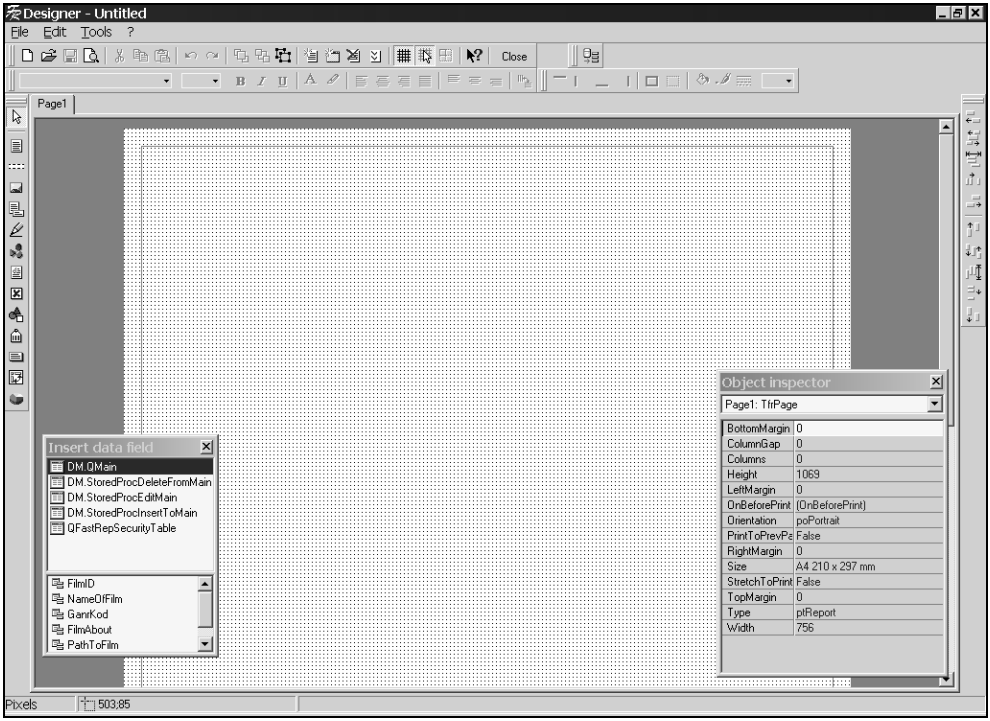


Рис. 8.13. Дизайнер отчетов FastReport

Замечание

При просмотре отчета из дизайнера отчета те наборы данных, которые используются в данном отчете, будут автоматически открыты (их свойство `Active` установлено в `True`); для того, чтобы их закрыть, придется вручную изменить свойство `Active`.

При нажатии на кнопку **Insert DB Fields** появляется окно **Insert fields** для указания тех источников данных, информация из которых будет использоваться в отчете.

В выпадающем списке **Available tables** выбирается набор данных, ниже выбираются атрибуты. Если установлен флаг **Include headers**, то вместе с атрибутом будет добавляться и его название. Если установлен флаг **Include bands**, то компоненты для отображения данных будут сразу размещаться на бендах.

Замечание

Для того чтобы лучше понять назначение элементов окна **Insert fields**, советую поэкспериментировать с настройкой элементов окна и таким образом попробовать создать несколько вариантов одного и того же отчета.

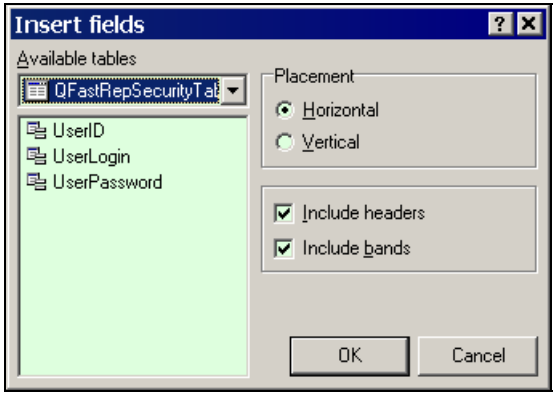


Рис. 8.14. Окно добавления атрибутов в отчет

В окне дизайнера также могут присутствовать:

- палитра компонентов — по умолчанию расположена в левой части окна, включает все основные компоненты для построения отчета (рис. 8.15);
- панель для выравнивания элементов отчета — по умолчанию расположена справа, предназначена для выравнивания и группировки элементов отчета (рис. 8.16);
- инспектор объектов — позволяет настраивать компоненты, изменяя их свойства (вызывается выбором пункта меню **Tools\Toolbars\Object Inspector**) (рис. 8.17);
- панель **Insert data field** — позволяет методом перетаскивания добавлять атрибуты в отчет (вызывается выбором пункта меню **Tools\Toolbars\Insert data field**) (рис. 8.18).

Рис. 8.15. Палитра компонентов FastReport

Рис. 8.16. Панель для выравнивания элементов

Структура отчета задается с помощью бендов. Бенд можно добавить, разместив соответствующий компонент (второй компонент в палитре компонентов в виде двух параллельных пунктирных линий) на холсте отчета. После этого

на экране появится окно **Insert new band**, в котором с помощью переключателя выбирается тип создаваемого бенда (рис. 8.19).

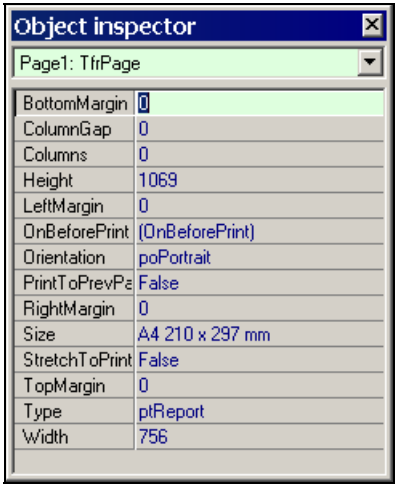


Рис. 8.17. Инспектор свойств FastReport

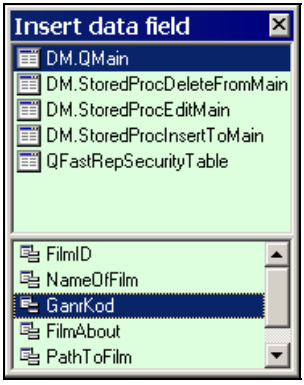


Рис. 8.18. Панель **Insert data field**

Мы не будем рассматривать назначение абсолютно всех бендов, так как это можно сделать, обратившись к справке, поставляемой с системой построения отчетов: файл Генератор отчетов.chm. Рассмотрим назначение основных бендов:

- `Report title` — заголовок отчета;
- `Report summary` — информация данной секции печатается один раз на последней странице отчета. Бенд используется для отображения аналити-

ческой информации, такой как количество записей, сумма по какому-либо атрибуту и т. д.;

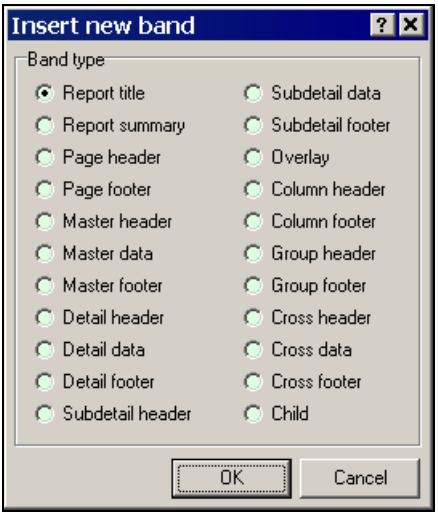


Рис. 8.19. Окно для выбора типа бенда

- `Page header` — верхний колонтитул;
- `Page footer` — нижний колонтитул;
- `Master header` — печатается перед бендом `Master data`;
- `Master data` — предназначен для отображения информации из набора данных;
- `Master footer` — печатается после бенда `Master data`;
- `Detail data` — предназначен для отображения информации из набора данных, связанного с тем, который используется для бенда `Master data`.

При выборе бенда, содержащего в своем названии слово "data", на экране появится дополнительное окно **Band data source** для выбора источника данных (рис. 8.20).

В данном окне отображаются доступные компоненты `frDBDataSet`, которые и служат источниками данных для отчета. Следует выбрать необходимый источник и нажать **OK**.

Замечание

Также в окне выбора источника данных присутствует так называемый виртуальный источник данных — Virtual Dataset. Он предназначен для тех случаев, когда данные берутся из нестандартного источника или на одном бенде будут

использоваться данные сразу из нескольких источников — `frDBDataSet`. Окно **Band data source** можно также вызвать открытием у бенда свойства `DataSource`.

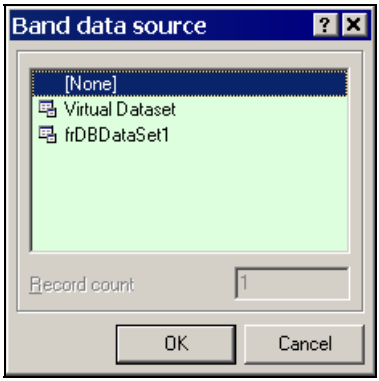


Рис. 8.20. Выбор источника данных для бенда

Далее на бенде необходимо разместить компоненты, предназначенные непосредственно для отображения данных.

Для того чтобы вывести отчет на экран, необходимо вызвать метод `ShowReport` компонента `frReport`. В общем случае код выглядит следующим образом:

```
frReport.ShowReport
```

8.5. Усложняем пример

Попробуем создать новый отчет, точно такой же, как предыдущий, только немного другим способом. Затем будем усложнять структуру отчета, чтобы научиться использовать различные функции системы построения отчетов FastReport.

Располагаем на форме `FormForFastReport` еще один компонент `frReport`, называем его `frIntrestingReport`, свойство `StoreInDFM` устанавливаем в `True`. Открываем дизайнер отчетов и приступаем к созданию отчета. Первым делом размещаем бенд `Report title`, на нем — `Rectangle object` (пиктограмма листа бумаги, заполненного строчками текста). Компонент `Rectangle object` предназначен для отображения текстовой информации. Причем можно выводить как статическую информацию, так и из источника данных. Также можно использовать скрипты — набор команд, которые могут обращаться к элементам отчета и их свойствам, а также производить манипуляции с внешним видом элементов отчета.

Производим двойной щелчок левой кнопкой мыши на `Rectangle object`, после чего появится окно **Text editor**.

Замечание

Это же окно можно вызвать, обратившись к свойству `Memo`, компонента `Rectangle object`.

В верхней части окна **Text editor** расположены функциональные кнопки, остальная часть окна делится на две части, верхняя половина предназначена для указания текста или источника данных, нижняя — для написания скрипта.

В верхней части окна вводим "ВИДЕО МЕНЕДЖЕР-отчет" (рис. 8.21).

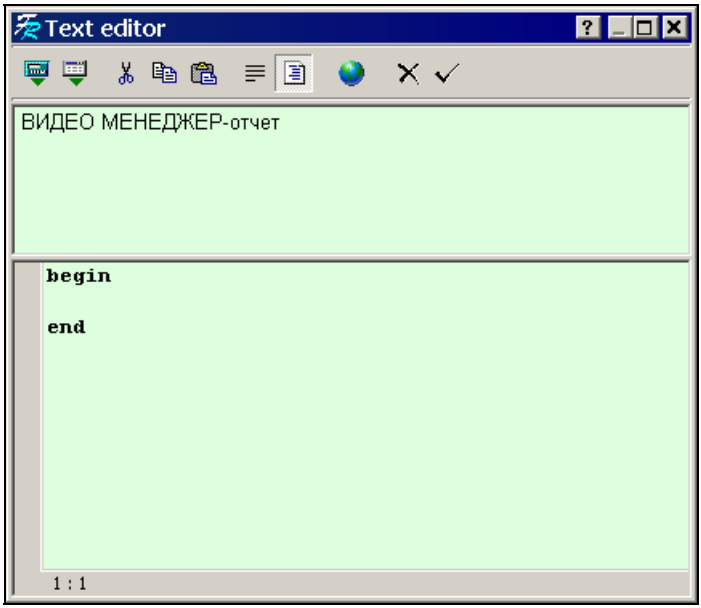


Рис. 8.21. Окно **Text editor**

Нажимаем **OK** (пиктограмма с галочкой).

Для компонентов, отображающих текст, можно настраивать параметры шрифта с помощью свойства `Font` либо с помощью панели **Text** (рис. 8.22).

С помощью панели **Rectangle** можно задавать отображение данных в виде таблицы, для этого необходимо выбрать размещенный в отчете компонент **Rectangle object** и задать с помощью кнопок панели **Rectangle** те грани, которые будут ограничены линиями, также можно задать толщину линий и стиль.

Рис. 8.22. Панель **Text**

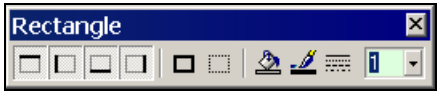


Рис. 8.23. Панель **Rectangle**

Размещаем в отчете еще один бенд — `Master data`, в появившемся окне **Band data source** выбираем источник данных — `frDBDataSet`. Далее на новом бенде располагаем `Rectangle object`. В появившемся окне **Insert data field** нажимаем кнопку **Insert data field** (вторая слева). Появится окно **Insert data field**, в котором необходимо выбрать набор данных и тот атрибут, данные которого будут отображаться компонентом **Rectangle object**. Необходимо сделать выбор, как показано на рис. 8.24, и нажать **OK**.

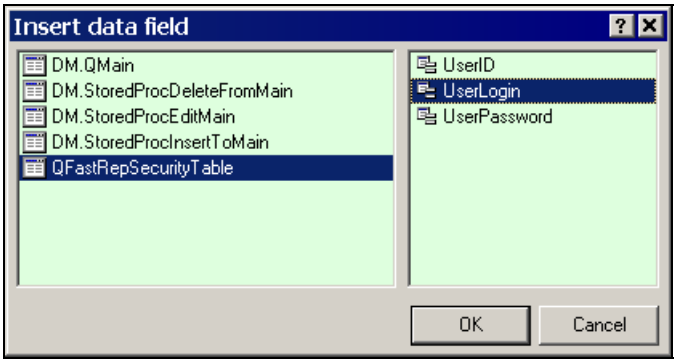


Рис. 8.24. Выбор атрибута,
информация из которого будет отображаться в отчете

После этого окно **Text editor** примет вид, как на рис. 8.25.

`[QFastRepSecurityTable."UserLogin"]` — данная запись указывает, что данные будут браться из набора данных `QFastRepSecutiryTable` и атрибута `UserLogin`. Имя атрибута заключается в кавычки, а вся запись берется в квадратные скобки.

Нажимаем **OK**. Можно посмотреть полученный отчет, он будет практически точно таким же, как в прошлый раз.

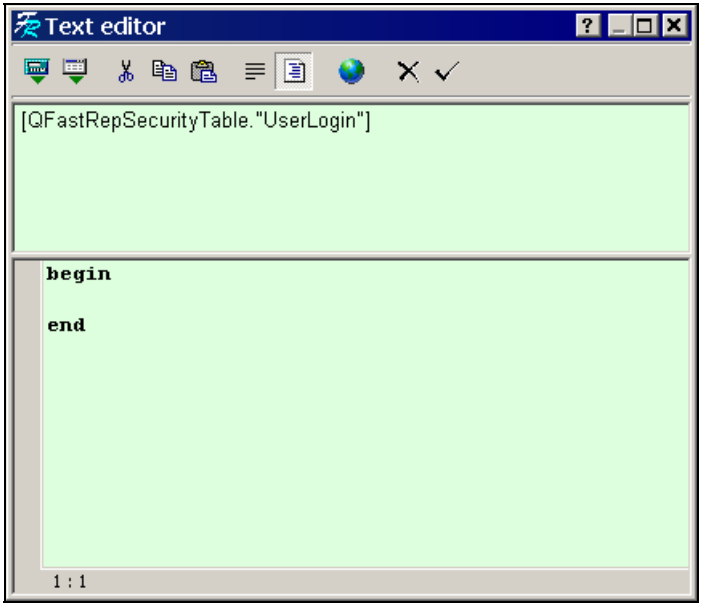


Рис. 8.25. Окно **Text editor**

Замечание

Чтобы просмотреть отчет, можно запустить приложение и выбрать соответствующую кнопку, либо выбрать пункт меню **File\Preview** в дизайнере отчетов. Но при этом необходимо будет учесть, что свойство `Active` набора данных `QFastRepSecutiryTable` будет установлено в `True`. Для того чтобы это изменить, необходимо будет модифицировать свойство вручную.

Размещаем между `Report title` и `Master data` еще один бенд — `Page header`, на нем — `Rectangle object`. В появившемся окне **Insert data field** вводим текст "Имя пользователя". В результате всех манипуляций у меня отчет в дизайнере выглядит так, как показано на рис. 8.26.

Размещаем в отчет еще один бенд — `Page footer`, на нем располагаем `Rectangle object`. В появившемся окне **Text editor** вводим текст "Сформирован", далее нажимаем кнопку **Insert expression**. Появится окно **Expression builder**, предназначенное для составления выражений, в которых можно использовать источники данных (кнопка **Data field**), системные переменные (кнопка **Variable**), функции (кнопка **Function**), а также логические операторы (больше, меньше, равно и т. д.).

Нажимаем кнопку **Variable**, появится окно **Variables**, в котором нам доступны системные переменные, необходимо выбрать **Date** (текущая дата) и нажать **OK** (рис. 8.28).

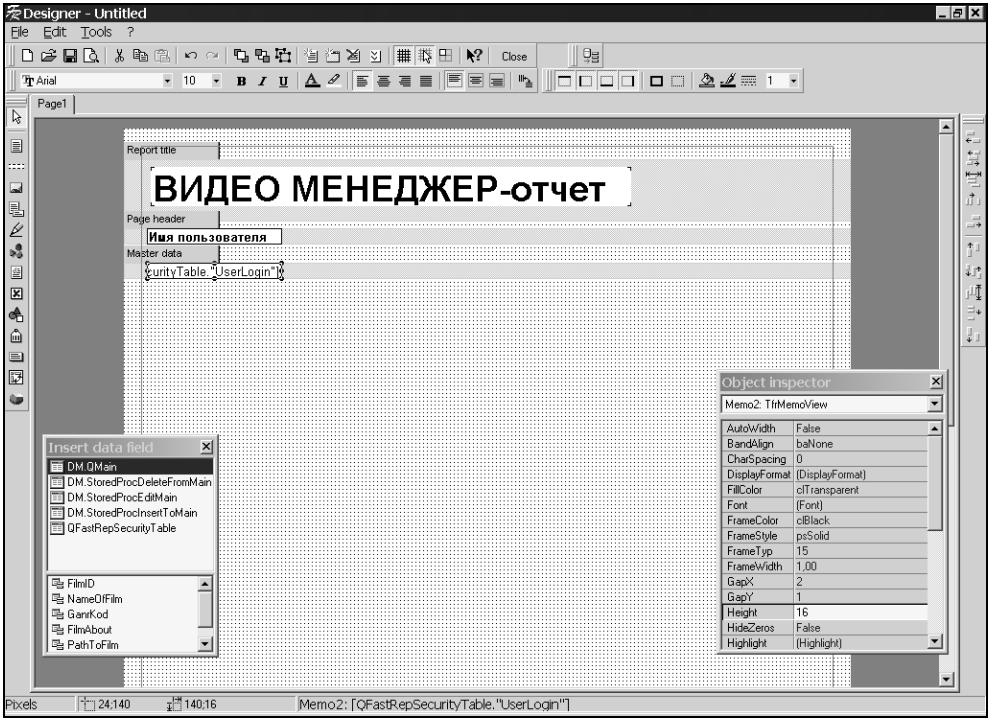


Рис. 8.26. Вид отчета в дизайнере

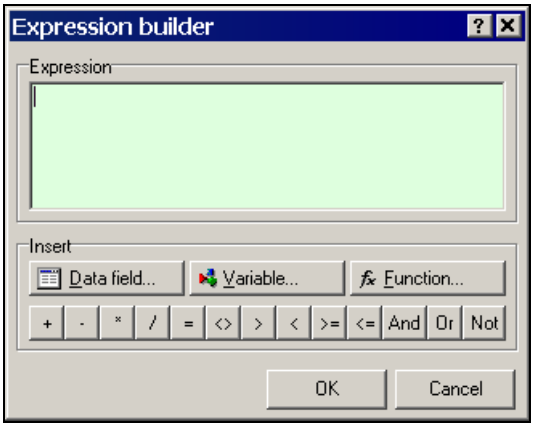


Рис. 8.27. Окно **Expression builder**, предназначенное для построения выражений

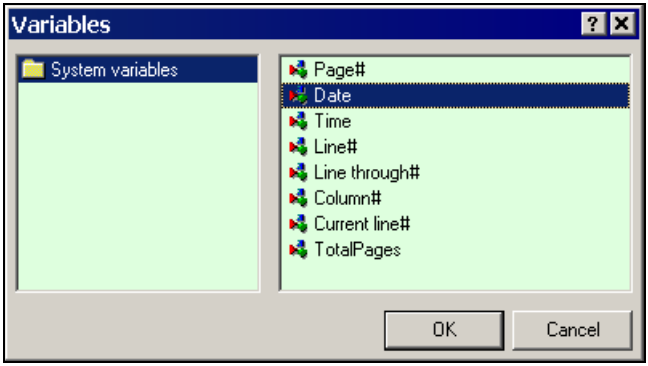


Рис. 8.28. Окно **Variables**

В окне **Expression builder** появится запись **Date**, нажмем **OK**, теперь в окне **Text editor** появилась запись вида: `[DATE]`.Необходимо привести ее к виду: "`Сформирован [DATE]`", где "`Сформирован`" — это текст, введенный нами, а `[DATE]` — системная переменная, которая в отчете будет заменена на текущую дату.

Замечание

Обратите внимание, что не нужно использовать оператор объединения — знак плюс. Достаточно правильно записать всю информацию в одну строку, там, где необходимо, разделяя ее пробелом.

Переменную `[DATE]`, как и другие системные переменные, можно добавить другим способом. Для этого необходимо в окне **Expression builder** нажать кнопку **Function**, что приведет к вызову окна **Available functions**, в котором выбираем группу **Date and time**. После этого станет доступна функция `DATE`. В нижней части окна **Available functions** можно увидеть комментарии для выбранной функции (рис. 8.29).

Добавляем еще один `Rectangle object` на бенд `Page footer`, который должен содержать текст "Страница №" и переменную `[PAGE#]`, отвечающую за текущий номер страницы. В результате всех манипуляций у меня отчет принял следующий вид (рис. 8.30).

Переходим к построению отчета типа master-detail. На форме `FormFastReport` размещаем компонент `DataSource`, называем его `DSFastRepSecurityTable`, в свойство `DataSet` устанавливаем `QFastRepSecurityTable`. Помещаем компонент `ADOTable`, называем его `TReiting`, связываем его с `ADOConnection1`. В свойстве `TableName` выбираем `Reiting`.

Следующим действием нам надо будет связать `QFastRepSecurityTable` и `TReiting` по коду читателя. Для этого в свойстве `MasterSource` компонента `TReiting` выбираем `DSFastRepSecurityTable`.

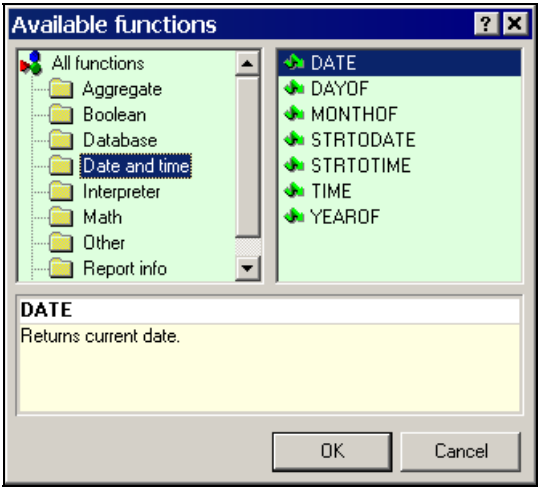


Рис. 8.29. Окно выбора системных функций

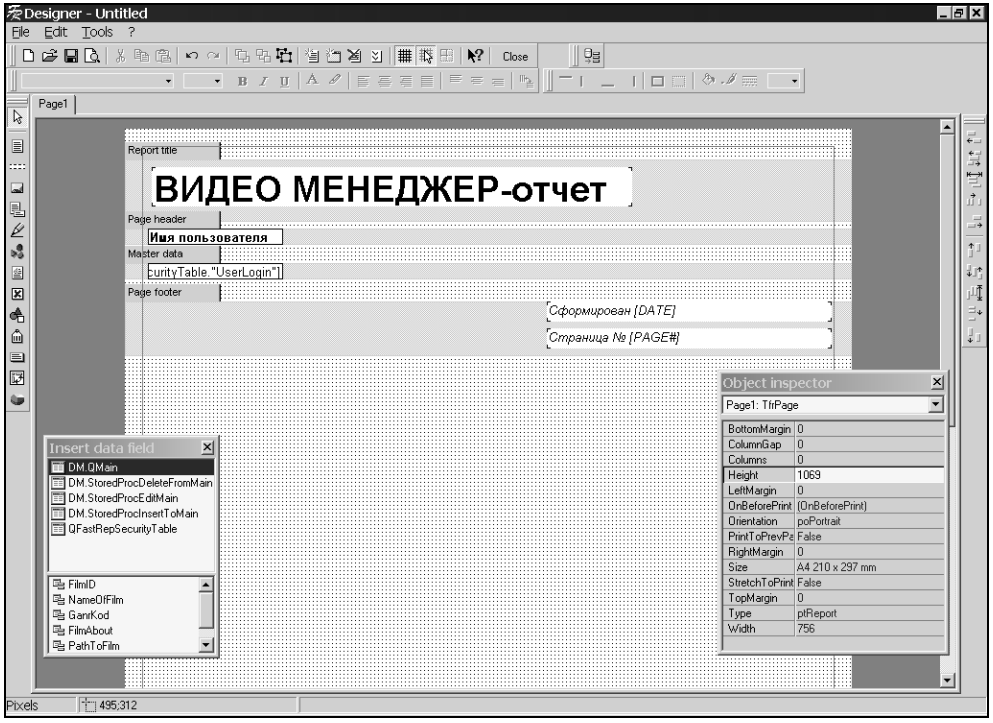


Рис. 8.30. Отчет в дизайнере с бендом `Page footer`

Открываем свойство `MasterFields` и в появившемся окне **Field Link Designer** в списке **Detail Fields** выбираем атрибут `UserKod`, а в списке **Master Fields** выбираем элемент `UserID` и нажимаем **Add**, далее **OK**.

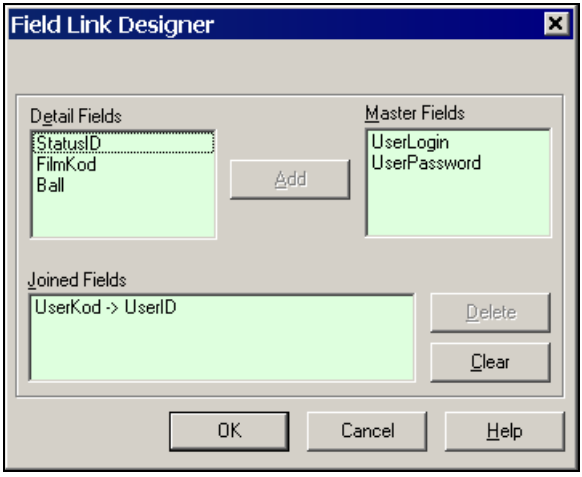


Рис. 8.31. Связывание наборов данных по коду пользователя

Далее располагаем `frDBDataSet`, в свойстве `DataSet` выбираем `TReiting`. Помещаем еще один `ADOTable`, называем его `TRepMain`, в свойстве `Connection` — `ADOConnection1`, в свойстве `TableName` выбираем `Main`. Для всех наборов данных создаем объекты-столбцы в редакторе столбцов. Вид формы после размещения всех компонентов представлен на рис. 8.32.

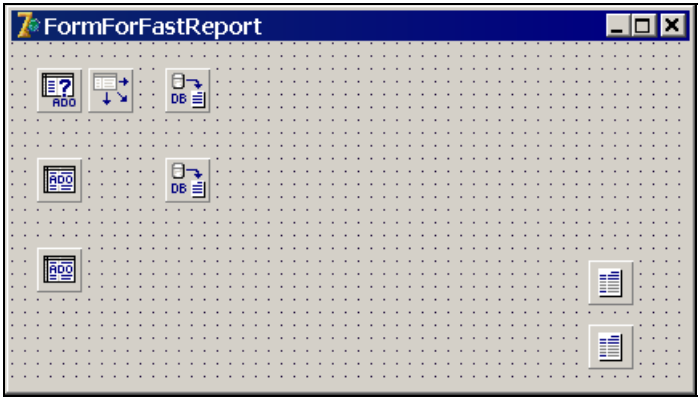


Рис. 8.32. Форма `FormForFastReport` с размещенными на ней компонентами для построения отчета

Теперь необходимо создать Lookup-поле для компонента TReiting в соответствии с рис. 8.33.

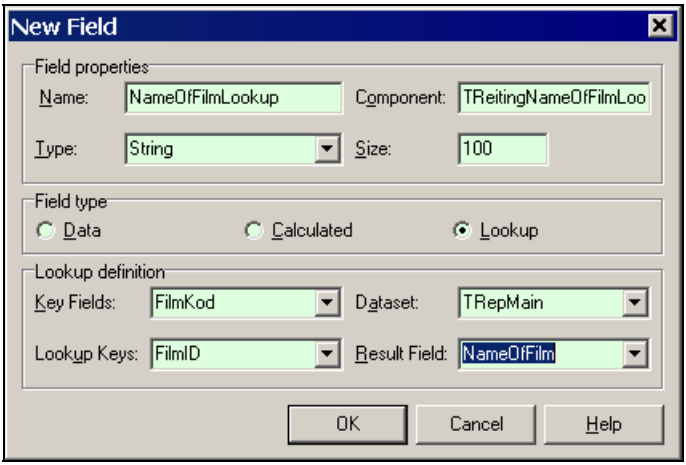


Рис. 8.33. Создание Lookup-поля

Двойным щелчком на frReport2 вызываем дизайнер отчетов. Размещаем между Master data и Page footer новый бенд — Detail data, связываем его с frDBDataSet2. На бенд помещаем 2 компонента Rectangle object, которые настраиваем на отображение атрибутов NameOfFilmLookup и Ball набора данных TRepMain. В результате отчет в дизайнере должен выглядеть как на рис. 8.34.

Замечание

Интерес представляет компонент Barcode object, который позволяет отображать штрихкоды. После размещения его на холсте отчета появляется окно **Barcode editor** (данное окно можно вызвать с помощью свойства Barcode) (рис. 8.35). В выпадающем списке **Type of barcode** выбирается тип штрих-кода, группа элементов **Rotation** позволяет с помощью соответствующего переключателя поворачивать сам штрихкод относительно отчета.

Располагаем на Page Footer компонент Barcode object, в группе элементов **Rotation** устанавливаем переключатель **270**. Открываем поле **Code**, появится уже знакомое нам окно **Expression builder**, в котором можно задать выражение, по которому будет строиться штрихкод. Я выбрал переменную DATE. В результате отчет принял вид, как на рис. 8.36.

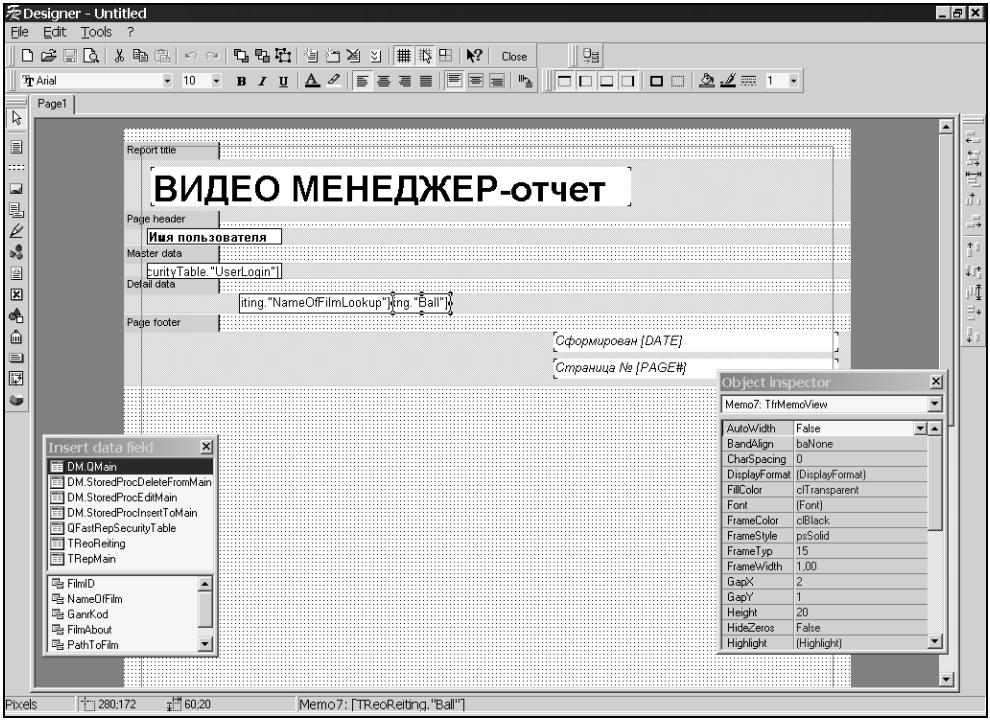


Рис. 8.34. Вид отчета в дизайнере с компонентом `Barcode object`

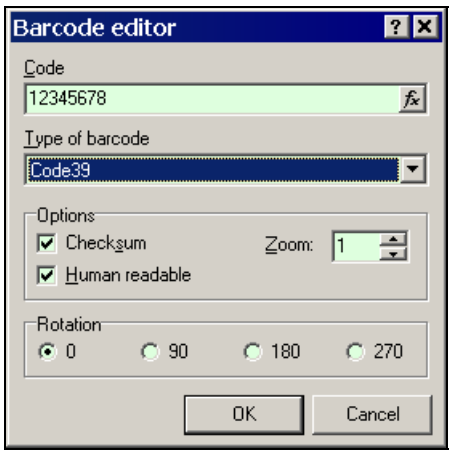


Рис. 8.35. Окно **Barcode editor**, предназначенное для выбора параметров штрихкода

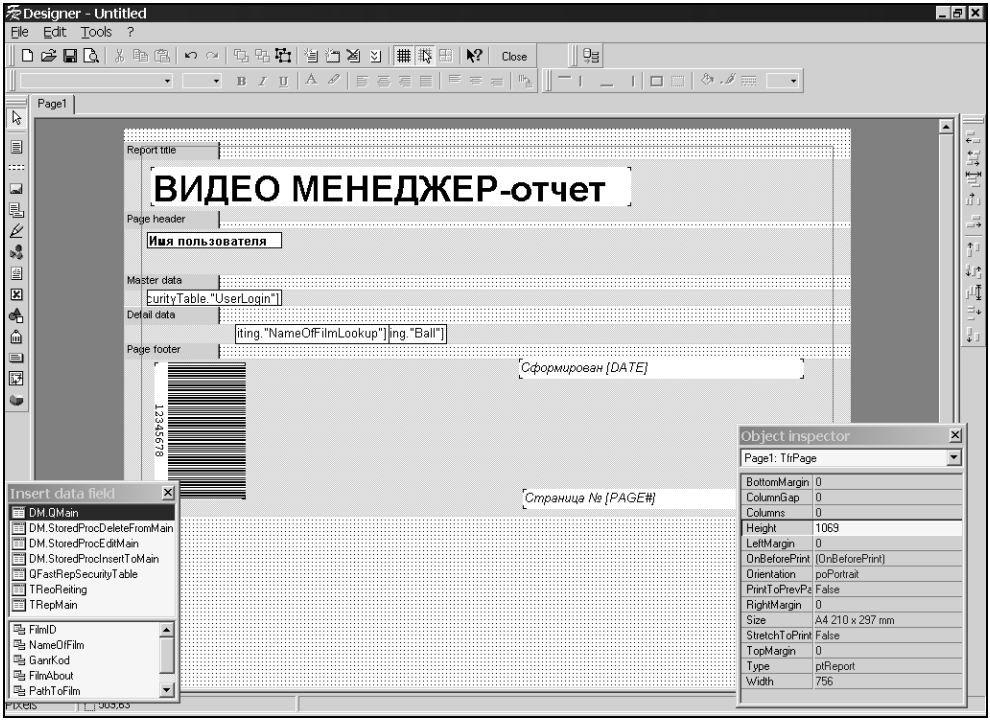


Рис. 8.36. Отчет, содержащий элемент штрихкод

Замечание

Если вы работаете с незарегистрированной версией FastReport, то при использовании в отчете компонента `Bar code object`, во время выполнения программы отчет формироваться не будет.

Для того чтобы отчет можно было редактировать в режиме RUN-TIME, то есть когда запущено приложение, необходимо разместить всего один компонент — `frDesigner`. После этого можно запустить программу, сформировать отчет, и после двойного щелчка на отчете появится дизайнер отчета, в котором можно редактировать сформированный отчет.

Замечание

В сложных отчетах очень тяжело предусмотреть все моменты, поэтому случается, что текст не помещается в отведенные ему границы. Также бывает, что данные неверно рассчитаны, и нужно исправить их всего в одном элементе, а ждать, пока отчет исправят, нет времени. Для этого и существует возможность запускать дизайнер в режиме RUN-TIME. Таким образом, пользователь может своими силами внести небольшие изменения в отчет.

8.6. Фильтры

Если во время выполнения программы в сформированном отчете в панели инструментов нажать кнопку сохранить с пиктограммой дискеты, то появится диалоговое окно **Save report**. Если попытаться выбрать формат для сохранения в выпадающем списке **Тип файла**, то можно увидеть, что сохранить отчет можно только в формате FastReport — FRP.

Для того чтобы отчет можно было сохранить в другом формате, необходимо воспользоваться компонентами, размещенными на вкладке **FastReport exports**. Причем для их функционирования необходимо просто расположить их на форме, содержащей компоненты для построения отчета (рис. 8.37 и 8.38).

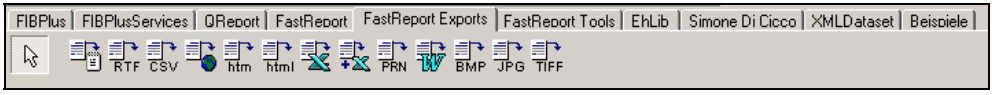


Рис. 8.37. Вкладка **FastReport Exports**, версия 2.5 (Delphi 7)

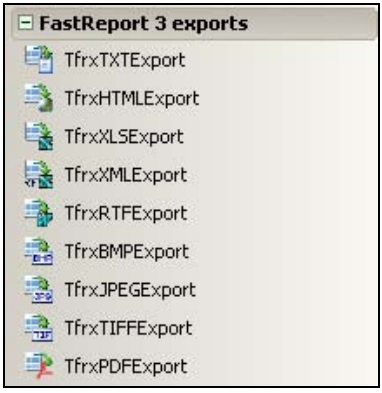


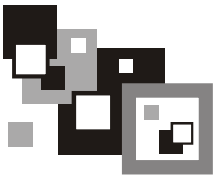
Рис. 8.38. Вкладка **FastReport 3 exports**, версия 3 (Delphi 2005)

На компакт-диске в каталоге ГЛАВА 8\SOURCE можно увидеть пример данной программы.

В каталоге ГЛАВА 8\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 8\VIDEO можно посмотреть видеоролики, посвященные теме, рассмотренной в данной главе.

Глава 9



XML

9.1. Вступление

XML — универсальный формат файла, содержащий структурированную информацию, как правило, используется в роли промежуточного звена для обмена данными между различным программным обеспечением (далее ПО) и информационными системами (далее ИС).

XML-файл — это обычный текстовый файл, но структурированный определенным образом. Технология XML используется как основа для хранения записей клиентских наборов данных в локальных файлах. Также XML — это одно из средств создания локальных однопользовательских БД, прежде всего при переносе Windows-приложений, работающих с БД, в среду Linux.

Одно из достоинств XML — независимость представления информации, которая позволяет этому формату широко использоваться в электронном бизнесе, а фирмам — производить независимое друг от друга ПО.

Замечание

Мы рассматриваем технологию XML в контексте баз данных. Если рассматривать XML с точки зрения Web-программирования, то определение будет немного отличаться: основная задача XML — отделить содержимое сайта от его дизайна.

Язык XML (eXtensible Markup Language) был разработан рабочей группой XML Working Group консорциума World Wide Web (W3C). Официальный сайт: **http://www.w3.org** (информация по XML: **http://www.w3.org/XML/**).

Замечание

Описание XML можно также найти на сайте **http://www.msdn.microsoft.com/xml/**.

Работа с набором данных в формате XML осуществляется с помощью компонента `ClientDataSet`. Для того чтобы можно было работать с XML, необходимо, чтобы в системе была зарегистрирована библиотека midas.dll.

9.2. Структура XML-файла

Язык XML имеет строго определенный синтаксис. Любой вложенный элемент должен быть полностью определен внутри элемента, в состав которого он входит. Это обусловлено тем, что структура XML-файла должна быть понятной для программы, которая обрабатывает и отображает информацию, содержащуюся в файле XML. Каждый элемент файла должен иметь начальный и конечный *теги* (тег — служебное слово, заключенное в угловые скобки).

В первой стоке XML-файла содержится заголовок, он выглядит следующим образом:

```
<?xml version="1.0" standalone="yes" ?>
```

Далее следует основной раздел, содержащий все данные, описывающие структуру таблицы и сами записи. Это тег первого уровня (верхнего уровня) — `<DATAPACKET>`. Тег `<DATAPACKET>` включает в себя два вида тегов:

- `<METADATA>` — предназначен для описания структуры таблицы;
- `<ROWDATA>` — тег, после которого следуют записи таблицы.

Тег `<METADATA>` может содержать теги `<FIELDS>` и `<PARAMS>`:

- `<FIELDS>` — описывает структуру атрибутов таблиц, каждый атрибут с его характеристиками описывается отдельным вложенным тегом;
- `<PARAMS>` — не имеет вложенных тегов, в нем содержится служебная информация, например, о начальном значении счетчика, предназначенного для атрибута, у которого значения должны присваиваться автоматически.

Тег `<ROWDATA>` содержит теги `<ROW>`, которые, в свою очередь, предназначены для описания каждой записи таблицы в отдельности.

Предположим, что у нас есть файл Reader.XML, который описывает структуру таблицы Reader (табл. 9.1 и 9.2).

Таблица 9.1. *Таблица "Читатели" (Reader)*

Имя атрибута	Описание	Тип данных
`ReaderID`	Поле-счетчик	Счетчик
`FIO`	ФИО читателя	Текст(100)
`Pasport`	Номер и серия паспорта	Текст (10) Текст, потому что в серии паспорта содержатся символы

Таблица 9.2. *Данные таблицы "Читатели" (Reader)*

ReaderID	FIO	Pasport
1	Никитенко А. И.	1A2TY
2	Иванов В. В.	1B2345CA
3	Семенов И. П.	234ADE
4	Овсянка М. М.	7ER345G
5	Жилина Н. А.	78GH7D8

Для того чтобы открыть файл Reader.XML, можно воспользоваться браузером Internet Explorer, но предварительно нужно будет поменять заголовок в файле следующим образом (это действие вызвано тем, что в файле содержатся русские буквы, а без явного указания кодировки такой файл просмотреть не удастся):

```
<?xml version="1.0" encoding="Windows-1251"?>
```

После открытия файла можно будет увидеть следующий результат:

```
<?xml version="1.0" standalone="yes"?>
<DATAPACKET Version="2.0">

<METADATA>
<FIELDS>
<FIELD attrname="ReaderID" fieldtype="i4" readonly="true"
 SUBTYPE="Autoinc"/>
<FIELD attrname="FIO" fieldtype="string" WIDTH="100"/>
<FIELD attrname="Pasport" fieldtype="string" WIDTH="10"/>
</FIELDS>

<PARAMS AUTOINCVALUE="8"/>

</METADATA>

<ROWDATA>
<ROW ReaderID="1" FIO="Никитенко А.И." Pasport="1A2TY"/>
<ROW ReaderID="2" FIO="Иванов В.В." Pasport="1B2345CA"/>
<ROW ReaderID="3" FIO="Семенов И.П." Pasport="234ADE"/>
<ROW ReaderID="4" FIO="Овсянка М.М." Pasport="7ER345G"/>
<ROW ReaderID="5" FIO="Жилина Н.А." Pasport="78GH7D8"/>
</ROWDATA>

</DATAPACKET>
```

Предположим, мы внесли в этот файл еще две записи (табл. 9.3).

Таблица 9.3. *Новые данные для таблицы "Читатели" (Reader)*

ReaderID	FIO	Pasport
6	Сидоров М. К.	YY78JKL
7	Петренко Т. В.	J7K8I9FF

Теперь, если открыть файл Reader.XML в браузере, он будет иметь следующий вид:

```
<?xml version="1.0" standalone="yes"?>
<DATAPACKET Version="2.0">

<METADATA>
<FIELDS>
<FIELD attrname="ReaderID" fieldtype="i4" readonly="true"
 SUBTYPE="Autoinc"/>
<FIELD attrname="FIO" fieldtype="string" WIDTH="100"/>
<FIELD attrname="Pasport" fieldtype="string" WIDTH="10"/>
</FIELDS>

<PARAMS CHANGE_LOG="6 0 4 7 0 4" AUTOINCVALUE="8"/>

</METADATA>

<ROWDATA>
<ROW ReaderID="1" FIO="Никитенко А.И." Pasport="1A2TY"/>
<ROW ReaderID="2" FIO="Иванов В.В." Pasport="1B2345CA"/>
<ROW ReaderID="3" FIO="Семенов И.П." Pasport="234ADE"/>
<ROW ReaderID="4" FIO="Овсянка М.М." Pasport="7ER345G"/>
<ROW ReaderID="5" FIO="Жилина Н.А." Pasport="78GH7D8"/>
<ROW RowState="4" ReaderID="6" FIO="Сидоров М.К."
 Pasport="YY78JKL"/>
<ROW RowState="4" ReaderID="7" FIO="Петренко Т.В."
 Pasport="J7K8I9FF"/>
</ROWDATA>

</DATAPACKET>
```

Можно заметить, что в теге `<PARAMS>` появился новый параметр — `CHANGE_LOG`, который отвечает за журнал изменений (то есть те изменения, которые происходят с данными, хранящимися в формате XML, сохраняются в журнале изменений).

Замечание

Наличие журнала изменений сильно увеличивает размер файла, но зато всегда можно вернуть случайно удаленную информацию.

9.3. Практика

Для начала попробуем построить таблицу, которая будет храниться в XML-файле и иметь структуру, указанную в табл. 9.1.

Создаем новый проект, сохраняем его под именем `Xml_proj`, главную форму называем именем `MainForm`, сохраняем ее под именем `UMain`. Располагаем на форме компонент `ClientDataSet`, открываем его свойство `FieldDefs`, в результате на экране появится окно (рис. 9.1).

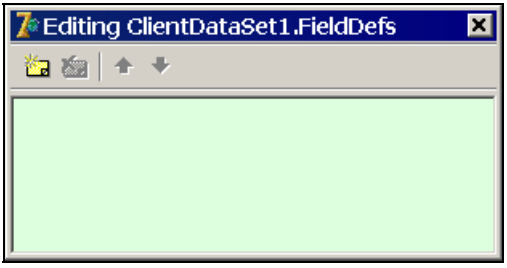


Рис. 9.1. Окно для задания атрибутов будущей таблицы

В данном окне мы будем создавать структуру будущей таблицы. Щелкаем в нем правой кнопкой мыши и в выпадающем меню выбираем пункт **Add**, после чего в окне появится новая запись вида: `0 - ClientDataSet1Field1` (рис. 9.2).

Обратим свое внимание на инспектор объектов и поменяем некоторые свойства:

- в свойстве `Name` задаем имя атрибута — `ReaderID`;
- в свойстве `Type` задаем тип атрибута — `ftAutoInc` (автоинкрементный).

Создаем остальные атрибуты (рис. 9.3):

- атрибут `FIO` — тип `String`, свойство `Size` задаем равное 100;
- атрибут `Pasport` — тип `String`, свойство `Size` задаем равное 10.

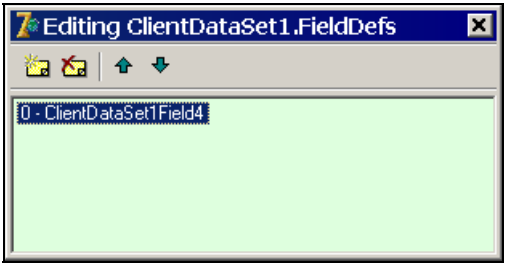


Рис. 9.2. Окно для задания атрибутов будущей таблицы с добавленной записью по умолчанию

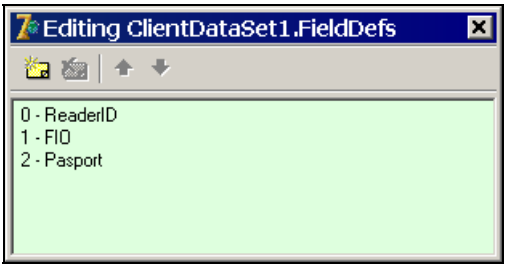


Рис. 9.3. Окно для задания атрибутов будущей таблицы с добавленными записями

Теперь производим щелчок правой кнопкой мыши на компоненте `ClientDataSet1` и в выпадающем меню выбираем пункт **Create DataSet**, что означает: создать набор данных (создать таблицу). Теперь необходимо еще раз щелкнуть по `ClientDataSet1` и в выпадающем меню выбрать пункт **Save to MyBase Xml Table**. Появится окно для сохранения файла, вводим `XMLReader` (рис. 9.4) и нажимаем кнопку **Сохранить**.

Располагаем компонент `DataSource`, который связываем с `ClientDataSet1` посредством свойства `DataSet`. Теперь помещаем компонент `DBGrid`, который связываем с `DataSource`.

Замечание

У компонента `ClientDataSet` так же, как и у аналогичных ему компонентов для работы с данными, есть свойство `Active`, с помощью которого можно открывать или закрывать набор данных.

У компонента `ClientDataSet1` свойство `Active` устанавливаем в `False`. Открываем свойство `FileName` и выбираем файл XMLReader.XML (файл, с которым будет работать `ClientDataSet1`). Помещаем кнопку `Button`, которая будет предназначена для открытия и закрытия таблицы Reader, называем ее `OpenCloseButton`, в свойстве `Caption` пишем "Открыть н.д.".

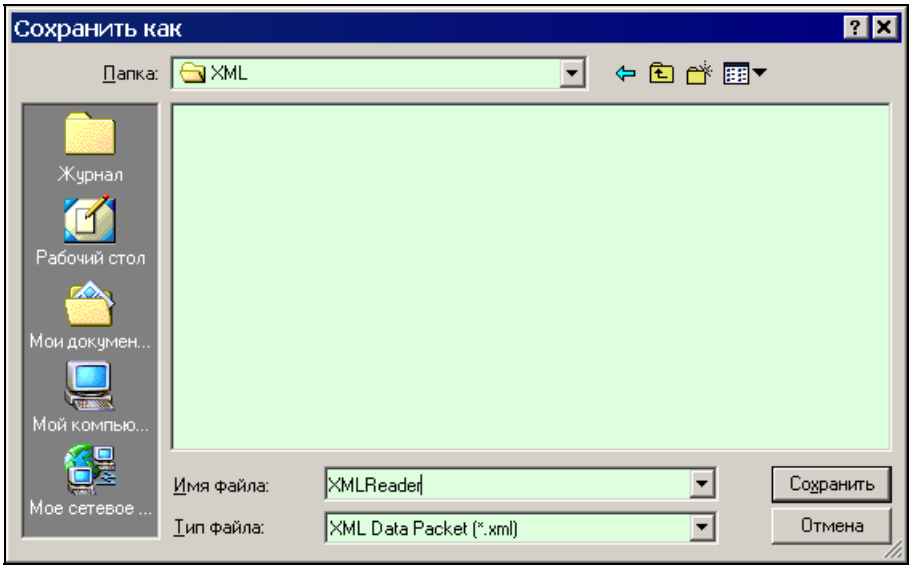


Рис. 9.4. Окно для сохранения файла

Для события `OnClick` пишем код:

```
procedure TMainForm.OpenCloseButtonClick(Sender: TObject);
begin
 {Закрытие набора данных}
 if OpenCloseButton.Tag=1 then
 begin
  try
   {Закрываем набор данных}
   ClientDataSet1.Close;
   {Меняем назначение кнопки}
   OpenCloseButton.Caption:='Открыть н.д.';
   {Указываем, что следующий раз при нажатии на кнопку
   набор данных будет открываться}
   OpenCloseButton.Tag:=0;
   {Выходим}
   exit;
  except
   ShowMessage('Не удалось закрыть набор данных');
   exit;
  end;
 end;
```

```
 {Открытие набора данных}
 ClientDataSet1.Open;
 {Меняем назначение кнопки}
 OpenCloseButton.Caption:='Закрыть н.д.';
 {Указываем, что следующий раз при нажатии на кнопку
 набор данных будет закрываться}
 OpenCloseButton.Tag:=1;
end;
```

Помещаем кнопку `Button`, называем ее `SaveButton`, в свойстве `Caption` пишем "Сохранить изменения" и пишем для нее следующий код:

```
procedure TMainForm.SaveButtonClick(Sender: TObject);
begin
 ClientDataSet1.SaveToFile();
end;
```

Для того чтобы сохранить данные, мы вызываем метод `SaveToFile()` компонента `ClienDataSet`. Мы не указываем никаких параметров, потому что у нас установлено свойство `FileName` и сохранение будет происходить в файл, который указан в данном свойстве.

Помещаем на форму еще одну кнопку `Button`, называем ее `UndoButton`, в свойстве `Caption` пишем "Отменить изменения" и создаем для нее следующий обработчик события `OnClick`:

```
procedure TMainForm.UndoButtonClick(Sender: TObject);
begin
 ClientDataSet1.UndoLastChange(true);
end;
```

Для отмены изменений необходимо вызвать метод `UndoLastChange`; если при этом в качестве параметра указано `True`, то курсор будет переведен на ту запись, изменения для которой были отменены, в противном случае курсор останется на месте.

Замечание

Если необходимо, чтобы журнал изменений не велся, необходимо установить свойство `LogChange` компонента `ClientDataSet` в `False`. Это можно сделать только во время выполнения программы. В общем виде код будет выглядеть следующим образом: `ClientDataSet1.LogChanges:=false`.

Помещаем на форму кнопку Button, называем ее ClearLogButton, в свойстве Caption пишем "Очистить журнал" и создаем для нее следующий обработчик события OnClick:

```
procedure TMainForm.ClearLogButtonClick(Sender: TObject);
begin
 ClientDataSet1.MergeChangeLog;
end;
```

Замечание

Для того чтобы метод MergeChangeLog выполнился успешно, необходимо, чтобы свойство Active компонента ClientDataSet, для которого вызывается данный метод, было установлено в True.

После выполнения всех вышеописанных действий форма MainForm у меня выглядит как на рис. 9.5.

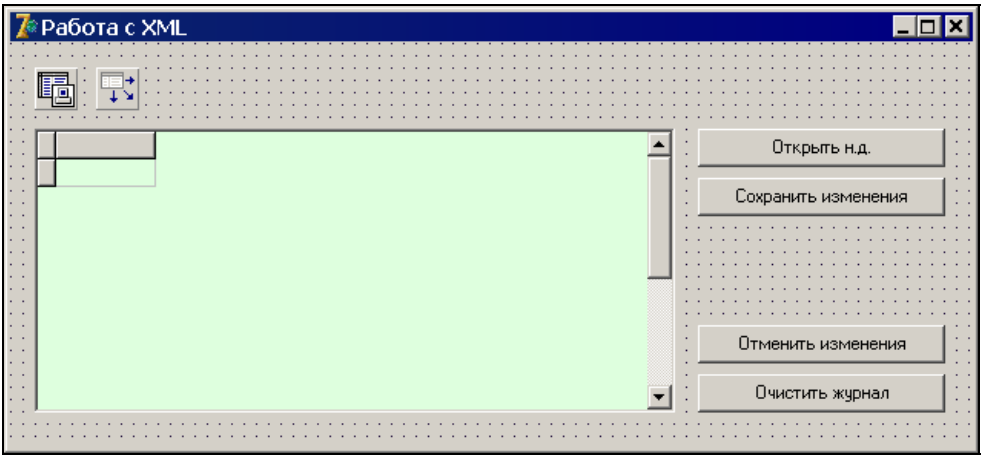


Рис. 9.5. Форма MainForm

Усложним пример. Располагаем на форме еще один компонент — ClientDataSet. Оставим его имя ClientDataSet2, данное по умолчанию, необходимо создать в нем таблицу "Книги" (табл. 9.4).

Таблица 9.4. *Таблица "Книги" (Book)*

Название атрибута	Описание	Тип данных
BookID	Поле-счетчик	Счетчик
ReaderKod	Код читателя	Числовой
NameOfBook	Название книги	Текстовый(100)

Производим щелчок правой кнопкой мыши на компоненте ClientDataSet2 и в выпадающем меню выбираем пункт **Create DataSet**. Теперь необходимо еще раз щелкнуть правой кнопкой мыши по ClientDataSet2 и в выпадающем меню выбрать пункт **Save to MyBase Xml Table**. Появится окно для сохранения файла, вводим XMLBook и нажимаем кнопку **Сохранить**.

Располагаем компонент DataSource, который связываем с ClientDataSet2 посредством свойства DataSet. Теперь помещаем компонент DBGrid, который связываем с DataSource2. У компонента ClientDataSet2 свойство Active устанавливаем в False. Открываем свойство FileName и выбираем файл XMLBook.XML.

Теперь свяжем таблицы Reader и Book между собой. Для этого в свойстве MasterSource компонента ClientDataSet2 устанавливаем DataSource1. Далее открываем свойство MasterFields компонента ClientDataSet2 и указываем в соответствии с рис. 9.6 атрибуты, по которым будет осуществляться связь.

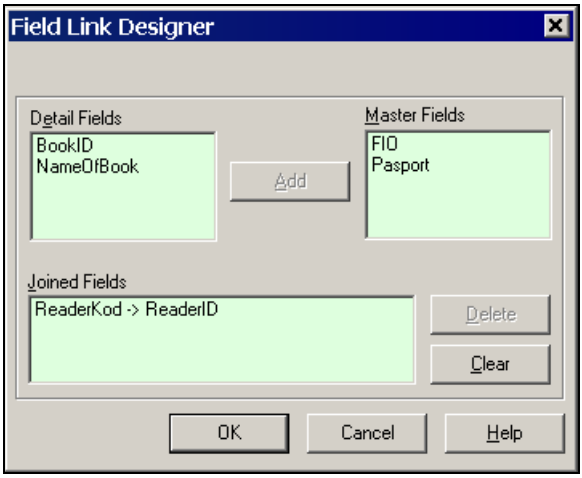


Рис. 9.6. Создание отношения между таблицами Reader и Book

Помещаем кнопку Button, которая будет предназначена для открытия/закрытия таблицы Book, называем ее OpenCloseBookButton, в свойстве Caption пишем "Открыть связанную таблицу". Для события OnClick создаем следующий обработчик:

```
procedure TMainForm.OpenCloseBookButtonClick(Sender: TObject);
begin
 {Закрытие набора данных}
 if OpenCloseBookButton.Tag=1 then
 begin
  try
   {Закрываем набор данных}
   ClientDataSet2.Close;
   {Меняем назначение кнопки}
   OpenCloseBookButton.Caption:='Открыть связанную таблицу';
   {Указываем, что следующий раз при нажатии на кнопку
   набор данных будет открываться}
   OpenCloseBookButton.Tag:=0;
   {Выходим}
   exit;
  except
   ShowMessage('Не удалось закрыть набор данных');
   exit;
  end;
 end;

 {Открытие набора данных}
 ClientDataSet2.Open;
 {Меняем назначение кнопки}
 OpenCloseBookButton.Caption:='Закрыть связанную таблицу';
 {Указываем, что в следующий раз при нажатии на кнопку
 набор данных будет закрываться}
 OpenCloseBookButton.Tag:=1;
end;
```

Располагаем еще одну кнопку, называем ее SaveAllButton, меняем свойство Caption, оно должно содержать текст "Сохранить все изменения", и для события OnClick пишем следующий код:

```
procedure TMainForm.SaveButtonClick(Sender: TObject);
begin
```

```
 ClientDataSet1.SaveToFile();
 ClientDataSet2.SaveToFile();
end;
```

Окончательный вариант формы `MainForm` представлен на рис. 9.7.

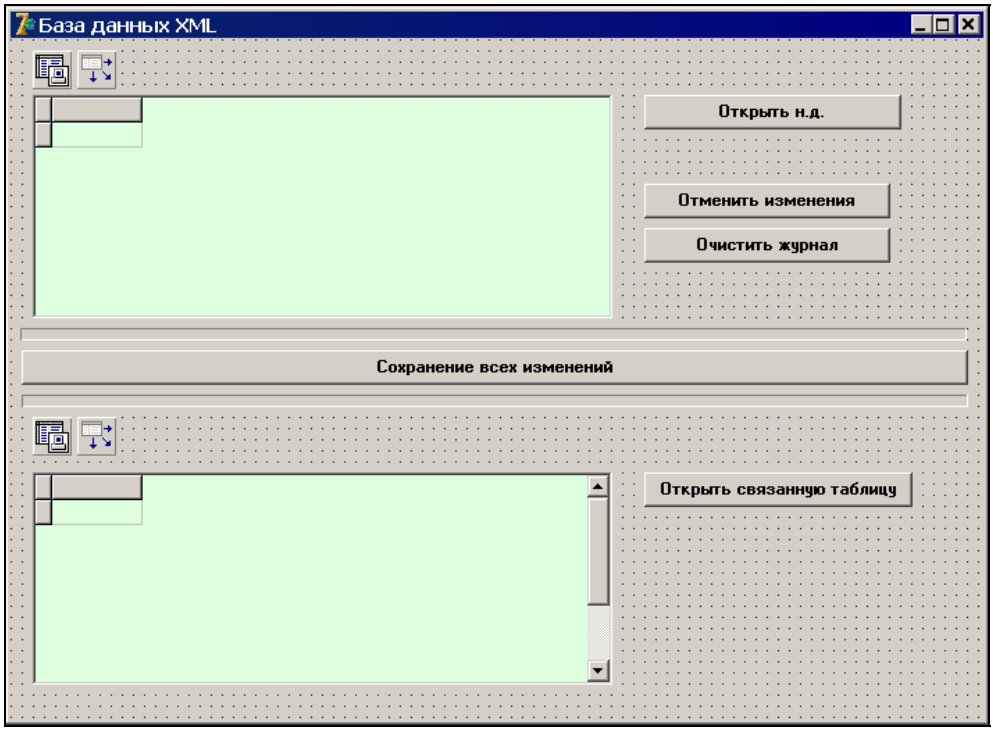


Рис. 9.7. Окончательный вариант формы `MainForm`

9.4. Полезный пример

Предположим, у нас стоит задача: для каждого читателя в конце года сохранять в XML-формате информацию о том, какие книги он прочитал. Допустим, что первоначально данные хранятся в студенческой библиотеке (программа Библиотекарь, разработанная в *главе 5*), и информацию и эти данные затем необходимо передать в другую ИС. Обмен как раз и происходит через XML-файлы.

Для реализации задания нам понадобятся два файла: первый будет содержать имена читателей, а второй — названия книг, которые они прочитали. Информация о книгах будет несколько избыточна из-за того, что мы будем хранить в базе полные названия книг, а не их коды.

- Информация о читателях будет храниться в файле XMLReader.XML, созданном ранее, в соответствии с табл. 9.1, только необходимо, чтобы атрибут `ReaderID` имел тип не счетчик, а обычный `Integer`, иначе не получится сохранить ссылочную целостность между таблицами Book и Reader.
- Информация о книгах будет храниться в файле XMLBook.XML, созданном ранее, в соответствии с табл. 9.4.

Создаем новый проект, предлагаю сохранить его в той же папке, что и предыдущий (у меня эта папка называется XML). Проект сохраняем под именем `TransformToXml_proj`, главную форму называем именем `TransformForm`, сохраняем под именем `UTransform`. Располагаем на форме компонент `IBDatabase`, который настраиваем на работу с базой Library.FDB, которую мы разработали в *главе 5*. Помещаем компонент `IBTransaction`, связываем его с `IBDatabase`. Размещаем компонент `IBQuery`, называем его `QReader`, связываем с компонентом `IBDatabase`, в свойство `SQL` пишем запрос выбора всех читателей:

```
SELECT * FROM READER
```

Вызываем редактор столбцов и создаем объекты-столбцы. Размещаем `DataSource` и `DBGrid`. Компонент `DataSource1` связываем с `QReader`, а `DBGrid` — с `DataSource1`. Помещаем еще один компонент `IBQuery`, называем его `QBook`, связываем с компонентом `IBDatabase`, в свойство `SQL` пишем следующий запрос:

```
SELECT distinct Book.Nameofbook, Abonement.Readerkod
FROM BOOK, ABONEMENT, MOVE
WHERE
              (Book.BookID=Move.BookKod)
                     AND
              (Move.AbonementKod=Abonement.AbonementID)
```

Данный запрос выбирает названия книг и коды читателей, которые их брали, с учетом связей между таблицами. Мы используем команду `distinct` для того, чтобы не выводить несколько раз дублирующие записи, которые могут возникнуть, если один и тот же читатель брал одну и ту же книгу неоднократно.

Вызываем редактор столбцов и создаем объекты-столбцы. Размещаем `DataSource` и `DBGrid`. Компонент `DataSource2` связываем с `QBook`, а `DBGrid` — с `DataSource2`. Теперь размещаем 2 компонента `Button`, которые должны содержать текст "Получить данные" и "Transform", называем их `GetDataButton` и `TransformButton` (рис. 9.8).

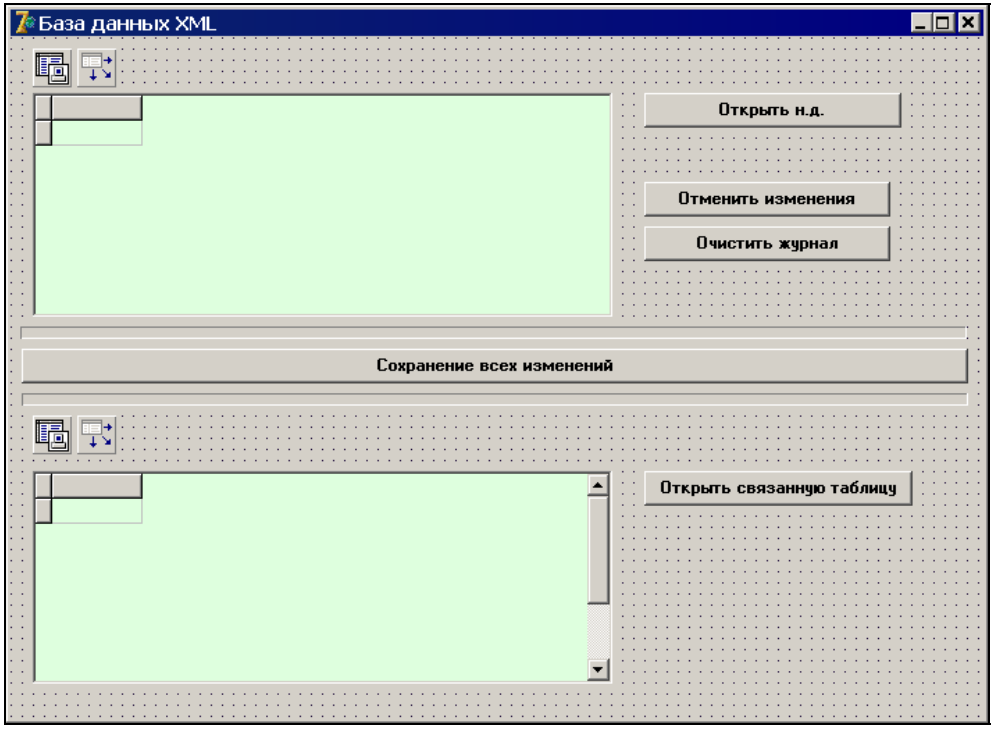


Рис. 9.8. Вид окна **База данных XML**

Для кнопки **Получить данные** пишем следующий код:

```
procedure TTransformForm.GetDataButtonClick(Sender: TObject);
begin
 {Открываем базу данных}
 IBDatabase1.Open;
 {Открываем наборы данных}
 QReader.Open;
 QBook.Open;
end;
```

Рядом с кнопкой **Transform** располагаем 2 компонента `ClientDataSet`, называем их `ClientDataSetReader` и `ClientDataSetBook`. Связываем их с файлами XML:

- в свойстве `FileName` компонента `ClientDataSetReader` указываем файл XMLReader.XML;
- в свойстве `FileName` компонента `ClientDataSetBook` указываем файл XMLBook.XML.

Создаем для ClientDataSetReader и ClientDataSetBook объекты-столбцы в редакторе столбцов. Для кнопки **Transform** пишем код для конвертации данных из СУБД Firebird в формат XML:

```
procedure TTransformForm.TransformButtonClick(Sender: TObject);
begin
 {Если соединение с базой не установлено
 или по какой-то причине один из наборов
 данных закрыт, то сообщаем об этом и выходим
 из процедуры}
 if (QReader.Active=false) or (QBook.Active=false) then
 begin
  ShowMessage('Необходимо открыть наборы данных');
  exit;
 end;

{-------------------Обрабатываем файл XMLReader.XML------}
 {Очищаем файл XMLReader.XML}
 ClientDataSetReader.Open;
 while not(ClientDataSetReader.RecordCount=0) do
   ClientDataSetReader.Delete;

 {Приступаем к добавлению данных}
 QReader.First;
 {Пока не дошли до конца, добавляем данные в XML таблицу}
 while not(QReader.Eof) do
 begin
  ClientDataSetReader.Insert;
  ClientDataSetReaderReaderID.Value:=QReaderREADERID.Value;
  ClientDataSetReaderFIO.Value:=QReaderFIO.Value;
  ClientDataSetReaderPasport.Value:=QReaderPASPORT.Value;
  QReader.Next;
 end;

 {Сохранение изменений}
 ClientDataSetReader.SaveToFile();
 {Очищаем журнал изменений}
 ClientDataSetReader.MergeChangeLog;
 {Закрытие набора данных}
 ClientDataSetReader.Close;
```

```
{-------------------Обрабатываем файл XMLBook.XML------}
 {Очищаем файл XMLBook.XML}
 ClientDataSetBook.Open;
 while not(ClientDataSetBook.RecordCount=0) do
   ClientDataSetBook.Delete;

 {Приступаем к добавлению данных}
 QBook.First;
 {Пока не дошли до конца, добавляем данные в XML-таблицу}
 while not(QBook.Eof) do
 begin
  ClientDataSetBook.Insert;
  ClientDataSetBookReaderKod.Value:=QBookREADERKOD.Value;
  ClientDataSetBookNameOfBook.Value:=QBookNAMEOFBOOK.Value;
  QBook.Next;
 end;

 {Сохранение изменений}
 ClientDataSetBook.SaveToFile();
 {Очищаем журнал изменений}
 ClientDataSetBook.MergeChangeLog;
 {Закрытие набора данных}
 ClientDataSetBook.Close;
end;
```

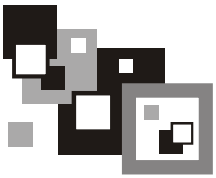
Теперь можно запустить программу и произвести конвертацию данных в XML-формат. После этого можно запустить проект `Xml_proj`, разработанный в *разд. 9.3*, и убедиться, что конвертация прошла успешно.

На компакт-диске в каталоге ГЛАВА 9\SOURCE можно увидеть пример данной программы.

В каталоге ГЛАВА 9\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 9\VIDEO можно посмотреть видеоролики, посвященные созданию разрабатываемого приложения.

Глава 10



Проектирование базы данных в среде ERwin

10.1. Вступление

В настоящее время разработка баз данных, как одного из основных элементов информационных систем, — это серьезная задача, требующая одновременной работы целого коллектива специалистов. При этом возникает ряд глобальных вопросов, которые требуют решения.

- Как максимально автоматизировать процесс построения базы данных?
- Как лучше документировать процесс разработки?
- Как поддерживать разработанную модель базы данных — то есть как вносить изменения и дополнения в структуру, затрачивая на это минимальное количество времени и сил?

Для решения этих вопросов предназначены *CASE-средства* (Computer Aided Software Engineering, разработка программного обеспечения с помощью компьютера). К таким средствам относится *ERwin*, работа с которым будет описана в данной главе.

Когда вы разрабатываете серьезную базу данных, то невозможно всю информацию хранить в голове или на листочке рядом, потому что рано или поздно запутаешься или встанет проблема донести эту информацию до других людей. Намного удобнее использовать для разработки и документирования процесса разработки специализированные средства, которые позволяют максимально сосредоточиться на разрабатываемой задаче, к таким средствам и относится ERwin. Он позволяет, в буквальном смысле, рисовать будущую базу данных, к тому же можно комментировать каждый элемент, добавлять произвольные фигуры и выделять рабочие области различными цветами. Можно просматривать построенную модель базы данных в различных вариантах. Все это позволяет не только разрабатывать, но и проводить презентации, а также использовать это средство при обучении специалистов, которые будут сопровождать или поддерживать разрабатываемую вами систему.

10.2. Основные понятия

В ERwin для проектирования баз данных используется ER-диаграмма (Entity-Relationship Diagram — диаграмма сущность-связь). Технология *ER-моделирования* была разработана П. Ченом в конце 1970-х годов. Основным понятием технологии является сущность, для визуализации которой служит прямоугольник. В ERwin используется два варианта ER-моделирования:

- IDEF1X — Integration DEFinition for Information Modeling (интегрированное описание для информационного моделирования);
- IE — Information Engineering (проектирование информации).

Основное различие между IDEF1X и IE заключается в том, что в них используются различные знаки (или нотации) для представления одной и той же модели. Для того чтобы показать это на примере, забежим немного вперед, поэтому, если не все будет понятно, то пугаться не стоит, далее мы поговорим об этом более подробно. Для построения модели базы данных в ERwin существует специальная панель **ToolBox**. ERwin позволяет разрабатывать два вида моделей: *логическую* (Logical) и *физическую* (Physical). Так вот, в зависимости от выбранного варианта построения модели — IDEFIX или IE — панель **ToolBox** будет иметь различные элементы для построения одних и тех же элементов, на рис. 10.1—10.6 можно увидеть разницу.

Рис. 10.1. Панель **ToolBox** для логической модели в варианте IDEF1X

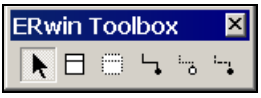


Рис. 10.2. Панель **ToolBox** для физической модели в варианте IDEF1X

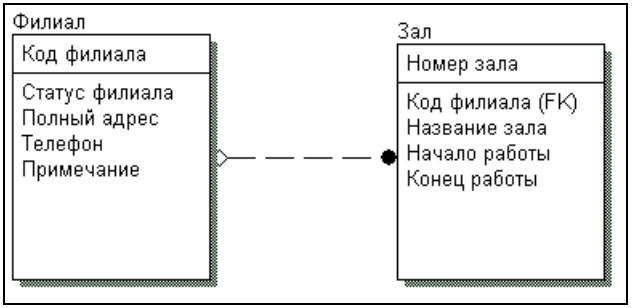


Рис. 10.3. Пример фрагмента логической модели в варианте IDEF1X

Рис. 10.4. Панель **ToolBox** для логической модели в варианте IE

Рис. 10.5. Панель **ToolBox** для физической модели в варианте IE

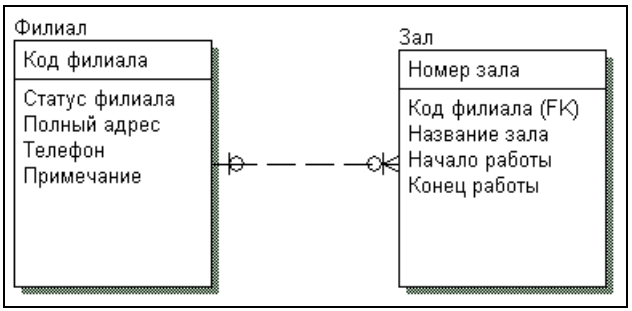


Рис. 10.6. Пример фрагмента логической модели в варианте IE

10.2.1. Логическая модель

Как уже упоминалось раньше, модель базы данных, проектируемая в ERwin, может быть логической и физической.

- Логическая модель отображает объекты точно так же, как они выглядят в реальном мире. Модель строится из сущностей, атрибутов и связей (рис. 10.7).

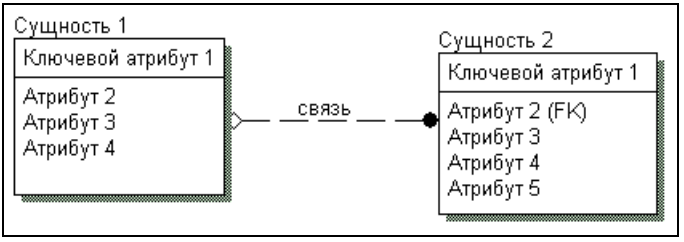


Рис. 10.7. Фрагмент логической модели (вариант IDEF1X)

- Сущность — это объект реального мира, например, "Здание", "Журнал регистрации", "Персона" и т. д. Сущность в физической модели превращается в таблицу. Сущность обладает следующими основными признаками:
  - имеет имя;

- она представляет класс, а не единичный экземпляр, например, не может быть сущности "сливочное мороженное" или "мороженное пломбир" — это все единичные экземпляры. Сущностью в данном случае будет либо просто "Мороженное", либо "Продукт питания", все зависит от того, какую базу данных мы разрабатываем.

Сущность может быть зависимой или независимой:

- ❒ независимая сущность не нуждается в информации из других сущностей, ее первичный ключ не включает в себя первичные ключи других сущностей. В ERwin независимая сущность отображается в виде прямоугольника;
- ❒ зависимая сущность ссылается на информацию из другой (других) сущности, ее первичный ключ включает в себя первичный ключ одной или нескольких родительских сущностей. В ERwin она отображается в виде прямоугольника с закругленными углами.

Чтобы лучше понять, в чем разница между зависимой и независимой сущностью, рассмотрим небольшой пример. Предположим, что у нас есть две сущности: "Дом" и "Квартира".

- ❒ В сущности "Дом" содержится следующая информация: номер дома, улица, дата постройки дома.
- ❒ В сущности "Квартира" содержится следующая информация: номер квартиры, площадь квартиры.

Каждому дому принадлежит какое-то количество квартир. Для того чтобы однозначно идентифицировать информацию в сущности "Дом", можно ввести атрибут "Код дома", тогда, для того чтобы однозначно идентифицировать информацию в сущности "Квартира", необходимо будет перенести атрибут "Код дома" в сущность "Квартира". Таким образом, получится, что связка атрибутов "Код дома" и "Номер квартиры" будут однозначно идентифицировать абсолютно всю информацию в сущности "Квартира". Получится, что сущность "Дом" — независимая, а сущность "Квартира" — зависимая, так как ее первичный ключ зависит от другой сущности (рис. 10.8).

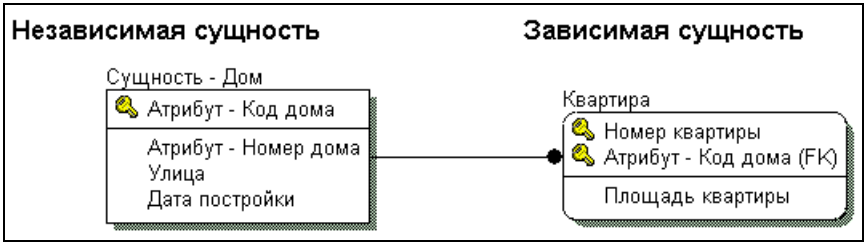


Рис. 10.8. Независимая и зависимая сущности

Замечание

В ERwin сущность (таблица) состоит из двух частей: верхняя — это область первичного ключа, нижняя предназначена для остальных атрибутов (полей).

Как уже известно, сущность состоит из атрибутов. *Атрибут*, как правило, характеризует конкретные факты или отражает свойства объекта или сущности, которой он принадлежит. Например, атрибут с названием "Номер дома" определяет те возможные значения, которые может принимать свойство физического объекта (дома), характеризующее числовую нумерацию строения. Как следует из названия атрибута "Номер дома", он может содержать только числа, он не предназначен для хранения имен людей или марок автомобилей.

Атрибут характеризуется следующими свойствами:

- имя;
- область определения — задается посредством типа данных атрибута.

Атрибут может либо входить в состав первичного ключа, либо нет.

10.2.2. Физическая модель

Физическая модель предназначена для реализации базы данных под конкретную СУБД, соответственно, обладает всеми теми ограничениями, которые накладывает конкретная СУБД.

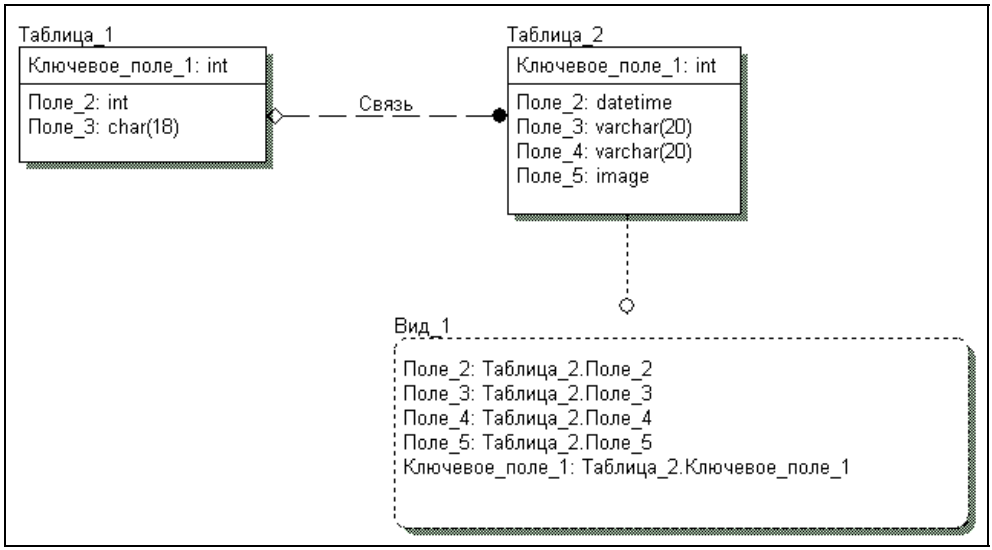


Рис. 10.9. Фрагмент физической модели (вариант IDEF1X)

Для одной логической модели может быть несколько физических. Физическая модель строится из таблиц, полей, связей и видов (рис. 10.9).

- Таблица — это сущность, перешедшая в физическую модель.
- Поле — это атрибут, перешедший в физическую модель.

10.3. Установка ERwin

Замечание

В данной главе рассматривается установка ERwin версии 4.0.

Запускаем файл установки ERwin, после этого начнется загрузка мастера установки (рис. 10.10).

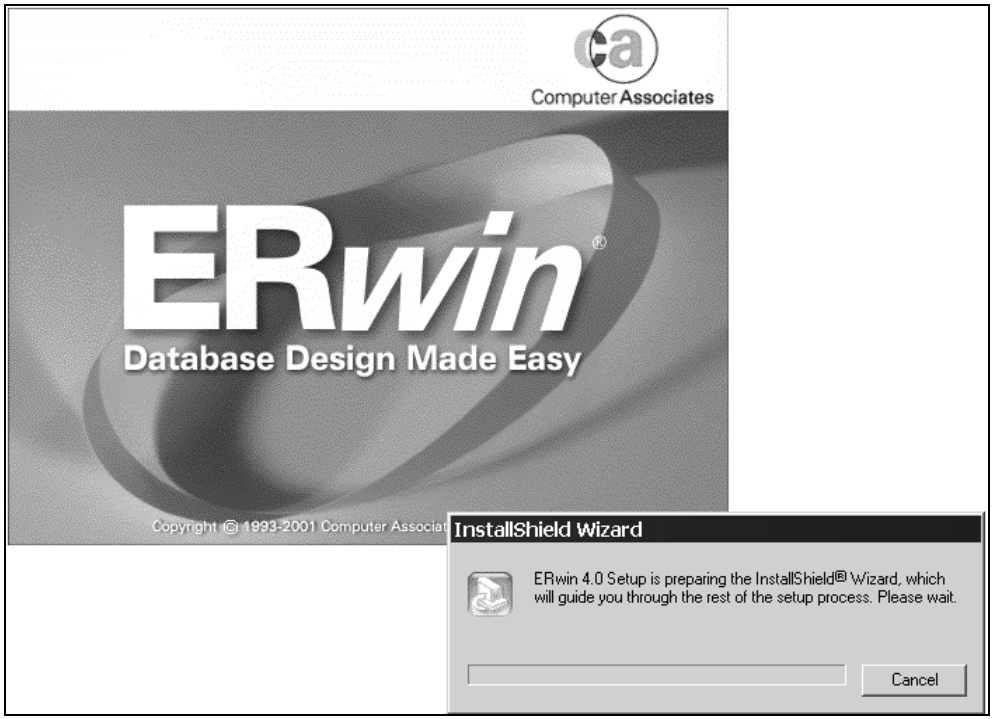


Рис. 10.10. Загрузка мастера установки ERwin

После того как мастер загрузится, появится окно приветствия. Необходимо нажать несколько раз кнопку **Next**, пока не начнется процесс установки системы. После этого можно приступать к работе с системой.

Замечание

Для того чтобы ERwin нормально отображал русские буквы в Windows XP, необходимо в разделе реестра HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Nls\CodePage для параметра "1252" прописать значение "c_1251.nls".

10.4. Анализ предметной области

На этом этапе собирается максимальный объем информации, относящейся к предметной области, для которой планируется разработать базу данных. Информацию можно получить либо посредством изучения документов, характеризующих предметную область, либо при помощи интервью с заказчиком, заинтересованным в разработке.

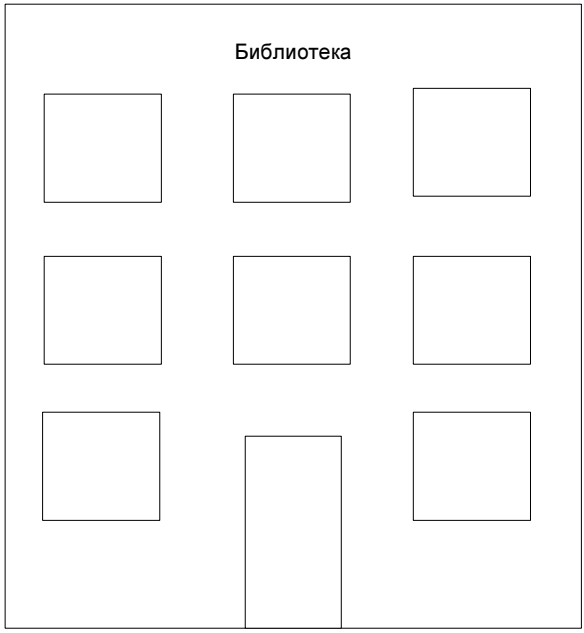


Рис. 10.11. Библиотека

В данной главе попробуем разработать полностью от начала до конца базу данных Библиотекарь. Мы разрабатывали похожее приложение в *главе 5*. В данной главе мы построим базу данных с помощью ERwin и после этого перенесем ее в среду СУБД MS SQL Server 2000.

Перед нами стоит задача разработать базу данных для библиотеки, которая обладает сетью филиалов. Соответственно, основная ее задача — это обслу-

живание читателей посредством предоставления им информации, содержащейся в книгах, которые и хранятся в библиотеке.

Для того чтобы построить модель будущей базы, необходимо произвести анализ, в результате которого мы определим объекты-сущности, информацию о которых нам необходимо будет хранить, а также определим, как взаимодействуют объекты между собой.

Как мы уже выяснили выше, основное направление организации — предоставление населению книг. Книги хранятся в библиотеке, которая в свою очередь располагается в каком-то здании. Соответственно библиотека может состоять из нескольких филиалов — отдельных зданий.

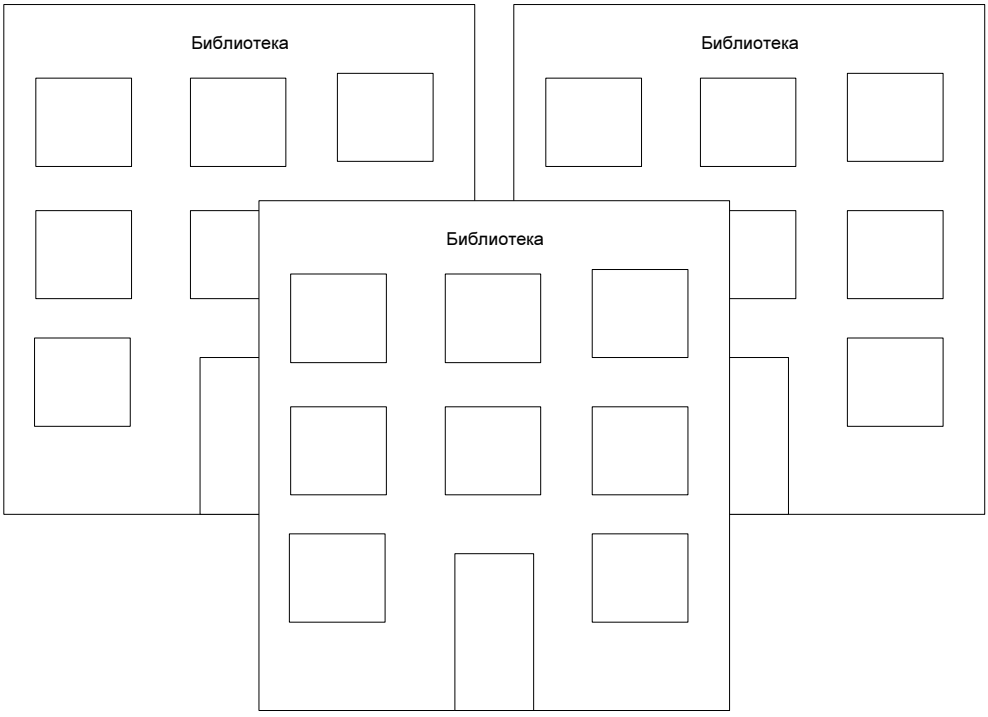


Рис. 10.12. Библиотека состоит из филиалов

Здание может быть главным офисом библиотеки, а может быть филиалом, соответственно, появляется такое свойство, как статус (рис. 10.13).

Замечание

Также здание может быть куплено, но не отремонтировано, тогда свойство "статус" примет значение "в ремонте".

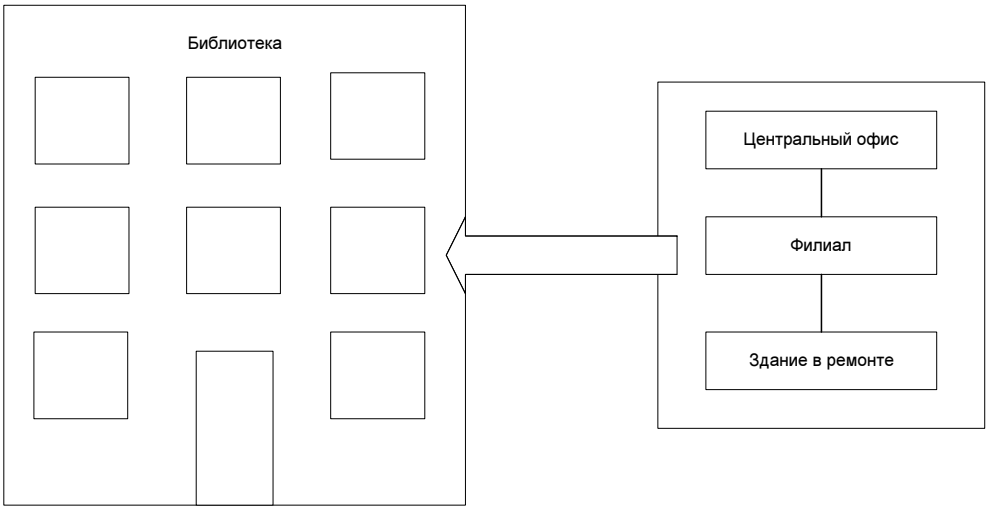


Рис. 10.13. У здания есть статус

Поскольку здания могут быть разбросаны географически (находиться в разных городах, странах, на разных улицах), то, следовательно, у них есть адрес расположения (рис. 10.14).

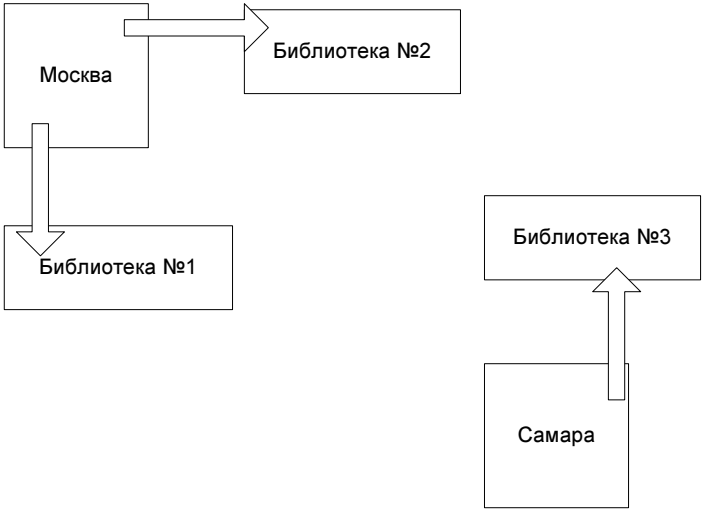


Рис. 10.14. Здания могут быть разбросаны географически

К тому же в здании может быть телефон, конечно номеров может быть несколько, но всегда есть один номер, который указывается в рекламе, гово-

рится клиентам. Номер, позвонив на который можно получить справку (рис. 10.15).

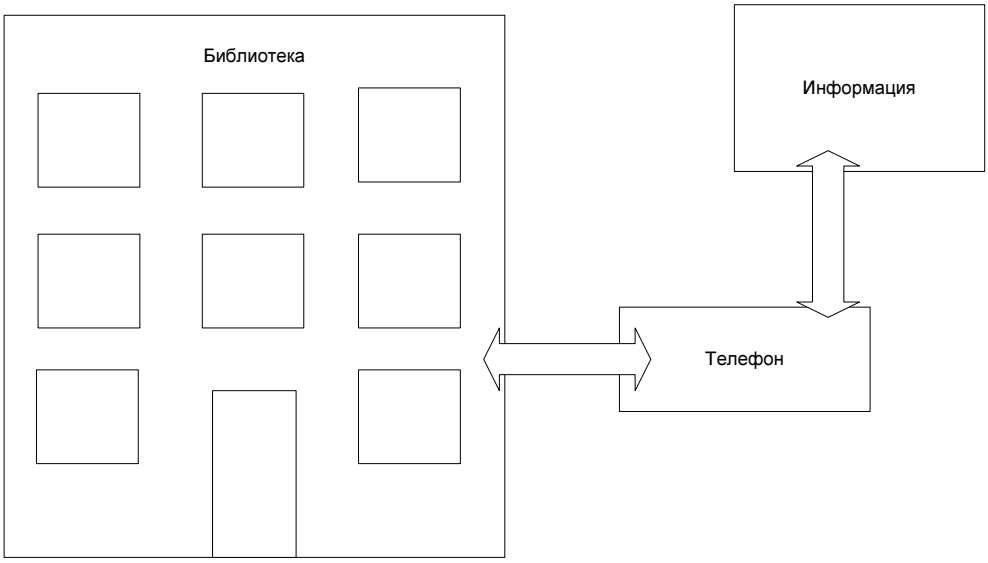


Рис. 10.15. У каждого здания, как правило, есть телефон

Таким образом, объект библиотека, как здание, характеризуется следующими параметрами:

- Статус — центральная, филиал, в ремонте и т. д.;
- Адрес — например: "445010 Самарская обл, г. Тольятти, улица Карла-Маркса, д. 9";
- Номер телефона — например: (8482)40-57-38.

Продолжаем анализировать. Каждая библиотека состоит из различных залов, например, читальный, зал выдачи книг, зал выдачи абонементов, зал по предоставлению коммерческих услуг и т. д. (рис. 10.16).

У каждого зала есть свой номер для того, чтобы его было легче найти читателю и чтобы указывать его непосредственно в документах внутренней отчетности библиотеки. Например, сотрудник библиотеки сможет сказать: "Вам необходимо пройти в зал номер 5 — Читальный". Или пример текста в отчете: "В зале номер 3 — Коммерческом — произошло короткое замыкание". Также у каждого зала есть свой график работы, конечно, он может быть у всех залов одинаковый, но это не обязательно. Конечно же, в каждом зале может быть отдельный городской номер и также может быть внутренний номер для общения сотрудников в пределах одного здания между собой.

Рис. 10.16. Каждое здание библиотеки состоит из залов

Таким образом, объект-зал, как составная часть здания, обладает следующими характеристиками:

- Номер — 1, 2, 15, 75;
- Название — Читальный, Зал коммерческих услуг;
- График работы;
- Номер телефона;
- Внутренний номер телефона.

В каждом зале работают сотрудники. У каждого сотрудника есть имя, а также документы необходимые для его трудоустройства, такие как паспорт, ИНН, диплом и др. Объект-сотрудник, который работает в определенном зале определенного здания, будет обладать следующими параметрами:

- Имя;
- Должность;
- Состоит ли он в браке;
- Дата приема на работу;
- Адрес проживания;
- Данные паспорта;

- Данные диплома;
- Специальность по диплому;
- Номер ИНН;
- Номер пенсионного свидетельства.

Книга, как основное средство предоставления информации читателям, имеет название и жанр, который характеризует информацию, содержащуюся в книге. Также у книги обязательно есть автор. Так как одна и та же книга может находиться в библиотеке в нескольких экземплярах, то, соответственно, появляются дополнительные свойства: количество книг в библиотеке и количество в наличии (рис. 10.17).

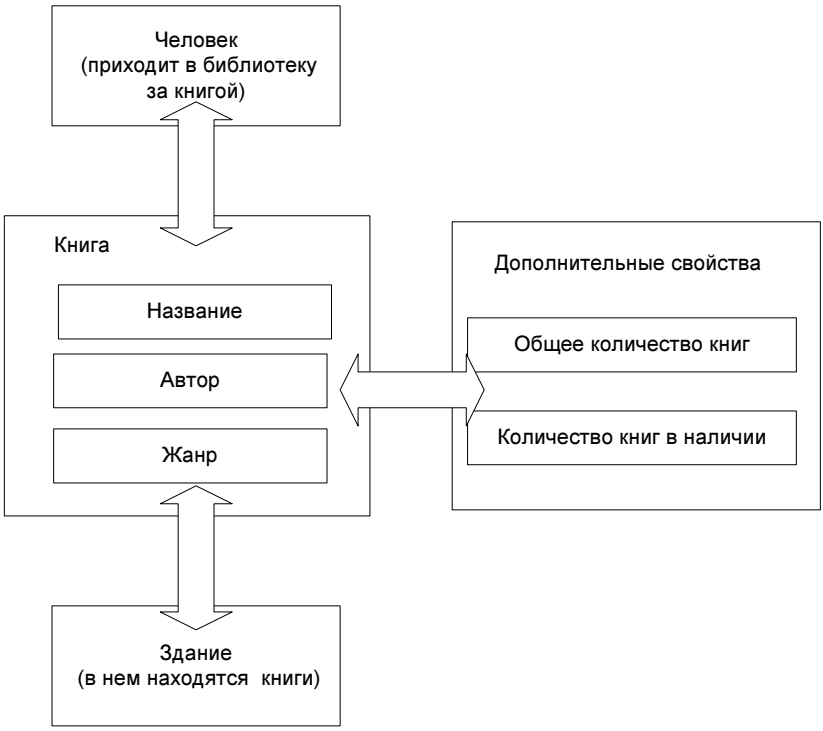


Рис. 10.17. У книги есть свойства (название, жанр, автор), она хранится в здании библиотеки и именно за ней приходит читатель

Объект-книга будет обладать следующими параметрами:

- Название;
- Автор;
- Жанр.

Плюс двумя дополнительными:

- Общее количество книг;
- Количество книг в наличии.

Библиотека работает для своих читателей. Так что читатель — один из ключевых объектов. У читателя есть имя, он проживает по какому-то адресу, так же у него есть паспорт и, соответственно, он либо работает, либо учится, либо совсем ничего не делает. Объект-читатель будет обладать следующими свойствами:

- Имя;
- Адрес;
- Данные паспорта;
- Место работы или учебы;
- Дата регистрации читателя.

Каждая книга выдается при условии наличия у читателя абонемента. У абонемента есть владелец в лице читателя. Поэтому объект-абонемент является посредником между читателем и книгой (рис. 10.18).

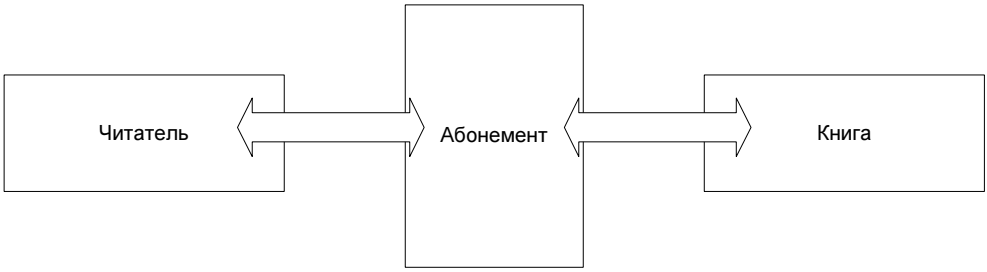


Рис. 10.18. Абонемент является посредником между читателем и книгой

Объект-абонемент обладает следующими характеристиками:

- Номер, который является уникальным в пределах каждого отдельного филиала;
- Филиал, который выдал абонемент;
- Владелец абонемента;
- Дата выдачи;
- Дата окончания;

- Признак недействительности абонемента — указывает, может ли по данному абонементу читатель получать книги и услуги, предоставляемые библиотекой.

Как мы уже выяснили, основное назначение библиотеки — предоставление информации читателю посредством книг. Но в библиотеке может быть развитая система предоставляемых услуг, например, оказание помощи в поиске книги, ксерокопирование и т. д., поэтому каждый отдел характеризуется набором услуг, которые он может предоставить читателю. Услуга, в свою очередь, может быть бесплатной или платной. Если услуга платная, то у нее есть цена. Но основной услугой все же является выдача книги читателю.

Объект-услуга обладает следующими параметрами:

- Название;
- Тип услуги (платная/бесплатная);
- Цена (если услуга платная).

Все услуги оказываются кому-то и кем-то, следовательно, эта информация где-то должна регистрироваться. Предлагаю назвать объект, который будет нести эту функцию, "Журналом регистрации услуги". Например, при выдаче книги в журнале регистрации будет указано: кто выдал книгу, на какой абонемент и когда была выдана книга. Таким образом, абонемент является не только посредником между читателем и книгой, но и между читателем и услугой. Еще один важный параметр — это учет: была ли услуга кратковременной, например, как ксерокопирование, или это долговременная услуга, например, выдача книги, то есть после регистрации действия время пользования услугой может быть длительным. Соответственно, если книга не возвращена в библиотеку, то мы должны каким-то образом отслеживать это. Введем параметр "Признак переноса записи в архив". Если параметр установлен, то действие услуги было закончено, и она предоставляет интерес, как архивная информация.

Объект "Журнал регистрации услуги" обладает следующими параметрами:

- Сотрудник, который оказал услугу;
- Читатель, которому была оказана услуга;
- Какая услуга была оказана;
- Если услуга — выдача книги, то необходимо указать, какая книга выдана;
- Дата оказания услуги;
- Цена, которую заплатил читатель (прейскурант цен может и меняться, а в журнале регистрации будет указываться цена, которая была актуальна на день оказания услуги);
- Признак переноса записи об оказании услуги в архив.

Попробуем привести весь аналитический отчет к одной схеме (рис. 10.19).

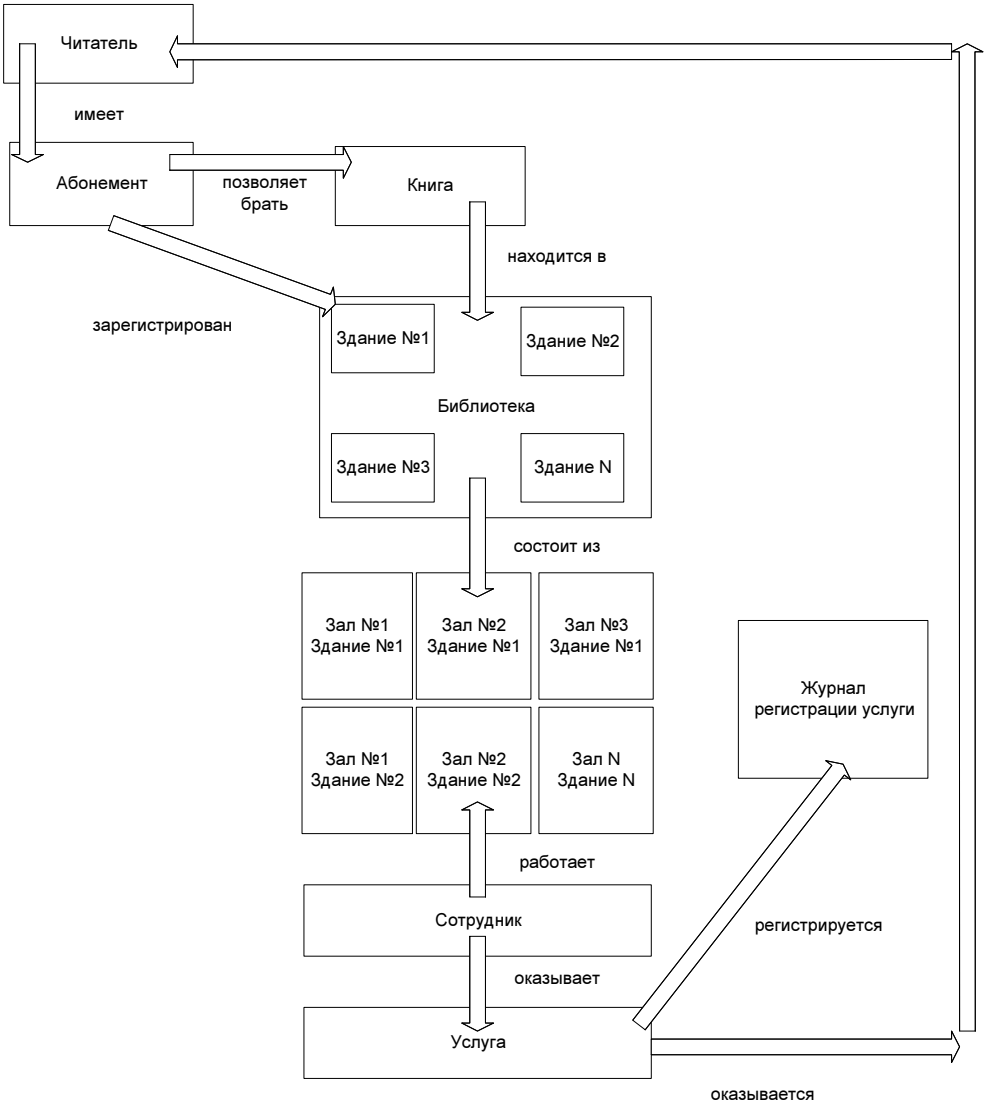


Рис. 10.19. Схема, характеризующая отношения между различными объектами в системе Библиотека

10.5. От объектов к сущностям

После того как анализ проведен, можно прейти к определению сущностей. Сразу поделим их на группы.

- Группа 1: "Здания и комнаты" (в нее входят следующие сущности):
  - Здание
  - Зал
- Группа 2: "Услуги и Документы" (то, за чем читатель приходит в библиотеку, и посредством чего он получает это):
  - Абонемент
  - Книга
  - Автор
  - Жанр
  - Услуга
  - Журнал регистрации услуги
- Группа 3: "Сотрудники и читатели":
  - Сотрудник
  - Читатель

Теперь определим более подробно каждую сущность и определим тип хранимых ею данных.

В табл. 10.1 и 10.2 рассмотрим сущности группы "Здания и комнаты".

Таблица 10.1. *Сущность "Здание" (Building)*

№	Имя атрибута	Тип данных	Ключ
1	Код здания	Число	*
2	Статус	Строка	
3	Полный адрес	Строка(100)	
4	Телефон	Строка(15)	
5	Примечание	Строка(200)	

Замечание

Как мы уже выяснили из проведенного выше анализа, библиотека может иметь развитую сеть филиалов. Как частный случай, она может состоять из одного филиала. Для обеспечения уникальности каждого филиала добавлен дополнительный атрибут "Код здания", который является первичным ключом.

Таблица 10.2. *Сущность "Зал" (Place)*

№	Атрибут	Тип данных	Ключ
1	Номер зала	Число	* — ключ будет составным и будет включать два атрибута
2	Код филиала	Число	
3	Название	Строка(100)	
4	Номер телефона (если есть)	Строка(15)	
5	Внутренний номер телефона (если есть)	Строка(10)	
6	Время начала работы	Время	
7	Время окончания работы	Время	
8	Примечание	Строка(200)	

В табл. 10.3—10.8 рассмотрим сущности группы "Услуги и документы".

Таблица 10.3. *Сущность "Абонемент" (Abonement)*

№	Атрибут	Тип данных	Ключ
1	Код абонемента	Число	*
2	Код филиала (который его выдал)	Число	
3	Код читателя (который владеет абонементом)	Число	
4	Имя читателя	Строка(100)	
5	Дата выдачи	Дата	
6	Дата окончания	Дата	
7	Признак недействительности абонемента	Логический	

Имя читателя является избыточной информацией, но зато облегчит нам работу с базой. Как вы, наверно, могли заметить, когда мы работали в *главе 5* с нашей базой данных, для того, чтобы, например, извлечь информацию о том, какие книги брал определенный читатель, нам приходилось собирать данные сразу из нескольких таблиц: Reader ("Читатель"), чтобы получить его имя; Move, чтобы отследить то, что брал читатель; а также Abonement ("Абонемент"), чтобы отследить, что книгу брал определенный читатель, у

которого есть определенный абонемент. В данном случае использование избыточности облегчит нам в дальнейшем построение запросов и получение информации.

Замечание

В данной сущности возможны два варианта первичного ключа, первый — это атрибут "Код абонемента", второй вариант — это атрибуты "Код филиала" и "Код читателя". Остановим свой выбор на атрибуте "Код абонемента", который будет обеспечивать уникальную нумерацию всех абонементов во всех филиалах библиотечной сети; таким образом, двух абонементов с одинаковыми номерами не будет. Когда в сущности (или в таблице) присутствует несколько возможных вариантов первичных ключей, неиспользуемый вариант называется *альтернативным ключом*.

Так как мы используем ключ "Код абонемента", то сущность "Абонемент" будет независимой. Если бы мы воспользовались ключом "Код филиала" — "Код читателя", то сущность стала зависимой, так как ключ включал бы атрибуты других сущностей.

Таблица 10.4. *Сущность "Книга" (Book)*

№	Атрибут	Тип данных	Ключ
1	Код книги	Число	*
2	Название	Строка(100)	
3	Код автора	Число	
4	Код жанра	Число	
6	Общее количество книг	Число	
7	Количество книг в наличии	Число	

Таблица 10.5. *Сущность "Автор" (Author)*

№	Атрибут	Тип данных	Ключ
1	Код автора	Число	*
2	Полное имя автора	Строка(100)	

Таблица 10.6. *Сущность "Жанр" (Ganr)*

№	Атрибут	Тип данных	Ключ
1	Код жанра	Число	*
2	Название жанра	Строка(100)	

Таблица 10.7. *Сущность "Услуга" (Service)*

№	Атрибут	Тип данных	Ключ
1	Код услуги	Число	*
2	Название услуги	Строка(50)	
3	Указатель платной услуги	Логический	
4	Цена услуги	Число	
5	Примечание	Строка(200)	

Таблица 10.8. *Сущность "Журнал регистрации услуги" (Service_kontrol)*

№	Атрибут	Тип данных	Ключ
1	Код записи в журнале	Число	*
2	Код услуги	Число	
3	Код сотрудника, который оказал услугу	Число	
4	Код абонемента, для владельца которого она была оказана	Число	
	Код книги, если услугой была выдача книги читателю	Число	
5	Дата оказания	Дата	
6	Сумма (оплаченная читателем по прайсу на момент оказания услуги)	Число	
7	Признак переноса записи в архив	Логический	

В табл. 10.9—10.10 рассмотрим сущности группы "Сотрудники и читатели".

Таблица 10.9. *Сущность "Сотрудник" (Worker)*

№	Атрибут	Тип	Ключ
1	Код сотрудника	Число	*
2	Код здания (где работает)	Число	
3	Номер зала (где работает)	Число	
4	Полное имя сотрудника	Строка(100)	
5	Должность	Строка(50)	
6	Брак (состоит ли он в нем)	Логический	

Таблица 10.9 *(окончание)*

№	Атрибут	Тип	Ключ
7	Дата приема на работу	Дата	
8	Адрес проживания	Строка(100)	
9	Паспорт	Строка(15)	
10	Диплом	Строка(15)	
11	Специальность	Строка(50)	
12	ИНН	Строка(15)	
13	Пенсионное свидетельство	Строка(15)	

Таблица 10.10. *Сущность "Читатель" (Reader)*

№	Атрибут	Тип	Ключ
1	Код читателя	Число	*
2	Полное имя читателя	Строка(100)	
3	Паспорт	Строка(15)	
4	Адрес проживания	Строка(100)	
5	Дата регистрации	Дата	

Вместо двух сущностей — "Читатель" и "Сотрудник" — можно использовать три: "Персона", "Читатель" и "Сотрудник". В сущности "Персона" хранится общая информация, касающаяся человека: имя, адрес. А в остальных сущностях специфическая информация, касающаяся той роли, которую представляет данный человек, например, если это читатель, то для него нет необходимости хранить его ИНН.

Используемый в этой главе вариант был выбран специально, как более простой. Если есть желание, модель всегда можно усложнить, добавив в нее дополнительные сущности.

10.6. Работа в ERwin

Запускаем ERwin. На экране появится окно, в котором можно открыть уже ранее созданную модель — переключатель **Open an existing file** — или создать новую — переключатель **Create a new model** (рис. 10.20).

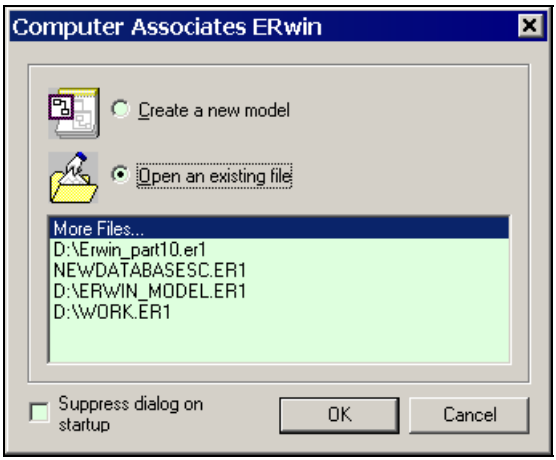


Рис. 10.20. Окно, которое появляется при запуске ERwin

Устанавливаем переключатель **Create a new model** и нажимаем **OK**.

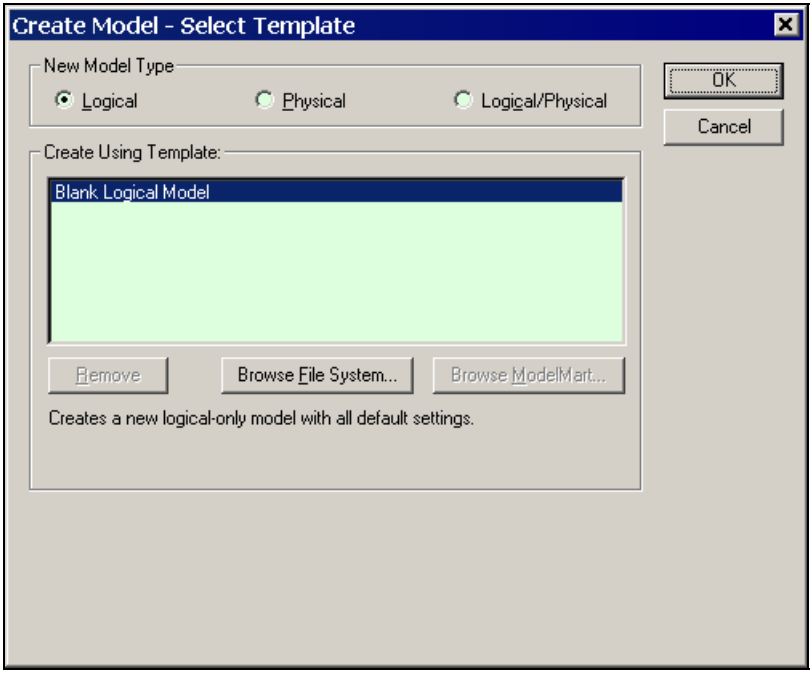


Рис. 10.21. Окно **Create Model - Select Template**

Замечание

Для создания новой модели можно также выбрать пункт **File\New** главного меню.

На экране появится окно **Create Model - Select Template** (Создание модели - Выберите шаблон) — рис. 10.21.

В группе элементов **New Model Type** доступны следующие переключатели:

- **Logical** — создание только логической модели;
- **Phisical** — создание только физической модели данных, при выборе этого переключателя станет доступной дополнительная группа элементов **Target Database**, в которой выбираются имя СУБД (выпадающий список **Database**) и ее версия (выпадающий список **Version**);
- **Logical/Phisical** — создание и логической, и физической модели в одном проекте, при установке этого переключателя также становится доступной группа элементов **Target Database** (рис. 10.22).

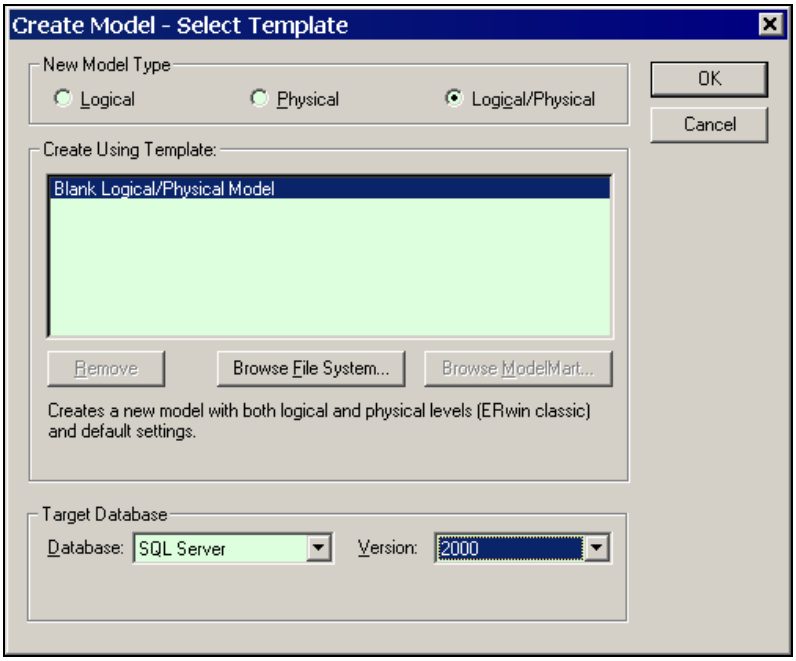


Рис. 10.22. Группа элементов **Target Database**

Устанавливаем переключатель **Logical/Phisical**, выбираем тип сервера **SQL Server**. В выпадающем списке **Version** выбираем **2000**. Нажимаем **OK**, после этого мы увидим окно, как показано на рис. 10.23.

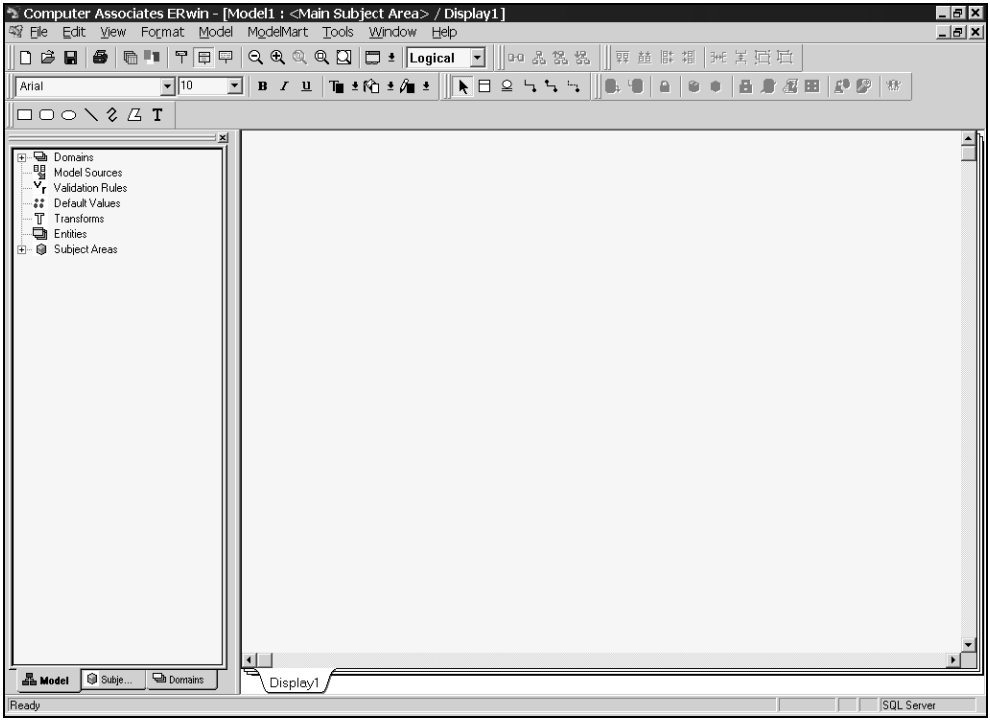


Рис. 10.23. Основное окно ERwin

Это рабочее пространство ERwin, в котором мы и будем создавать модель будущей базы данных. Рассмотрим подробнее, из каких элементов оно состоит. Сверху расположено главное меню (рис. 10.24).

Рис. 10.24. Главное меню ERwin

Под главным меню расположена панель инструментов (рис. 10.25).

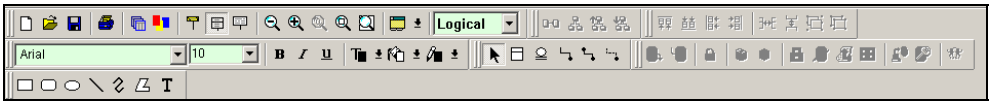


Рис. 10.25. Панель инструментов ERwin

Замечание

Панели, расположенные в панели инструментов, также как и во многих других программах, можно перетаскивать на рабочее пространство ERwin. Видимость различных панелей настраивается с помощью пункта **View\Toolbard** главного меню и установкой галочки напротив названия той панели, видимость которой вы хотите включить (рис. 10.26).

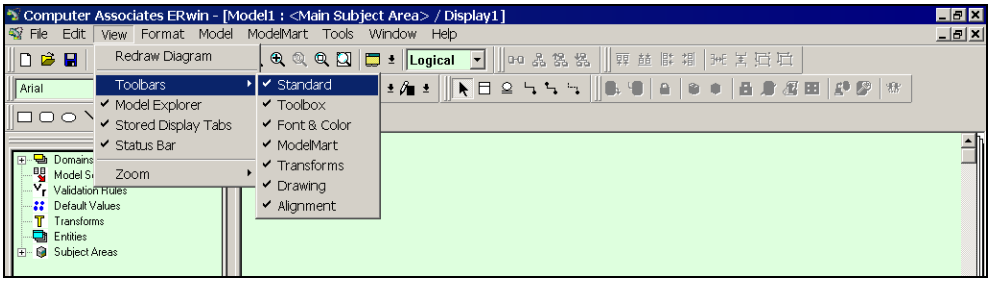


Рис. 10.26. Настройка видимости различных панелей

В панели **Standard** есть выпадающий список, который позволяет переключаться между логической моделью данных (**Logical**) и физической (**Physical**) (рис. 10.27.

Рис. 10.27. Панель **Standard**

В левой части рабочего пространства ERwin расположен инспектор модели, который содержит все свойства модели. У него есть три вкладки, расположенные внизу (рис. 10.28):

- вкладка **Model** — отображает все элементы модели;
- вкладка **Subject** — предметные области (об этом поговорим чуть позже);
- вкладка **Domains** — домены (об этом поговорим чуть позже).

В ERwin присутствует такой элемент, как **Stored Display** (Хранимые отображения), о нем мы поговорим немного позже, сейчас нас интересует вкладка, расположенная в нижней части рабочего пространства ERwin. Пока она у нас одна под названием **Display1**, но в дальнейшем мы это исправим (рис. 10.29).

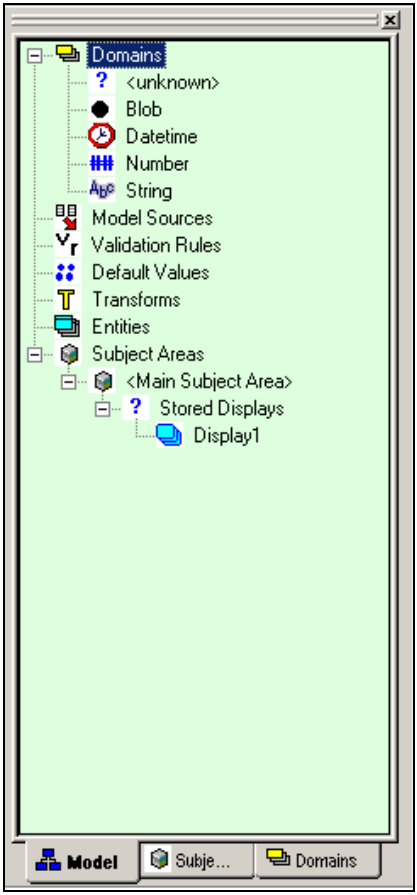


Рис. 10.28. Инспектор модели

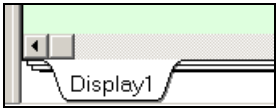


Рис. 10.29. Вкладка **Display1**

Замечание

В ERwin встроена замечательная справка, которая вызывается выбором пункта меню **Help\Tutorial** и состоит из 18 уроков, в каждом из них очень подробно и наглядно рассмотрена работа с программой. При наличии элементарных знаний английского языка можно воспользоваться помощью этого электронного руководства (мое знакомство с ERwin началось именно с него) — рис. 10.30.

Рис. 10.30. Электронное руководство по ERwin

Продолжаем работать с ERwin. Выбираем пункт меню **Model\Model Properties**, на экране появится окно **Model Properties**, предназначенное для настройки свойств модели, пока нас интересуют 2 поля:

- поле **Name** — имя модели, задаем DataBase Library;
- поле **Author** — имя автора, в которое необходимо ввести свои данные.

У меня окно **Model Properties** после ввода данных выглядит как на рис. 10.31.

Если в окне **Model Properties** сделать активной вкладку **History**, то можно будет посмотреть историю изменений модели (рис. 10.32).

Нажимаем **OK** для того, чтобы закрыть окно.

Для построения модели базы данных в ERwin существует специальная панель **ToolBox**. Как уже известно, ERwin позволяет разрабатывать два вида моделей: Логическую (**Logical**) и Физическую (**Physical**). Так вот, в зависимости от выбранного варианта построения модели (IDEFIX или IE) панель **ToolBox** будет иметь различные элементы для построения одних и тех же элементов (см. рис. 10.1—10.6).

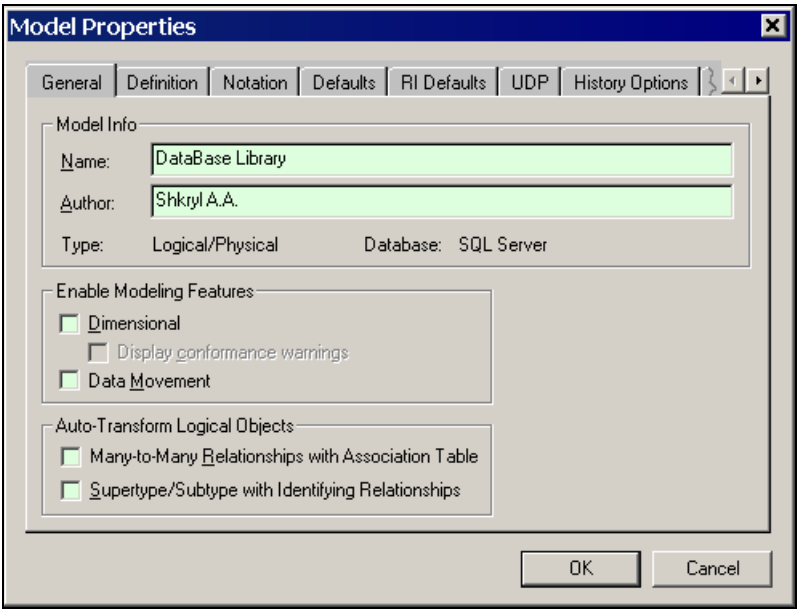


Рис. 10.31. Окно **Model Properties**

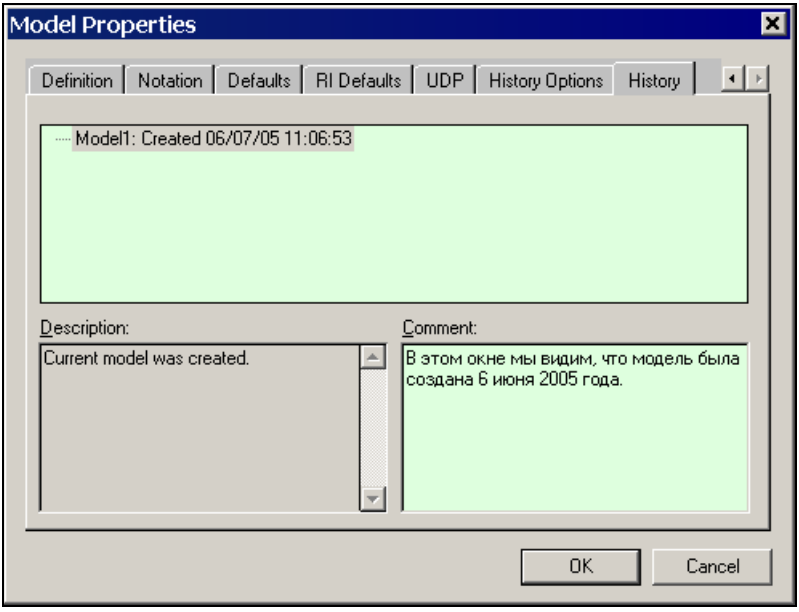


Рис. 10.32. Просмотр истории изменений

Замечание

Для того чтобы изменить вариант построения модели, необходимо выбрать пункт **Model\Model Properties** главного меню и после этого сделать активной вкладку **Notation**. Здесь мы можем указывать вариант построения логической и физической модели, причем у физической модели доступен дополнительный вариант **DM** (Dimensional Modeling — моделирование с помощью измерений). В этой главе мы будем использовать вариант построения модели **IDEF1X** (рис. 10.33).

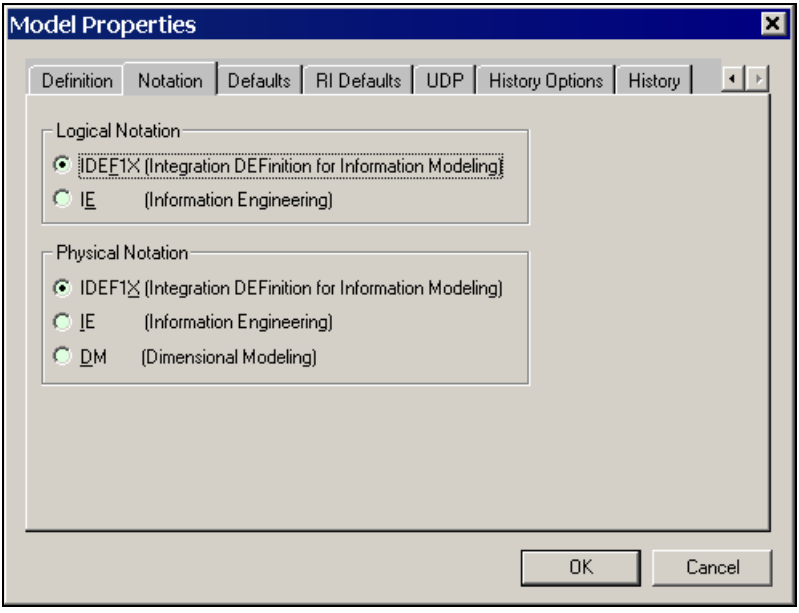


Рис. 10.33. Выбор варианта построения модели в ERwin

Приступим к процессу рисования будущей базы данных. В первую очередь мы построим логическую модель, в которой используются сущности, атрибуты и связи.

В панели инструментов щелкаем левой кнопкой мыши на объекте **Entity** (пиктограмма с изображением прямоугольника, поделенного внутри линией на две неравные части). После этого производим щелчок левой кнопкой мыши в любом месте рабочего пространства ERwin (рис. 10.34).

По умолчанию система дает имя сущности самостоятельно. Для того чтобы изменить его, необходимо просто ввести его с клавиатуры. Вводим "Здание" и нажимаем <Enter>.

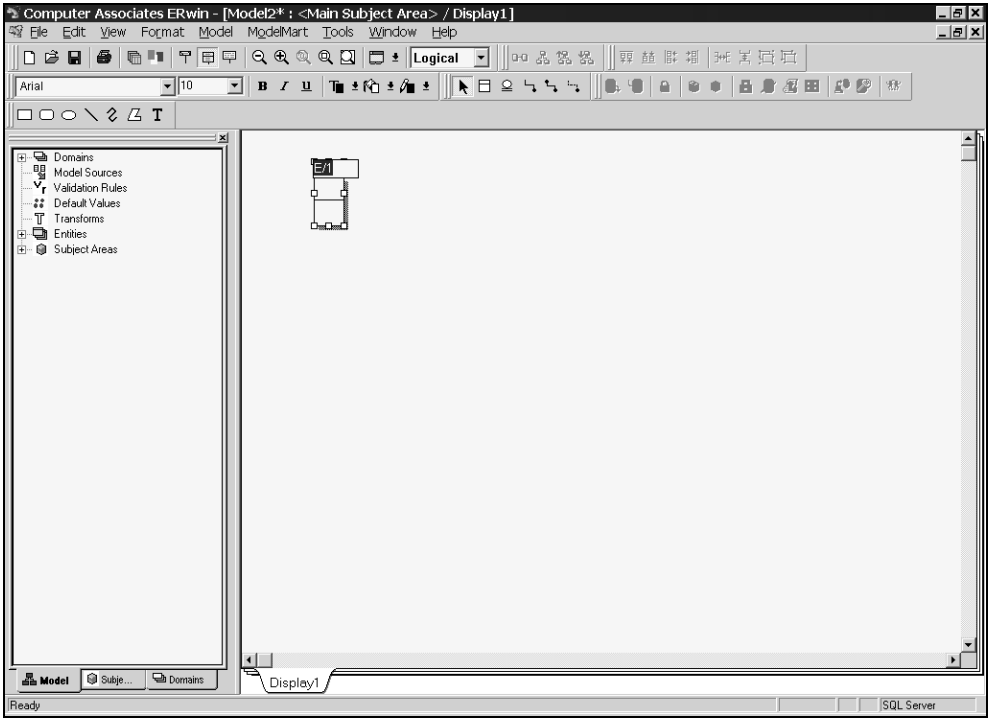


Рис. 10.34. Создание сущности

Если мы по ошибке разместили лишнюю сущность, то ее можно удалить, для этого необходимо выделить ее (это достигается нажатием левой кнопкой на изображении сущности) и нажать кнопку <Delete>, на экране появится окно **Delete Selected Objects**, в котором будет задан вопрос: "Удалить выделенный объект?" Если мы хотим сделать это, то просто нажимаем **OK** (рис. 10.35).

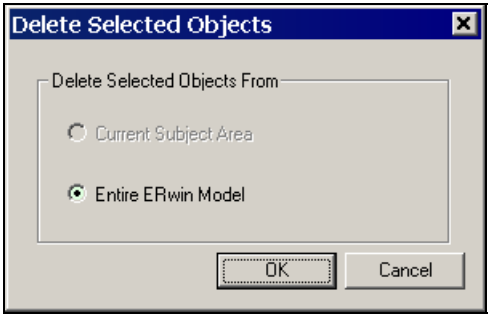


Рис. 10.35. Окно подтверждения операции удаления выбранного объекта

Замечание

Точно таким же образом можно удалять все остальные элементы модели.

После того как сущность создана, можно обратиться к ее свойствам. Для этого производим щелчок правой кнопкой мыши на ней и в выпадающем меню выбираем пункт **Entity Properties** (рис. 10.36).

Замечание

При открытии свойств элементов модели, таких как сущности, атрибуты, таблицы и т. д., как правило, появляются однотипные окна, в которых находится ряд вкладок, поэтому мы подробно рассмотрим свойства сущности, а обзор свойств остальных элементов будет более кратким.

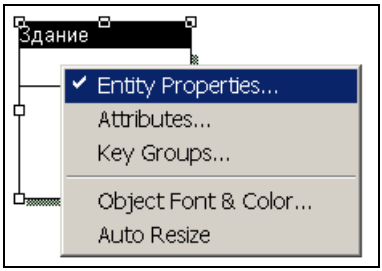


Рис. 10.36. Пункт **Entity Properties**

На экране появится окно **Entities** (рис. 10.37).

В выпадающем списке **Entity** выбирается сущность, свойство которой мы хотим изменять. В поле **Name** можно изменять имя сущности. Вкладка **Definition** позволяет задать описание сущности. Предлагаю сделать это в соответствии с рис. 10.38.

Замечание

Вкладка **Definition** присутствует в свойствах всех элементов модели базы данных, это объясняется тем, что если кто-то другой, или даже вы, многое время спустя, решите обратиться к своей модели, тогда можно будет быстро понять что к чему.

Вкладка **Icon** предназначена для задания иконки сущности, при выборе этой вкладки становятся доступными дополнительные элементы:

- выпадающий список **Small Icon** — задание иконки для сущности, для ее загрузки необходимо нажать кнопку с тремя точками, расположенную напротив выпадающего списка;

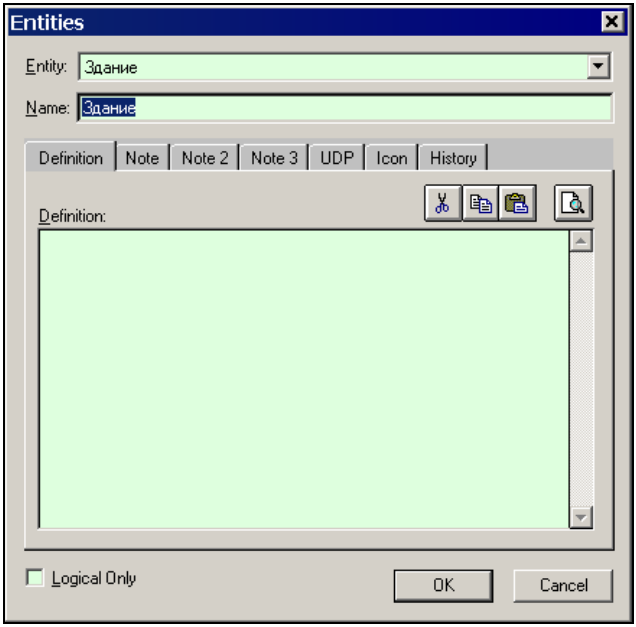


Рис. 10.37. Окно **Entities**

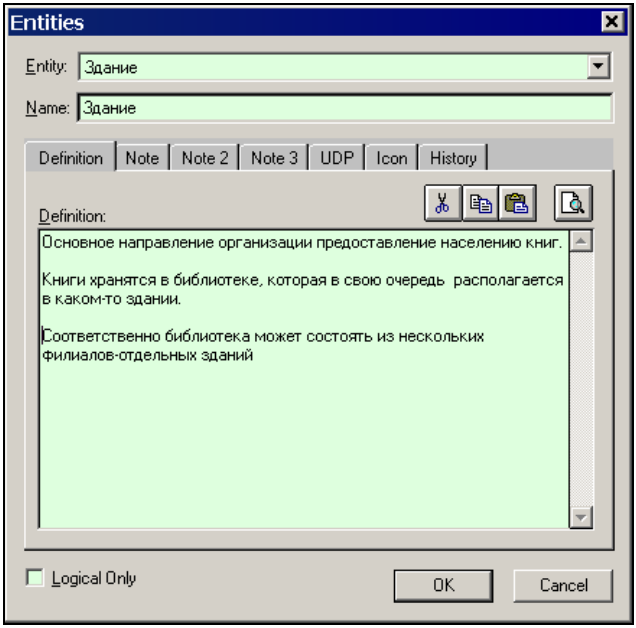


Рис. 10.38. Описание сущности

- выпадающий список **Large Icon** — задание картинки для сущности, для ее загрузки необходимо нажать кнопку с тремя точками, расположенную напротив выпадающего списка.

> **Замечание**
>
> Что делать, если у вас нет под рукой коллекции иконок? Можно воспользоваться иконками, поставляемыми с ERwin (если путь при установке был оставлен по умолчанию, то их можно найти по адресу C:\Program Files\Computer Associates\ERwin 4.0\Icons). Иконки для сущностей должны быть в формате BMP.

Я в своей модели в поле **Small Icon** загрузил файл Library.BMP из коллекции ERwin, а в поле **Large Icon** файл ПАРКЕТ.BMP, поставляемый вместе с Windows; предлагаю последовать моему примеру или использовать другие картинки (рис. 10.39).

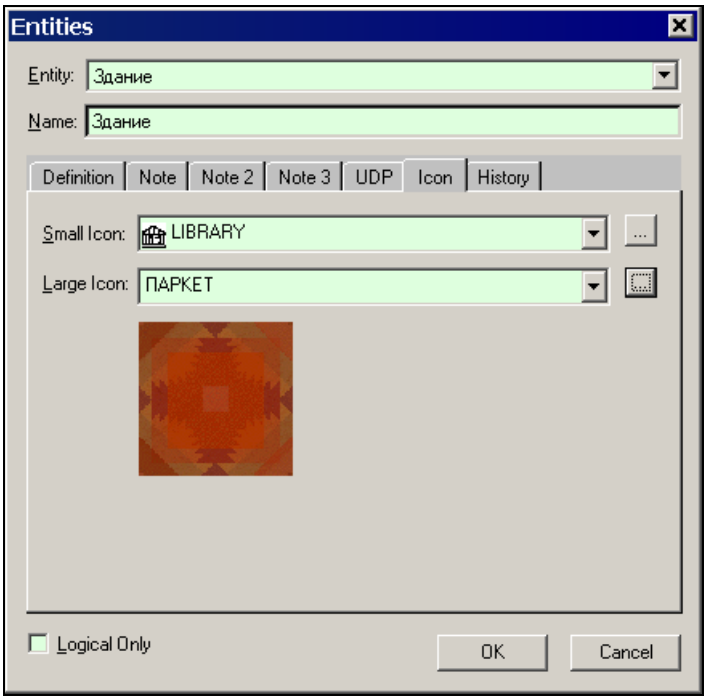


Рис. 10.39. Присвоение иконок сущности

Нажимаем **OK**. Теперь можно изменить внешний вид отображения модели, чтобы увидеть иконки. Щелкаем правой кнопкой мыши на рабочем пространстве и в выпадающем меню выбираем пункт **Display Level\Icon** (рис. 10.40).

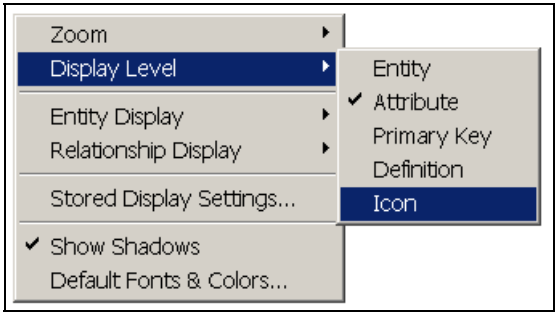


Рис. 10.40. Пункт **Display Level\Icon**

Замечание

Display Level — задает вид отображения сущностей, после выбора этого пункта раскроется ряд дополнительных пунктов. По умолчанию галочка установлена на пункте **Attribute** — это соответствует тому, что будут отображаться сущности и их атрибуты. Если выбрать пункт **Icon**, то будет отображена картинка, которую мы загружали посредством поля **Large Icon**; если выбрать пункт **Definition**, то будет выведена сущность с ее описанием внутри (рис. 10.41 и 10.42).

Щелкаем правой кнопкой мыши на рабочем пространстве и в выпадающем меню выбираем пункт **Display Level\Attribute** — вернемся, так сказать, к первоначальному виду.

Для того чтобы увидеть иконку, необходимо щелкнуть правой кнопкой мыши на рабочем пространстве и в выпадающем меню выбрать пункт **Entity Display\Entity Icon** (рис. 10.43).

После этого сущность будет иметь следующий вид — рис. 10.44.

Рис. 10.41. Вид сущности при выборе пункта **Display Level\Icon** в выпадающем меню

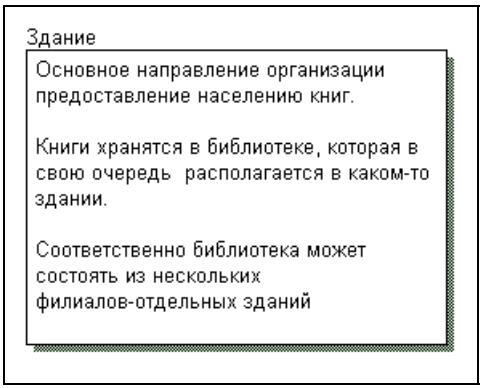


Рис. 10.42. Вид сущности при выборе пункта **Display Level\Definition** в выпадающем меню

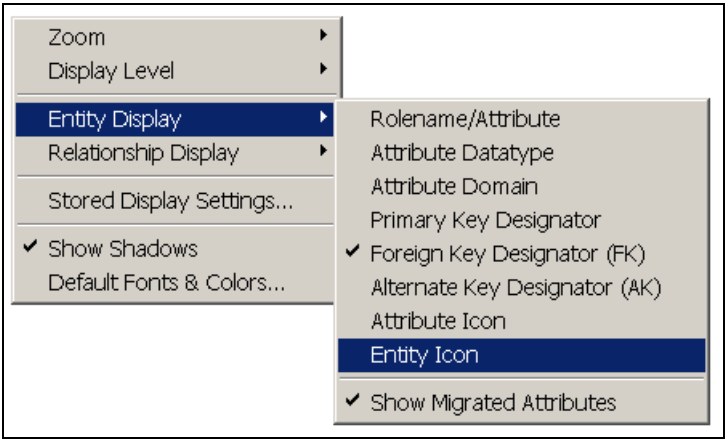


Рис. 10.43. Пункт **Entity Display\Entity Icon**

Рис. 10.44. Вид сущности при выборе пункта **Entity Displayl\Entity Icon** в выпадающем меню

Замечание

Предлагаю, когда мы создадим больше сущностей, вернуться к пунктам **Entity Display** и **Display Level** и поэкспериментировать с различными вариантами отображения для того, чтобы лучше понять их назначение.

Добавим в сущность "Здание" атрибуты. Эту операцию можно произвести несколькими способами.

10.6.1. Первый способ создания атрибута — прямое добавление

Производим щелчок правой кнопкой мыши на сущности "Здание" и в выпадающем меню выбираем пункт **Attributes**, на экране появится окно **Attributes** (рис. 10.45).

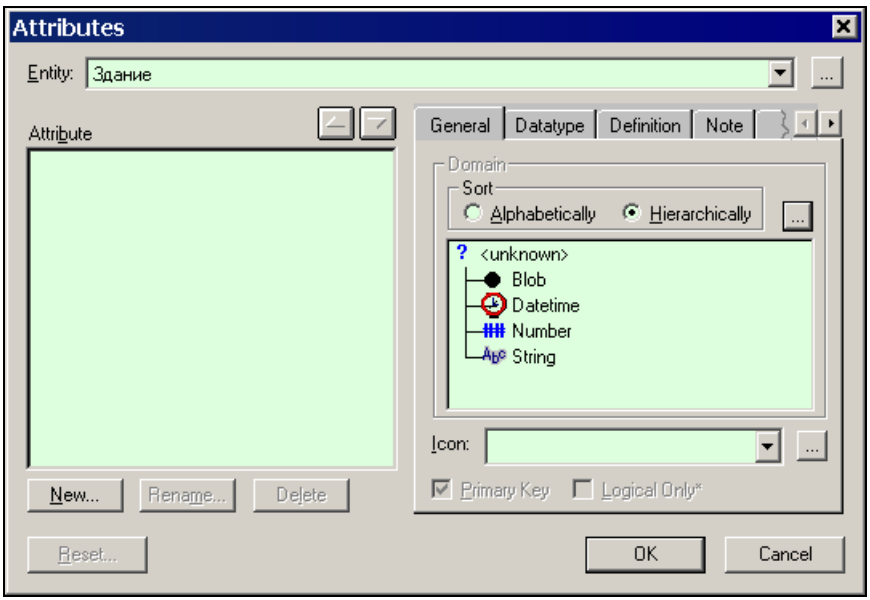


Рис. 10.45. Окно **Attributes**

В выпадающем списке **Entity**, расположенном в верхней части окна, выбирается сущность, с атрибутами которой мы будем работать. По умолчанию там установлена та, на которой мы производили щелчок.

Нас интересуют кнопки **New** — создать атрибут, **Rename** — переименовать атрибут и **Delete** — удалить атрибут. Нажимаем **New**, появится окно **New Attribute** (рис. 10.46).

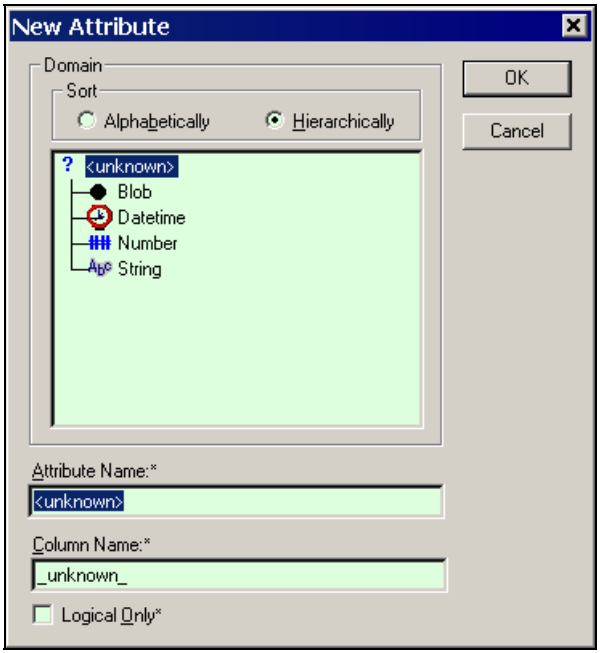


Рис. 10.46. Окно **New Attribute**, предназначенное для создания нового атрибута

Сначала нам необходимо выбрать тип данных будущего атрибута в группе элементов **Domain**. В ERwin присутствуют следующие типы данных:

- Blob — двоичные данные;
- DateTime — дата\время (два в одном);
- Number — числовой тип;
- String — строка.

Это базовые типы данных (или базовые домены — Domain), которые происходят от домена <unknown>.

Замечание

Все атрибуты или типы данных (домены), созданные разработчиком, происходят от базовых типов данных.

Выбираем домен Number. В поле **Attribute Name** задается имя атрибута, именно оно будет отображаться в сущности. В поле **Column Name** задается имя поля в таблице, когда работа с моделью будет осуществляться на физическом уровне.

Замечание

Имя атрибута может быть русским, имя поля — нет, так как именно оно будет передаваться в СУБД, когда мы будем создавать базу данных, на основе разработанной модели. А, как известно, большинство СУБД русских символов в названии своих элементов не поддерживают.

Задаем следующие значения:

- поле **Attribute Name** — Код здания;
- поле **Column Name** — building_id.

Нажимаем **OK**. После этого можно будет увидеть, что в сущности "Здание" появился атрибут "Код здания" (рис. 10.47).

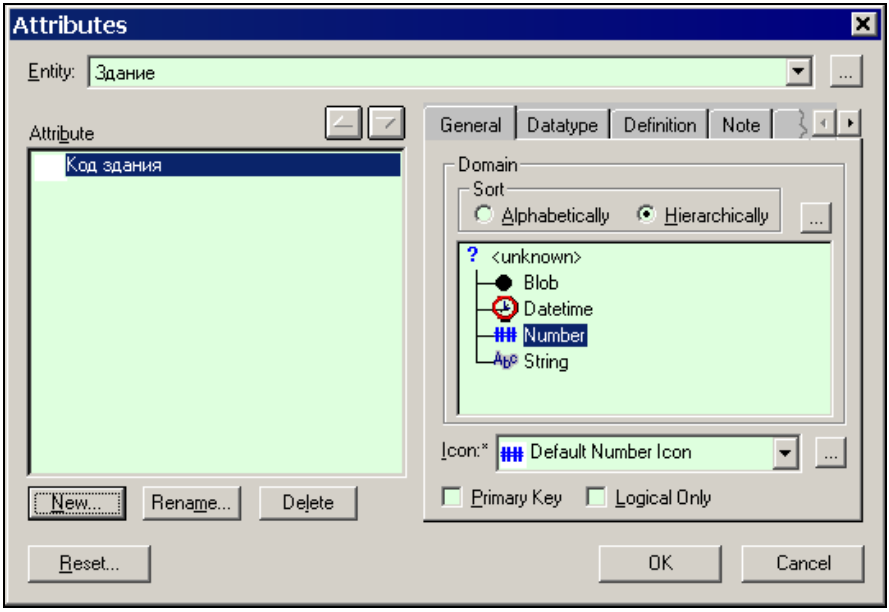


Рис. 10.47. Атрибут "Код здания" в сущности "Здание"

Если установить флажок **Primary Key**, то выделенный атрибут войдет в состав первичного ключа и рядом с его именем появится изображение ключа. С помощью кнопки с тремя точками, расположенной напротив выпадающего списка **Icon**, можно задавать иконку атрибуту.

10.6.2. Второй способ создания атрибута — на основе собственного домена

Создавать атрибуты удобнее, используя домены. Разработчик может создать собственный домен на основе базового типа данных ERwin (базовые типы данных мы уже рассмотрели выше).

Замечание

Домены — это как кубики, из которых мы построим наши будущие сущности. Мы создаем домены, даем им имена, настраиваем и потом собираем из них готовые сущности. Причем один и тот же домен можно использовать несколько раз, например, для случая, когда он используется в родительской сущности и в дочерней для организации связи между ними.

Нам необходимо будет создать домены, исходя из проведенного анализа предметной области и табл. 10.1—10.10.

В табл. 10.11 представлена информация по доменам, которые мы в дальнейшем создадим.

Таблица 10.11. *Домены, необходимые для построения модели базы данных "Библиотекарь"*

Имя домена в логической модели	Имя домена в физической модели	Базовый домен	Тип данных в СУБД (MS SQL Server 2000)	Может ли содержать значения NULL (Allow nulls — может, Not nulls — не может)
Сущность "Здание" (Building)				
Код здания	d_building_id	Number	Int	Not nulls
Статус	d_status	String(20)	Varchar(20)	Not nulls
Адрес (полный)	d_adress	String(100)	Varchar(100)	Not nulls
Телефон	d_telephone	String(15)	Varchar(15)	Allow nulls
Примечание	d_note	String(200)	Varchar(200)	Allow nulls
Сущность "Зал" (Place)				
Номер зала	d_place_no	Number	Int	Not nulls

Таблица 10.11 *(продолжение)*

Имя домена в логической модели	**Имя домена в физической модели**	**Базовый домен**	**Тип данных в СУБД (MS SQL Server 2000)**	**Может ли содержать значения NULL (Allow nulls — может, Not nulls — не может)**
Название (зала)	d_name	String(100)	Varchar(50)	Not nulls
Телефон	d_telephone (домен уже существует, см. сущность "Здание")	String(15)	Varchar(15)	Allow nulls
Внутренний телефон	d_in_telephone	String(10)	Varchar(10)	Allow nulls
Начало работы	d_begin_time	DateTime	Datetime	Not nulls
Конец работы	d_end_time	DateTime	Datetime	Not nulls
Примечание	d_note (домен уже существует, см. сущность "Здание")	String(200)	Varchar(200)	Allow nulls
Сущность "Абонемент" (Abonement)				
Код абонемента	d_abonement_id	Number	Int	Not nulls
Имя (читателя)	d_fio	String(100)	Varchar(100)	Not nulls
Дата выдачи абонемента	d_start_date	DateTime	Datetime	Not nulls
Дата окончания абонемента	d_end_date	DateTime	Datetime	Not nulls
Признак недействительности абонемента	d_priznak_of_end	Number (Boolean)	Bit	Not nulls

Таблица 10.11 *(продолжение)*

Имя домена в логической модели	Имя домена в физической модели	Базовый домен	Тип данных в СУБД (MS SQL Server 2000)	Может ли содержать значения NULL (Allow nulls — может, Not nulls — не может)
Сущность "Книга" (Book)				
Код книги	d_book_id	Number		Not nulls
Название (книги)	d_name (домен уже существует, см. сущность "Зал")	String(100)	Varchar(100)	Not nulls
Общее количество книг	d_count_of_book	Number	Int	Not nulls
Количество книг в наличии	d_count_in_library	Number	Int	Not nulls
Сущность "Автор" (Author)				
Код автора	d_author_id	Number	Int	Not nulls
Имя (автора)	d_fio (домен уже существует, см. сущность "Абонемент")	String(100)	Varchar(100)	Not nulls
Сущность "Жанр" (Ganr)				
Код жанра	d_ganr_id	Number	Int	Not nulls
Название (жанра)	d_name (домен уже существует, см. сущность "Зал")	String(100)	Varchar(100)	Not nulls
Сущность "Услуга" (Service)				
Код услуги	d_service_id	Number	Int	Not nulls

Таблица 10.11 *(продолжение)*

Имя домена в логической модели	**Имя домена в физической модели**	**Базовый домен**	**Тип данных в СУБД (MS SQL Server 2000)**	**Может ли содержать значения NULL (Allow nulls — может, Not nulls — не может)**
Название (услуги)	d_name (домен уже существует, см. сущность "Зал")	String(100)	Varchar(100)	Not nulls
Признак платной услуги	d_priznak_of_pay	Number (Boolean)	Bit	Not nulls
Цена услуги	d_service_price	Number	Int	Allow nulls
Примечание	d_note (домен уже существует, см. сущность "Здание")	String(200)	Varchar(200)	Allow nulls
Сущность "Журнал регистрации услуги" (Service_ control)				
Код записи в журнале	d_record_id	Number	Int	Not nulls
Признак переноса записи в архив	d_priznak_of_arhiv	Number (Boolean)	Bit	Not nulls
Дата оказания услуги	d_date_service	DateTime	Datetime	Not Nulls
Сумма, оплаченная читателем по прайсу, на момент оказания услуги	d_service_price (домен уже существует, см. сущность "Услуга")	Number	Int	Allow nulls
Сущность "Сотрудник" (Worker)				
Код сотрудника	d_personal_id	Number	Int	Not nulls

Таблица 10.11 *(продолжение)*

Имя домена в логической модели	**Имя домена в физической модели**	**Базовый домен**	**Тип данных в СУБД (MS SQL Server 2000)**	**Может ли содержать значения NULL (Allow nulls — может, Not nulls — не может)**
Имя (сотрудника)	d_fio (домен уже существует, см. сущность "Абонемент")	String(100)	Varchar(100)	Not nulls
Должность	d_appointment	String(50)	Varchar(50)	Not nulls
Брак (состоит ли сотрудник в нем)	d_marriage	Number (Boolean)	Bit	Not nulls
Дата приема на работу	d_date_begin_work	DateTime	Datetime	Not nulls
Адрес (проживания)	d_address (домен уже существует, см. сущность "Здание")	String(100)	Varchar(100)	Not nulls
Паспорт	d_pasport	String(15)	Varchar(15)	Allow nulls
Данные диплома	d_diplom	String(15)	Varchar(15)	Allow nulls
Специальность	d_profession	String(50)	Varchar(50)	Allow nulls
ИНН	d_INN	String(15)	Varchar(15)	Allow nulls
Пенсионное свидетельство	d_pension_ document	String(15)	Varchar(15)	Allow nulls
Сущность "Читатель" (Reader)				
Код читателя	d_reader_id	Number	Int	Not nulls
Имя (читателя)	d_fio (домен уже существует, см. сущность "Абонемент")	String(100)	Varchar(100)	Not nulls

Таблица 10.11 *(окончание)*

Имя домена в логической модели	Имя домена в физической модели	Базовый домен	Тип данных в СУБД (MS SQL Server 2000)	Может ли содержать значения NULL (Allow nulls — может, Not nulls — не может)
Адрес (проживания)	d_address (домен уже существует, см. сущность "Здание")	String(100)	Varchar(100)	Not nulls
Паспорт	d_passport (домен уже существует, см. сущность "Сотрудник")	String(15)	Varchar(15)	Not nulls
Дата регистрации	d_date_registry	DateTime	Datetime	Not nulls

В первом столбце таблицы расположены атрибуты сущностей, во втором — домены, на основе которых они должны быть созданы. В прошлый раз мы создавали атрибут "Код здания" на основе домена Number — он является базовым. А в данном случае мы будем создавать атрибуты на основе своих доменов, таким образом, для того чтобы создать атрибуты, нам придется сначала создать домены. В свою очередь, у каждого разработанного нами домена тоже будет базовый домен-предок, на основе которого он создается.

В столбце "Тип данных в СУБД" указано, каким типом будет обладать домен после переноса его в конкретную СУБД. Например, в ERwin есть тип данных `String`, а в MS SQL Server данному типу соответствуют два — `Varchar` и `Char`.

В последнем столбце определяется, может ли атрибут, созданный на основе домена, принимать значения `Null`.

Замечание

Это будет самая сложная часть главы — нам необходимо будет создать все указанные в табл. 10.11 домены, но только в единственном числе. Например, если атрибут "Примечание", построенный на основе домена d_note, встречается в нескольких сущностях, то домен нужно создать один раз и просто добавить его во столько сущностей, во сколько необходимо.

Возможно, на данном этапе не вся информация является понятной, но я предлагаю пройти этот урок до конца и уверяю, многие моменты прояснятся.

Приступаем к созданию домена. Выбираем пункт меню **Model\Domain Dictionary**, появляется окно **Domain Dictionary** (рис. 10.48).

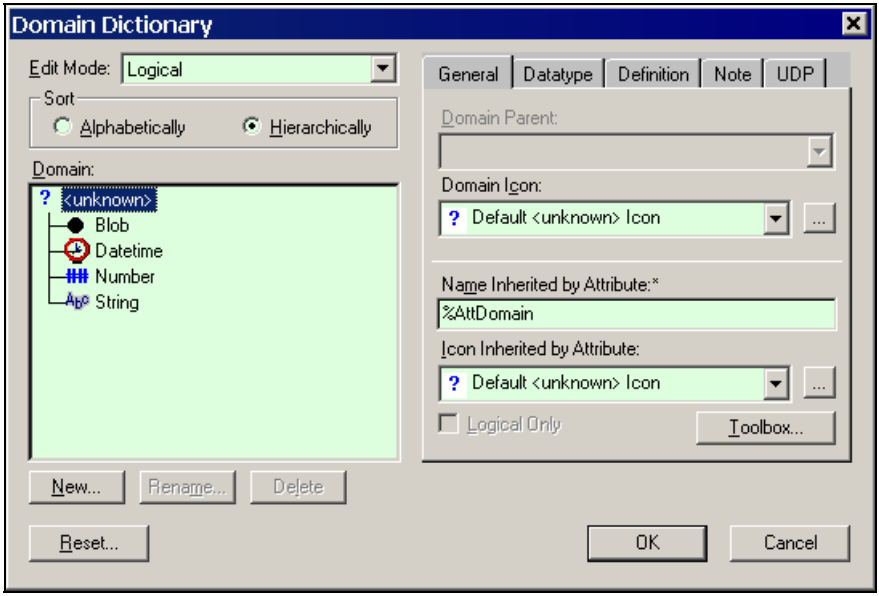


Рис. 10.48. Окно **Domain Dictionary**, предназначенное для работы с доменами

Нажимаем кнопку **New**, появится окно **New Domain** (рис. 10.49).

В поле **Logical Name** необходимо ввести "Код здания" — логическое имя. В поле **Physical Name** необходимо ввести "d_building_id" — физическое имя. В группе элементов **Domain Parent** выбираем домен-родитель, на основе которого будет создан новый домен. Выбираем **Number**. Нажимаем **OK**. Окно примет вид, как на рис. 10.50.

Разберемся подробнее с элементами окна **Domain Dictionary**. Группа элементов **Sort** предназначена для сортировки существующих доменов, она представлена следующими переключателями:

- **Alphabetically** — сортировка в алфавитном порядке (все домены расположены в виде единого списка, упорядоченного в алфавитном порядке);
- **Hierachically** — сортировка в виде дерева, когда вначале идет домен-предок, потом его потомки, — на мой взгляд, это очень удобный режим при работе с доменами.

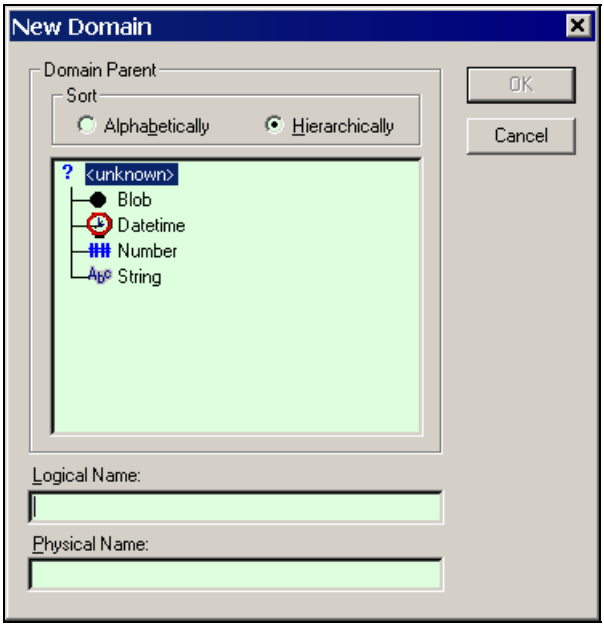


Рис. 10.49. Окно **New Domain**, предназначенное для создания домена

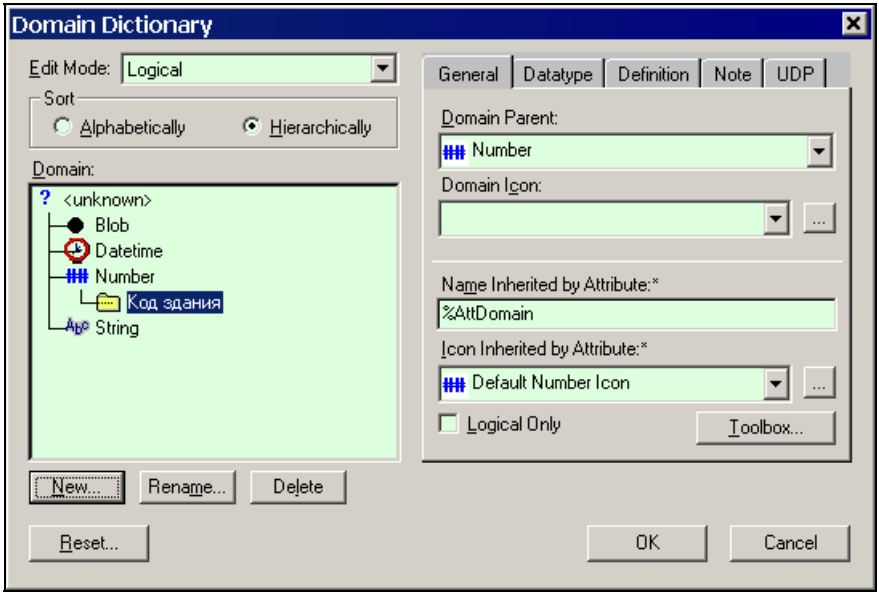


Рис. 10.50. Окно **Domain Dictionary**

Рассмотрим доступные нам элементы на активной по умолчанию вкладке **General**.

- Выпадающий список **Domain Parent** — в нем отображается родительский домен выбранного домена в левой части окна. Родителя можно менять. В нашем случае там отображается **Number**. Если бы мы создали еще один домен на основе домена "Код здания", то в выпадающем списке **Domain Parent** было бы указано "Код здания".
- Выпадающий список **Domain Icon** — в нем содержится пиктограмма домена, она не передается атрибутам, созданным на основе этого домена, а является чисто информационным полем; так как там ничего не выбрано, то домен имеет пиктограмму, данную ему по умолчанию системой в виде папки с точками по центру.
- Поле **Name Inherited by Attribute** — в нем отображается имя, наследуемое атрибутом. Когда мы будем создавать атрибуты на основе домена, то имя атрибуту будет присваиваться на основе скрипта, записанного в этом поле. По умолчанию здесь указано, что имя будет таким же, как и имя домена %AttDomain, пусть так оно и будет, это нас устраивает полностью (работу со скриптом мы рассмотрим немного позже).
- Выпадающий список **Icon Inherited by Attribute** — в нем содержится пиктограмма для атрибута, созданного на основе выбранного домена.

Сделаем активной вкладку **Datatype**. После этого установим флаг **Required**, означающий, что атрибут, созданный на основе домена, будет обязательным (то есть он не может содержать неопределенных значений `Null`) — рис. 10.51.

Теперь нам надо настроить домен для физической модели. В выпадающем списке **Edit Mode** выбираем **Physical** и делаем активной вкладку **General**, окно примет вид, как на рис. 10.52.

Имя домена поменялось на "d_building_id", именно это значение мы задавали в поле **Physical Name** окна **New Domain** (рис. 10.49).

Интерес представляет поле **Name Inherited by Column** — имя, наследуемое полем. В нем указан скрипт для присваивания имени полю на основе домена:

```
%ColDomain
```

Данный скрипт указывает, что имя колонки (поля) `Col` будет таким же, как имя домена — `Domain` — в физической модели, на основе которого она создана. Предположим, что мы не хотим, чтобы поля в таблицах у нас имели в начале букву `d`, так как у нас это признак домена. Для изменения этого воспользуемся командой `Substr()`, которая имеет следующий вид:

```
%Substr(<строка>,<начальная позиция>,<длина>)
```

Например, результатом скрипте вида `%Substr(my_primer,4,6)` будет `primer`.

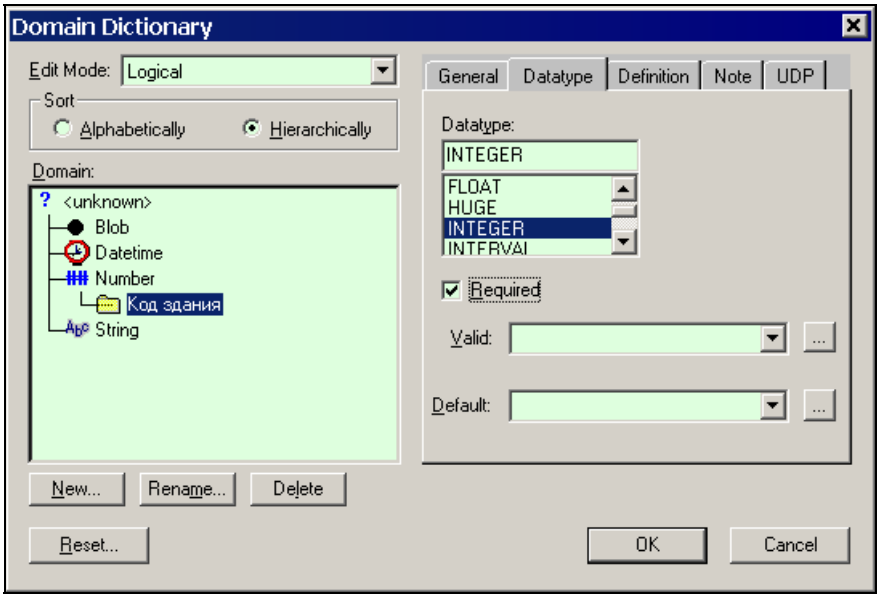


Рис. 10.51. Вкладка **Datatype** окна **Domain Dictionary**

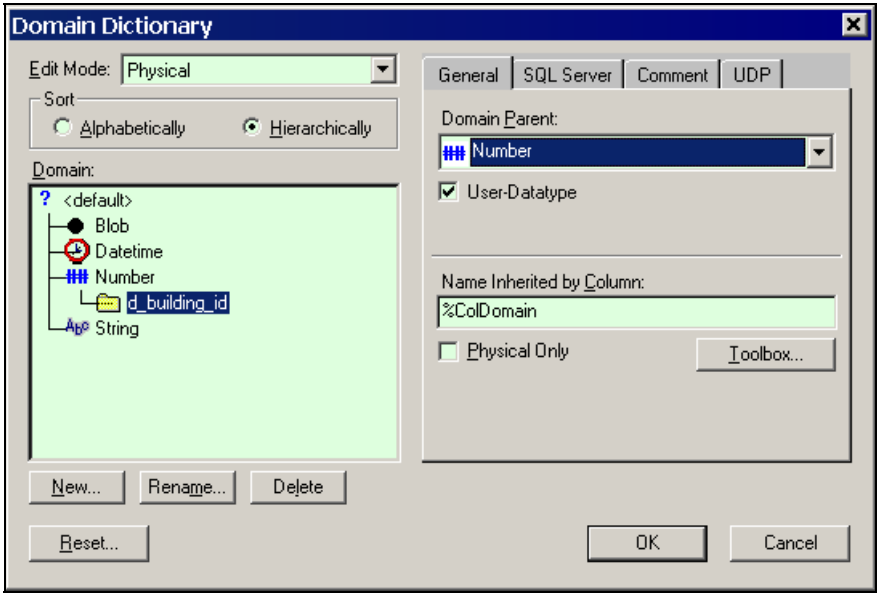


Рис. 10.52. Вкладка **General** окна **Domain Dictionary**

Нам необходимо ввести следующий скрипт для создания имени поля из имени домена, начиная с 3-го символа:

```
%Substr(%ColDomain,3,20)
```

То есть мы берем имя домена (`%ColDomain`) и, начиная с третьей позиции, возвращаем его значение; таким образом, все атрибуты у нас будут без буквы `d` в начале, максимальное имя поля будет равно 20 символам.

Нажимаем **OK**. Поздравляю, мы создали и настроили первый домен. Дальше будет легче.

Теперь нам надо присвоить этот домен атрибуту, который мы создавали не так давно, это делается очень просто. Производим двойной щелчок левой кнопкой мыши на сущности "Здание" и в появившемся окне **Attributes** меняем для пока единственного атрибута базовый домен на "Код здания" (раньше базовым доменом был Number), заодно устанавливаем флаг **Primary Key**, чтобы атрибут был первичным ключом (рис. 10.53).

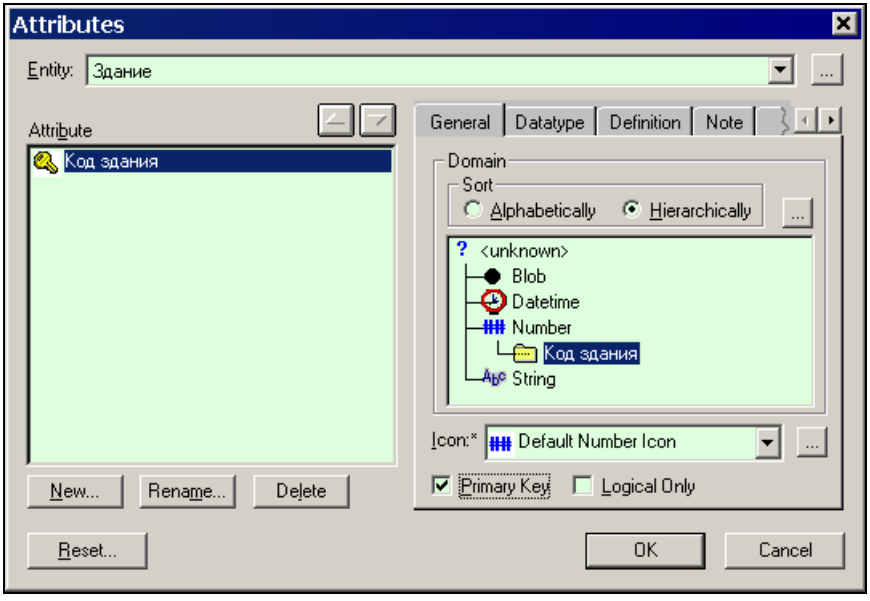


Рис. 10.53. Вкладка **General** окна **Attributes**

Нажимаем **OK**. Таким образом, мы присвоили атрибуту нового родителя.

Приступим к созданию следующего домена. Выбираем пункт меню **Model\Domain Directory**, в появившемся окне **Domain Dictionary** нажимаем кнопку **New**. Появится окно **New Domain**, в группе элементов **Domain Parent** выбираем **String**. В поле **Logical Name** пишем "Статус", в поле **Physical**

Name пишем "d_status", нажимаем **OK**. Переключаемся на вкладку **Datatype** (рис. 10.54).

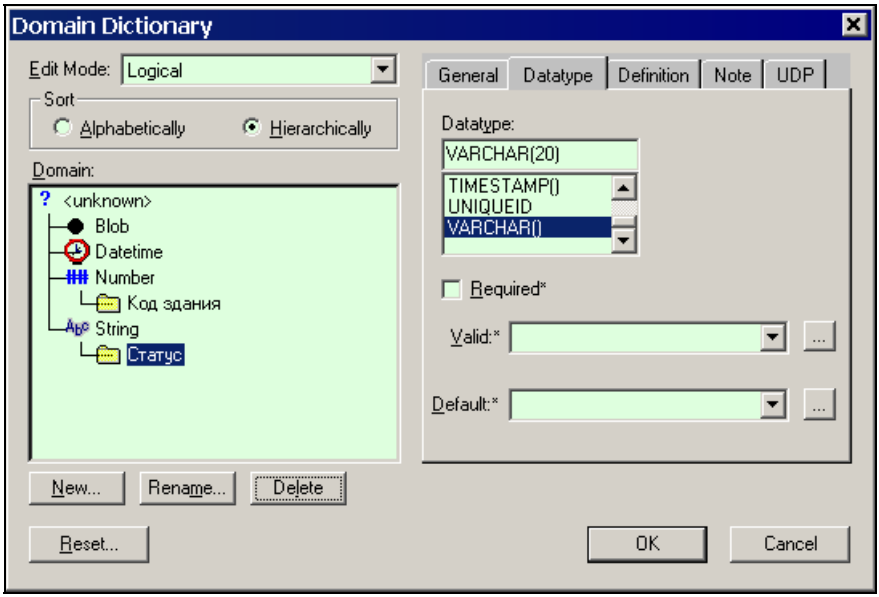


Рис. 10.54. Установка **Datatype** свойства "Статус"

Особый интерес представляет поле **Datatype** со списком под ним, которое позволяет задать тип данных более конкретно. Например, сейчас там указанно VARCHAR(20), но если домен будет содержать большее количество данных, к примеру, такую информацию, как адрес, то с помощью данного поля мы можем внести соответствующие изменения. Также именно здесь мы будем указывать, что домен логического типа, для этого нужно будет выбрать BOOLEAN.

Домен "Статус" не должен содержать пустых значений, поэтому устанавливаем флаг **Required**. После этого в выпадающем списке **Edit Mode** выбираем **Physical** и на вкладке **General** в поле **Name Inherited by Column** указываем скрипт `%Substr(%ColDomain,3,20)`. Нажимаем **OK**. Теперь у нас в проекте существуют два домена — "Код здания" и "Статус".

Далее нам необходимо создать все остальные домены, описанные в табл. 10.11. Для этого, начиная с домена "Адрес", создаем все по порядку, учитывая тот момент, что каждый домен нужно создать только один раз, не обращая внимания, что он может использоваться в нескольких сущностях. Также не забываем исправлять скрипт для присвоения имени атрибуту, соз-

данному на основе разработанного домена в физической модели. Там, где надо, устанавливать флаг **Required**.

Замечание

Если в инспекторе модели перейти на вкладку **Domains**, щелкнуть правой кнопкой мыши на **Domains** и в выпадающем меню выбрать пункт **Sort\Sort Hierarchically**, то можно будет увидеть результат, как на рис. 10.55.

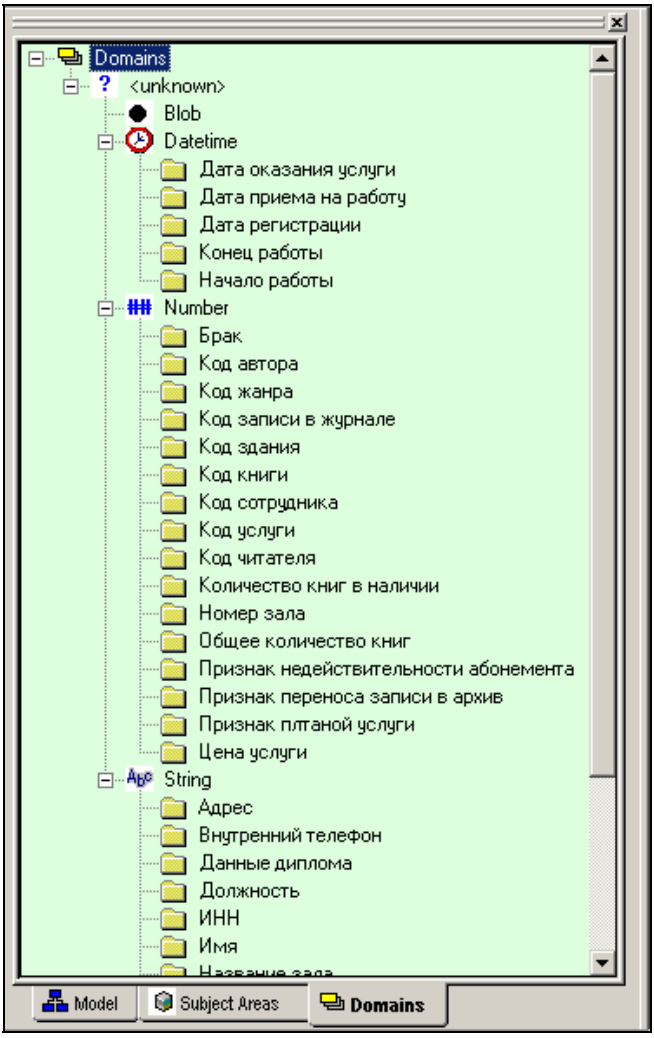


Рис. 10.55. Домены

После того как все домены созданы, можно приступить к созданию атрибутов на основе разработанных нами доменов. Щелкаем правой кнопкой мыши на сущности "Здание" и в выпадающем списке выбираем пункт **Attributes**, появится окно **Attributes**. У нас есть только один атрибут, добавим остальные. Опять обратимся к табл. 10.11, смотрим какие домены мы будем использовать для сущности "Здание".

Нажимаем кнопку **New**, в появившемся окне **New Attribute** выбираем домен "Статус", после этого поля **Attribute Name** и **Column Name** заполнятся автоматически (рис. 10.56).

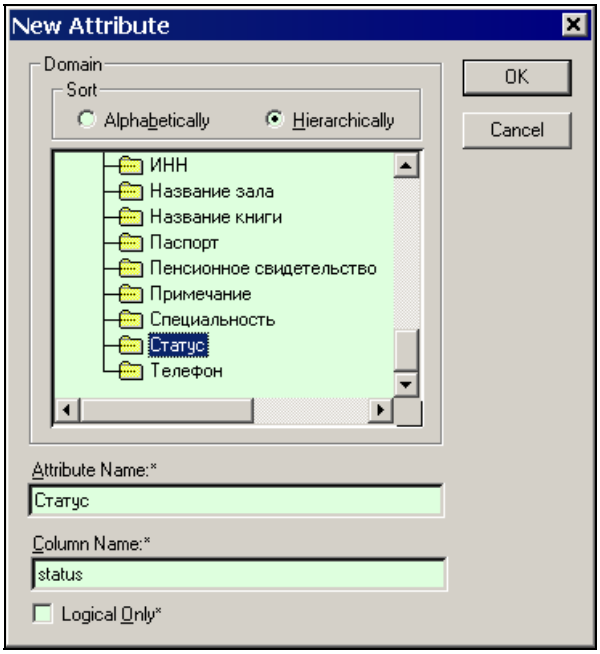


Рис. 10.56. Создание атрибута посредством домена

Нажимаем кнопку **OK**, после этого в сущность "Здание" добавится новый атрибут (рис. 10.57).

Таким же образом необходимо создать остальные атрибуты сущности "Здание":

- ❒ атрибут "Адрес" на основе домена "Адрес";
- ❒ атрибут "Телефон" на основе домена "Телефон";
- ❒ атрибут "Примечание" на основе домена "Примечание".

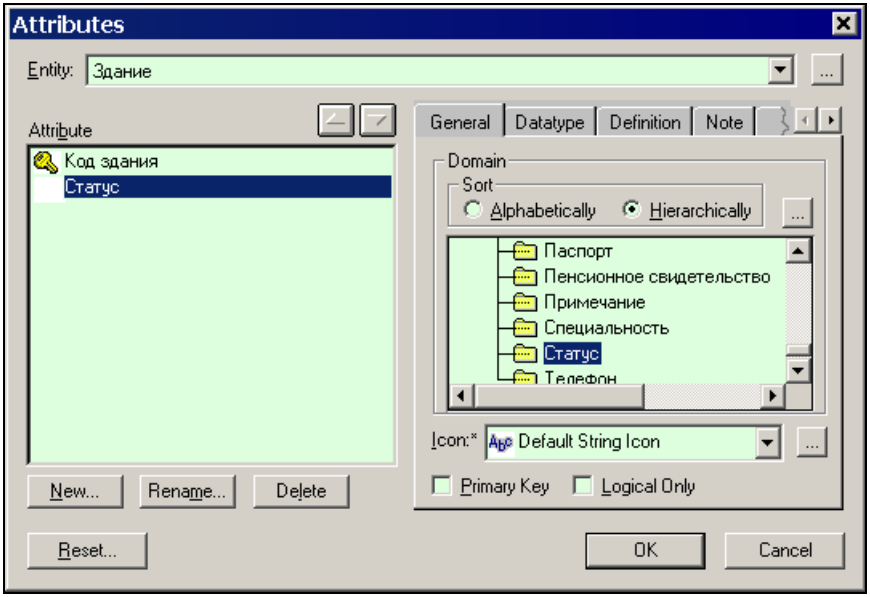


Рис. 10.57. Добавление нового атрибута сущности "Здание"

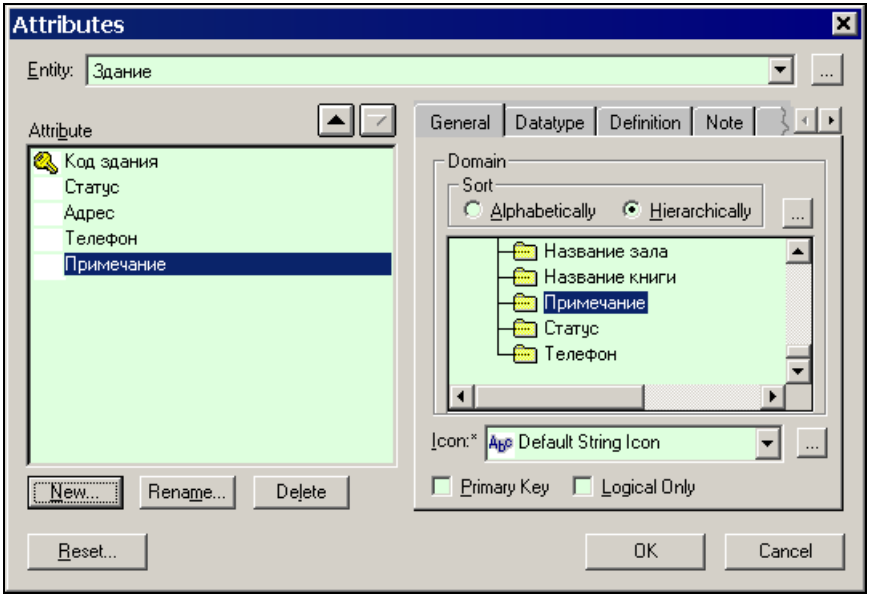


Рис. 10.58. Все атрибуты сущности "Здание"

Нажимаем **OK**. Переходим к физической модели и видим, что названия полей стали на английском, а имя таблицы осталось точно таким же, как и имя сущности — "Здание" (рис. 10.59).

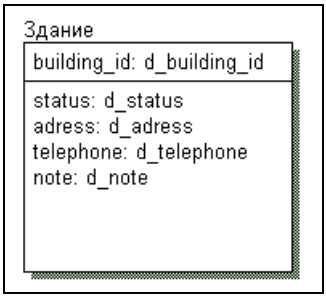


Рис. 10.59. Вид сущности "Здание" в физической модели

Нам необходимо изменить имя таблицы. Щелкаем на ней правой кнопкой мыши и в выпадающем меню выбираем пункт **Table Properties\Comment**, на экране появится окно **SQL Server Tables**, в поле **Name** пишем "Building", после этого нажимаем **OK**.

10.6.3. Третий способ создания атрибута — с помощью инспектора модели

Переключаемся обратно на логическую модель. Создаем новую сущность, называем ее "Зал" (в физической модели таблицу необходимо назвать Place).

Рис. 10.60. Создание сущности "Зал"

В инспекторе модели необходимо сделать активной вкладку **Domains**, теперь щелкаем левой кнопкой мыши по домену "Номер зала" и, не отпуская, перетаскиваем его в сущность "Зал", в область первичного ключа. Добавляем таким же образом остальные атрибуты, но теперь в область атрибутов. В результате должен получиться результат, как на рис. 10.60.

Теперь необходимо добавить в модель все остальные сущности и атрибуты согласно табл. 10.11 (у меня сущности выглядят, как на рис. 10.61—10.68).

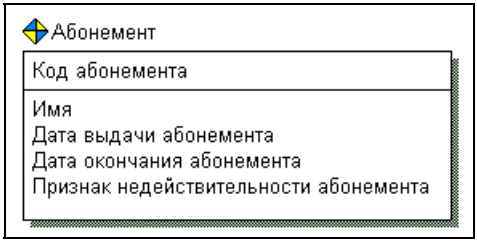


Рис. 10.61. Сущность "Абонемент"

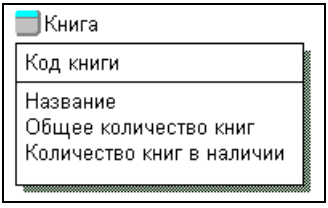


Рис. 10.62. Сущность "Книга"

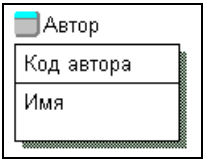


Рис. 10.63. Сущность "Автор"

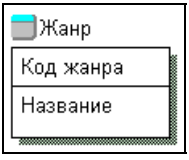


Рис. 10.64. Сущность "Жанр"

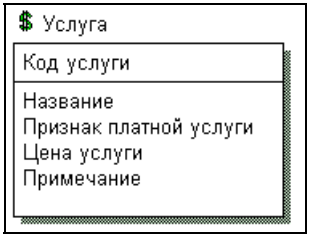


Рис. 10.65. Сущность "Услуга"

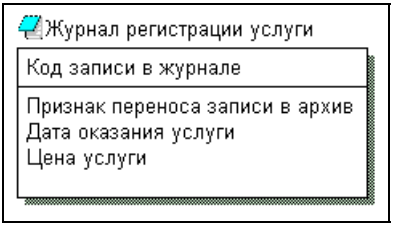


Рис. 10.66. Сущность "Журнал регистрации услуги"

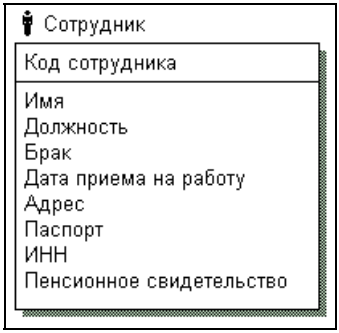


Рис. 10.67. Сущность "Сотрудник"

Рис. 10.68. Сущность "Читатель"

10.7. Определение связей в модели

Определение связей между сущностями (или таблицами) является вторым важным моментом после определения самих сущностей. Как уже известно, существуют зависимые и независимые сущности. Независимая имеет свой собственный первичный ключ, который не зависит от атрибутов других сущностей. Зависимая сущность имеет в своем первичном ключе атрибут другой сущности. Существует два вида связей.

- Идентифицирующая связь — та, которая организуется между зависимой и независимой сущностями, причем ключ независимой сущности мигрирует (добавляется) в зависимую сущность. Таким образом, ключ зависимой сущности состоит из атрибута (или атрибутов) собственных и мигрирующего атрибута. Для того чтобы организовать этот тип связи, необходимо выбрать пиктограмму с непрерывной линией и одним черным кружком на конце в панели **ToolBox** и, соответственно, щелкнуть сначала на главной, потом на дочерней сущности. После этого появится линия, соответствующему типу связи (рис. 10.69).
- Неидентифицирующая связь — та, которая организуется между независимыми сущностям. Для того чтобы организовать этот тип связи, необходимо выбрать пиктограмму с прерывистой линией и с черным кружком на конце в панели **ToolBox** и, соответственно, щелкнуть на одной, потом на другой сущности. После этого появится линия, соответствующая данному типу связи (рис. 10.70).

Приступаем к практике. Выбираем идентифицирующую связь, щелкаем левой кнопкой мыши сначала на сущности "Здание", потом на сущности "Зал", после этого атрибут "Код здания" мигрирует в сущность "Зал" (рис. 10.71).

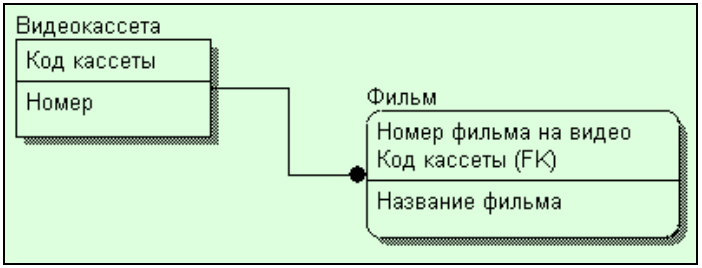


Рис. 10.69. Пример идентифицирующей связи
(на каждой видеокассете может быть несколько фильмов,
первичный ключ сущности "Фильм" состоит из порядкового номера фильма
и кода кассеты, на которой он расположен)

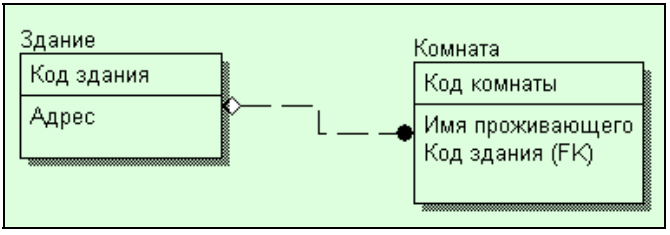


Рис. 10.70. Пример неидентифицирующей связи
(каждая комната принадлежит какому-то зданию)

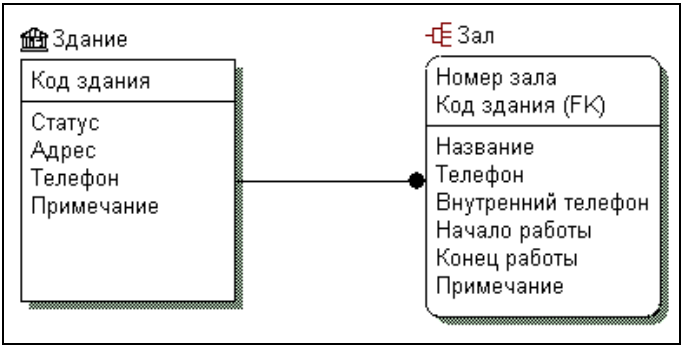


Рис. 10.71. Связывание сущности "Здание" и "Зал"

У зала есть номер, но он не может быть ключевым атрибутом, потому что номера могут повторяться в пределах одной сущности, а связка "Код здания" и "Номер зала" задает однозначную идентификацию каждого зала в пределах сущности, потому что номера залов в одном и том же здании повторяться не могут.

У связи, как и любого элемента модели, есть свойства. Для доступа к ним достаточно щелкнуть два раза на линии, характеризующей связь, левой кнопкой мыши, либо один раз правой, и в выпадающем меню выбрать пункт **Relationship Properties**. Появится окно **Relationships** (рис. 10.72).

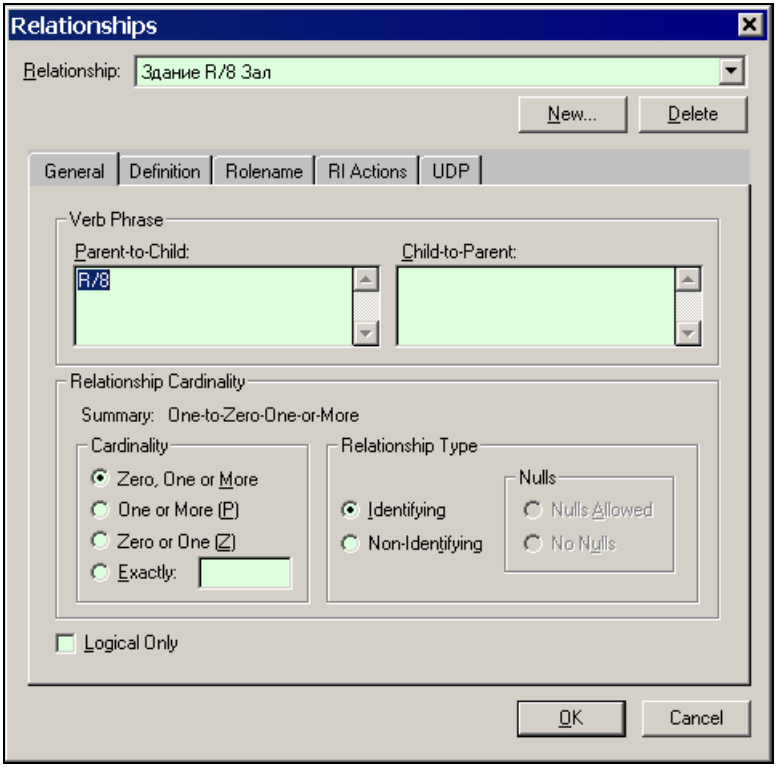


Рис. 10.72. Окно **Relationships**, предназначенное для настройки свойств связи

В верхней части расположено имя, которое по умолчанию составляется из имени сущностей, участвующих в связи, разделенных словом **E/1**, где вместо единицы может стоять любое число, его ERwin ставит автоматически.

В группе элементов **Verb Phrase** указывается ключевая фраза, которая, как правило, представляет собой глагол и описывает отношения между сущностями, например:

- связь между сущностями "Здание" и "Зал" можно охарактеризовать следующим образом: "Здание" состоит из "Залов" — это будет ключевая фраза **Parent-to-Child**, отношения родительской сущности к дочерней.

- Соответственно "Зал" принадлежит "Зданию", это будет ключевая фраза **Child-to-Parent**, отношения дочерней сущности к родительской.

> **Замечание**
>
> Задание ключевой фразы не является обязательным, так как это никаким образом не влияет на свойства и функциональность базы данных, но в то же время это облегчает реализацию, сопровождение и понимание структуры базы данных.

В поле **Parent-to-Child** задаем ключевое слово — "Состоит". В группе элементов **Cardinality** задается количество экземпляров дочерней сущности, соответствующее одному экземпляру в главной (если рассматривать определение данной группы элементов для физической модели, то оно будет звучать, как "в группе элементов **Cardinality** задается количество записей в дочерней таблице, соответствующее одной записи в главной таблице"). Доступны следующие переключатели:

- **Zero, One or More** — одному экземпляру родительской сущности соответствует ноль, один или более экземпляров в дочерней сущности;
- **One or More (P)** — одному экземпляру родительской сущности соответствует один или более экземпляров в дочерней сущности;
- **Zero or One (Z)** — одному экземпляру родительской сущности соответствует ноль или один экземпляр в дочерней сущности;
- **Exactly** — точное соответствие, задается числом.

В группе элементов **Relationship Type** можно поменять тип связи, доступны переключатели:

- **Identifying** — идентифицирующая связь;
- **Non-Indentifying** — неидентифицирующая связь, когда установлен данный переключатель, становятся доступными еще два **Nulls Allowed** и **No Nulls**, которые указывают, может ли атрибут внешнего ключа принимать значение **Null** или нет.

Так как в одном здании, теоретически, может быть любое количество комнат или даже ни одной, если здание новое, поэтому в группе элементов **Cardinality** оставляем установленным по умолчанию переключатель **Zero, One or More**.

Переходим на вкладку **RI Actions**, где будем производить настройки ссылочной целостности (рис. 10.73).

> **Замечание**
>
> Ссылочная целостность определяет, что будет делать СУБД по отношению к таблицам, участвующим в связи, при вставке, удалении и изменении их записей.

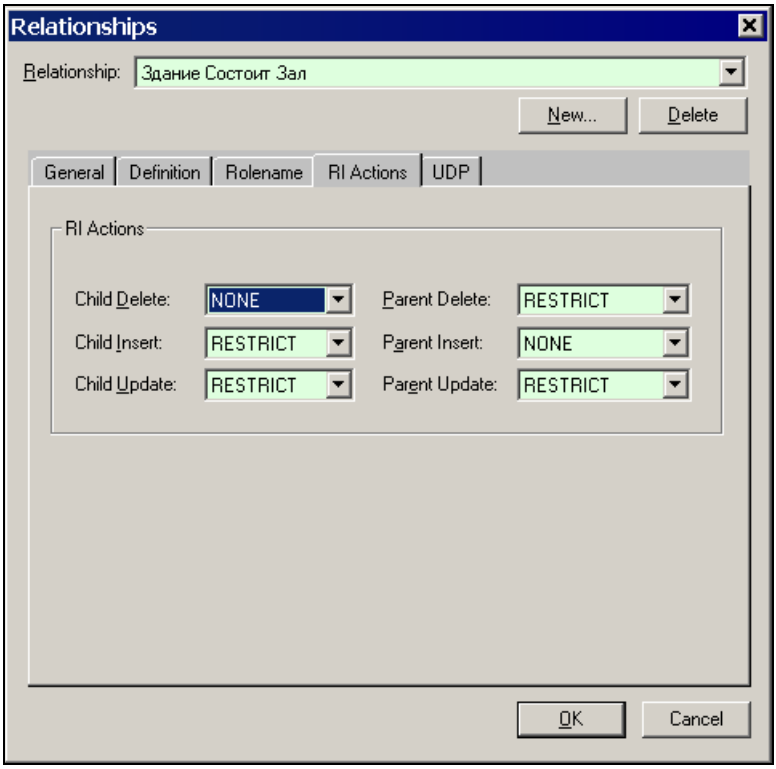


Рис. 10.73. Настройка ссылочной целостности

На данной вкладке находится набор выпадающих списков для каждого действия (добавление, изменение, удаление), возможного для родительской сущности (таблицы) и для дочерней. В них можно выбрать следующие значения:

- **RESTRICT** — запрет на действие;
- **CASCADE** — каскадные изменения;
- **SET NULL** — установка значения `NULL`;
- **SET DEFAULT** — установка значения по умолчанию;
- **NONE** — никаких действий в СУБД не предпринимается.

Нам необходимо установить такие же значения, как на рис. 10.73. Рассмотрим, что они означают:

- **Child Delete**, значение **NONE** — при удалении экземпляра дочерней сущности никаких действий предприниматься не будет (при удалении записи в дочерней таблице в СУБД ничего не произойдет);

- **Child Insert**, значение **RESTRICT** — нельзя добавить экземпляр в дочернюю сущность со ссылкой на несуществующий экземпляр родительской сущности, то есть нельзя добавить комнату саму по себе, она должна обязательно принадлежать какому-то зданию;
- **Child Update**, значение **RESTRICT** — нельзя изменить экземпляр дочерней сущности таким образом, чтобы он ссылался на несуществующий экземпляр родительской сущности;
- **Parent Delete**, значение **RESTRICT** — запрещено удаление экземпляра родительской сущности, если на нее ссылаются экземпляры из дочерней сущности, то есть если у здания есть комнаты, то здание удалить нельзя;
- **Parent Insert**, значение **NONE** — разрешено добавление нового экземпляра в родительскую сущность;
- **Parent Update**, значение **RESTRICT** — нельзя изменить первичный ключ экземпляра родительской сущности, если на него ссылаются экземпляры дочерней сущности.

Одну связь мы настроили, дальше будет проще. Нажимаем **OK**. Создаем неидентифицирующую связь между сущностями "Здание" и "Абонемент", свойства устанавливаем в соответствии с рис. 10.74.

Для данной связи у нас установлен переключатель **No Nulls**, означающий, что абонементы не могут быть сами по себе, они должны обязательно ссылаться на здание библиотеки, где произошла их выдача. Ссылочную целостность оставляем такую же, как на рис. 10.73.

Замечание

Настройка ссылочной целостности для всех связей будет одинаковая, поэтому не будет далее говориться о том, что ее надо настроить таким же образом, как на рис. 10.73.

Нажимаем **OK**. Создаем неидентифицирующую связь между сущностями "Автор" и "Книга". Ключевая фраза — "написал". В группе элементов **Cardinality** устанавливаем переключатель **Exactly** со значением 1. То есть у каждой книги обязательно должен быть автор и одному экземпляру сущности "Книга" может соответствовать только один экземпляр в сущности "Автор". Также необходимо установить переключатель **No Nulls**, так как не может быть книги без автора.

Создаем неидентифицирующую связь между сущностями "Жанр" и "Книга". Ключевая фраза — "соответствует". В группе элементов **Cardinality** устанавливаем переключатель **Exactly** со значением 1. То есть каждой книге соответствует какой-то жанр. Также необходимо установить переключатель **No Nulls**.

Создаем идентифицирующую связь между сущностями "Зал" и "Сотрудник". Атрибуты "Номер зала" и "Код филиала" мигрируют в сущность "Сотруд-

ник". Ключевая фраза — "работает". В группе элементов **Cardinality** должен быть установлен переключатель **Zero, One or More**.

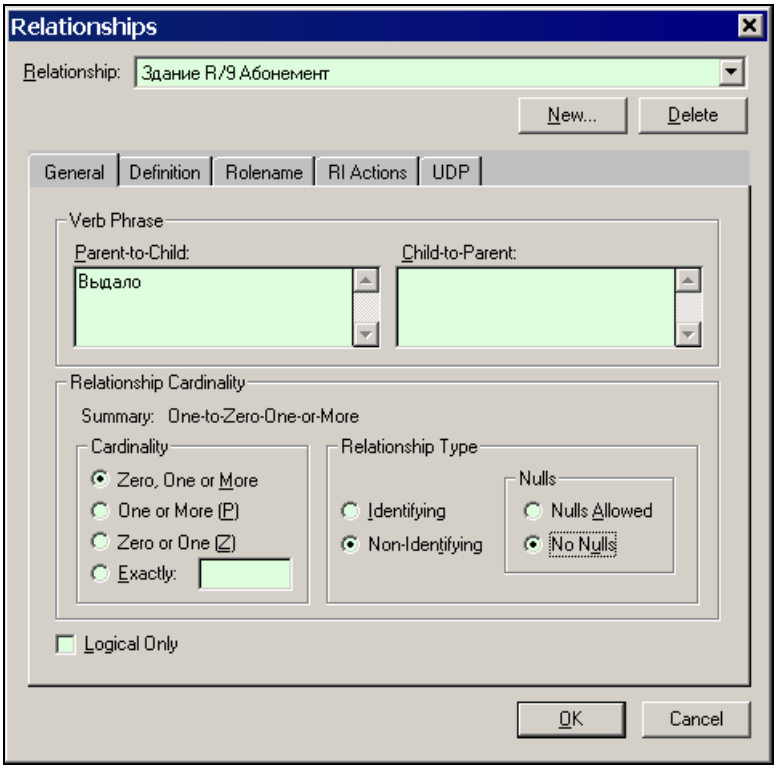


Рис. 10.74. Установка связи между сущностями "Здание" и "Абонемент"

Создаем неидентифицирующую связь между сущностями "Читатель" и "Абонемент". Ключевая фраза — "получает услуги через". В группе элементов **Cardinality** должен быть установлен переключатель **Zero, One or More**. Также необходимо установить переключатель **No Nulls**.

Создаем неидентифицирующую связь между сущностями "Сотрудник" и "Журнал регистрации услуг". Ключевая фраза — "сделал отметку". В группе элементов **Cardinality** должен быть установлен переключатель **Zero, One or More**. Также необходимо установить переключатель **No Nulls**.

Создаем неидентифицирующую связь между сущностями "Абонемент" и "Журнал регистрации услуг". Ключевая фраза — "фигурирует в". В группе элементов **Cardinality** должен быть установлен переключатель **Zero, One or More**. Также необходимо установить переключатель **No Nulls**.

Так как в разных зданиях могут встречаться одинаковые книги, то, соответственно, необходимо связать сущности "Книга" и "Здание", но мы не сможем этого сделать, потому что разработанные сущности не позволяют произвести данную операцию. Данная ошибка была совершена специально, чтобы показать, насколько важно не ошибиться на этапе проведения анализа предметной области. Для исправления ошибки необходимо создать новую сущность под названием "Расположение" (Disposition). Перед этим нам необходимо создать новый домен на основе домена **Number**:

- **Logical name** — Код расположения;
- **Physical name** — d_disposition_id;
- флаг **Required** установлен;
- необходимо указать скрипт в поле **Name Inherited by Column** при выбранном значении **Physical** в поле **Edit Mode**.

Создаем новый атрибут в сущности "Расположение" на основе домена "Код расположения", делаем его первичным ключом.

Теперь нам необходимо переместить атрибуты "Общее количество книг" и "Количество книг в наличии" сущности "Книга" во вновь созданную сущность "Расположение". Для этого необходимо просто навести курсор мыши по очереди на эти атрибуты, курсор примет форму руки. Нажать левую кнопку мыши и, удерживая ее, перенести атрибут в другую сущность. В результате получится сущность как на рис. 10.75.

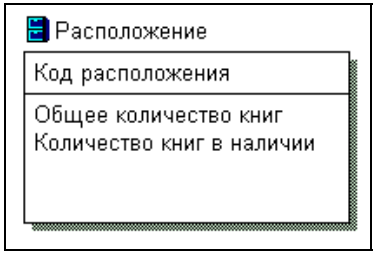


Рис. 10.75. Сущность "Расположение"

Создаем неидентифицирующую связь между сущностями "Книга" и "Расположение". Ключевая фраза — "имеет". В группе элементов **Cardinality** должен быть установлен переключатель **Zero, One or More**. Также необходимо установить переключатель **No Nulls**.

Создаем неидентифицирующую связь между сущностями "Здание" и "Расположение". В группе элементов **Cardinality** должен быть установлен переключатель **Zero, One or More**. Также необходимо установить переключатель **No Nulls**.

Создаем неидентифицирующую связь между сущностями "Услуга" и "Журнал регистрации услуги". Ключевая фраза — "регистрируется". В группе элементов **Cardinality** должен быть установлен переключатель **Zero, One or More**. Также необходимо установить переключатель **No Nulls**.

Настройка связей закончена, в результате схема будет выглядеть как на рис. 10.76.

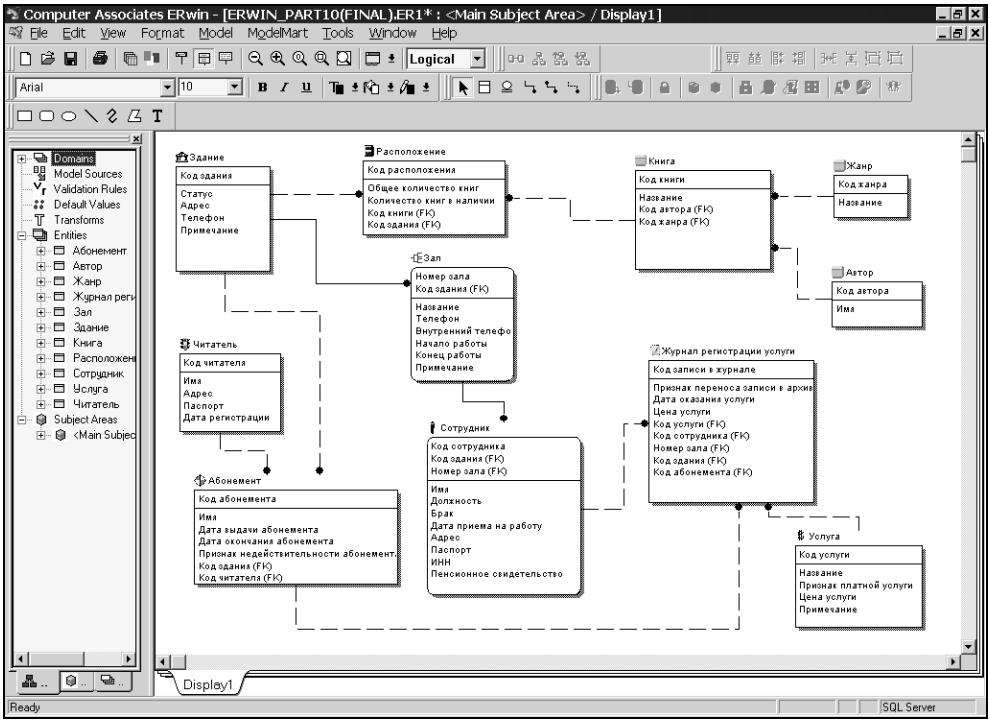


Рис. 10.76. Логическая модель базы данных "Библиотекарь"

10.8. Индексы

Если нужно создать индекс, то необходимо выбрать в логической модели пункт меню **Model\Key groups**, а в физической — **Model\Indexes**.

Появится дополнительное окно, в котором выбирается сущность (таблица), выбирается атрибут (поле) и нажимается кнопка **New**, после чего появится дополнительное окно, в котором можно поменять имя индекса и указать, будет ли индекс обладать уникальностью.

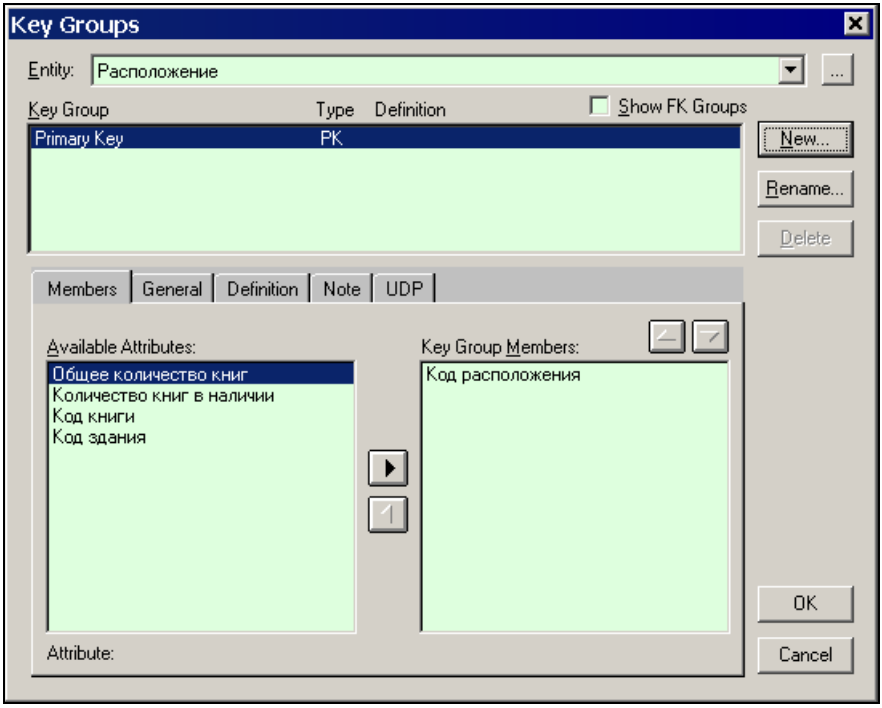


Рис. 10.77. Окно для создания индекса в логической модели

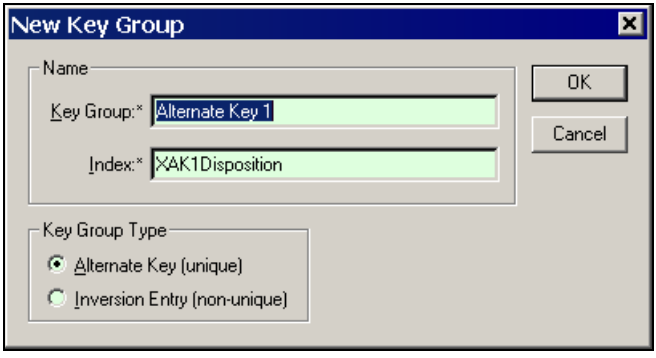


Рис. 10.78. Создание индекса

10.9. Subject Areas

В ERwin есть такое понятие, как *Subject Areas* (предметные области), которое предназначено для разделения модели данных на небольшие области. После

того как мы провели анализ предметной области в *разд. 10.4*, мы разделили все сущности на 3 группы *(см. разд. 10.5)*:

- Группа 1: "Здания и комнаты";
- Группа 2: "Услуги и Документы" (то, за чем читатель приходит в библиотеку, и посредством чего он получает это);
- Группа 3: "Сотрудники и читатели".

Замечание

Когда разрабатывается серьезная база данных, то таких групп (предметных областей) может быть намного больше, то же самое касается и сущностей, которые будут в них содержаться. Работу над отдельной группой можно поручить отдельному специалисту, и ему не будет смысла видеть группы остальных разработчиков.

Для работы с Subject Areas необходимо выбрать пункт меню **Model\Subject Areas**, на экране появится окно **Subject Areas** (рис. 10.79).

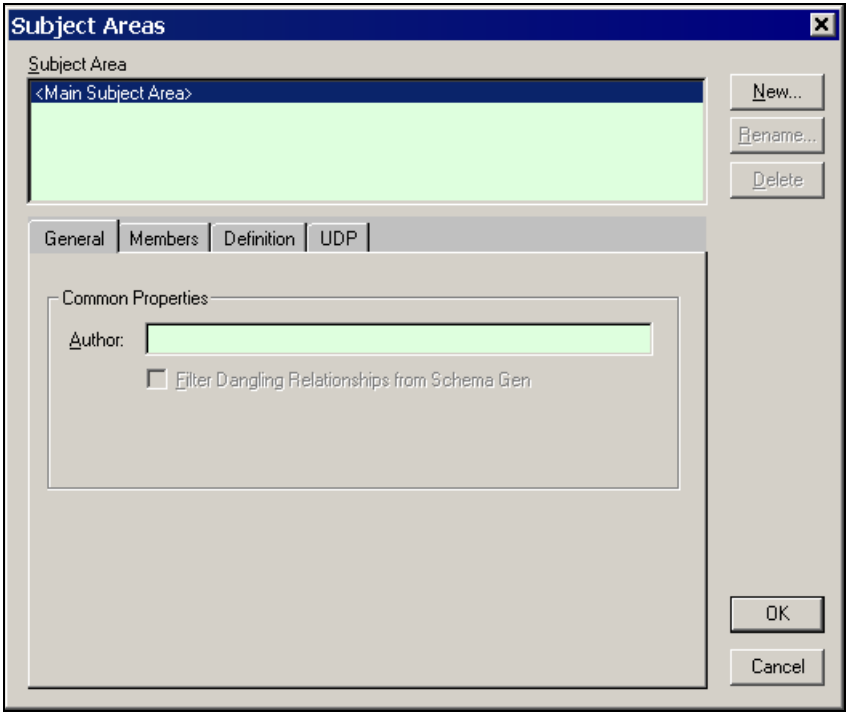


Рис. 10.79. Окно **Subject Areas** предназначено для разделения модели на отдельные области

Замечание

По умолчанию модель содержит всего одну область, называемую **Main Subject Area** (главная предметная область), которая включает все объекты в модели данных.

Нажимаем кнопку **New**, на экране появится окно **New Subject Area**, в поле **Name** необходимо ввести имя создаваемой области (рис. 10.80).

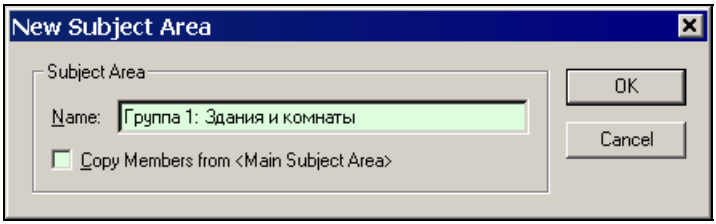


Рис. 10.80. Создание новой области

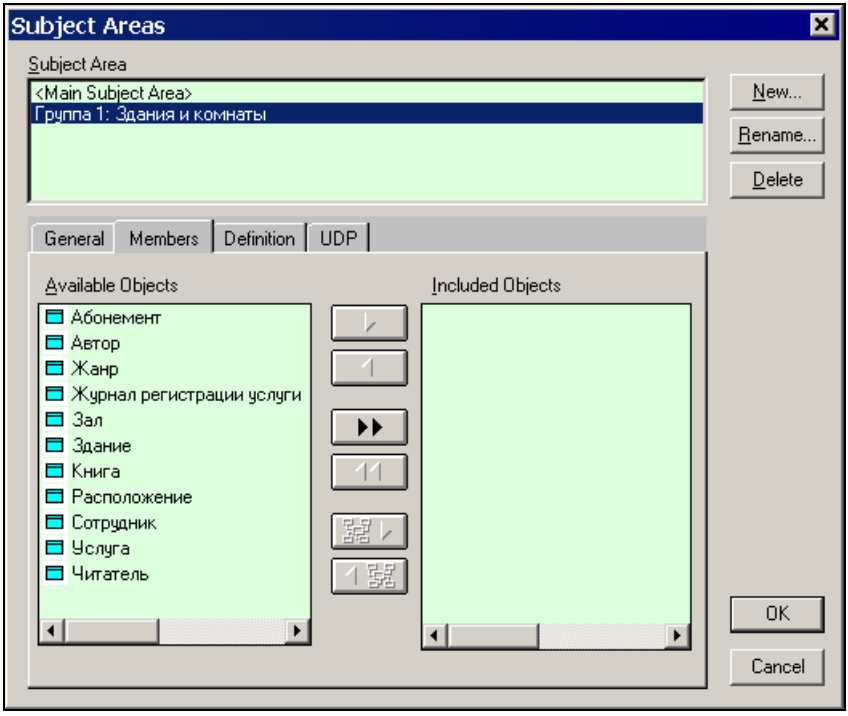


Рис. 10.81. Вкладка **Members** окна **Subject Areas**

Нажимаем **OK**, область добавилась в список. Теперь в поле **Author** вкладки **General** можно ввести имя автора, который создал предметную область. Переходим на вкладку **Members** (рис. 10.81).

С помощью данной вкладки задаются те сущности (таблицы), которые будут видны при активизации создаваемой предметной области. Добавление сущностей в область осуществляется с помощью кнопок с изображениями стрелок. Нам необходимо добавить сущности "Здание" и "Зал", нажимаем **OK**. После этого мы увидим результат, как на рис. 10.82.

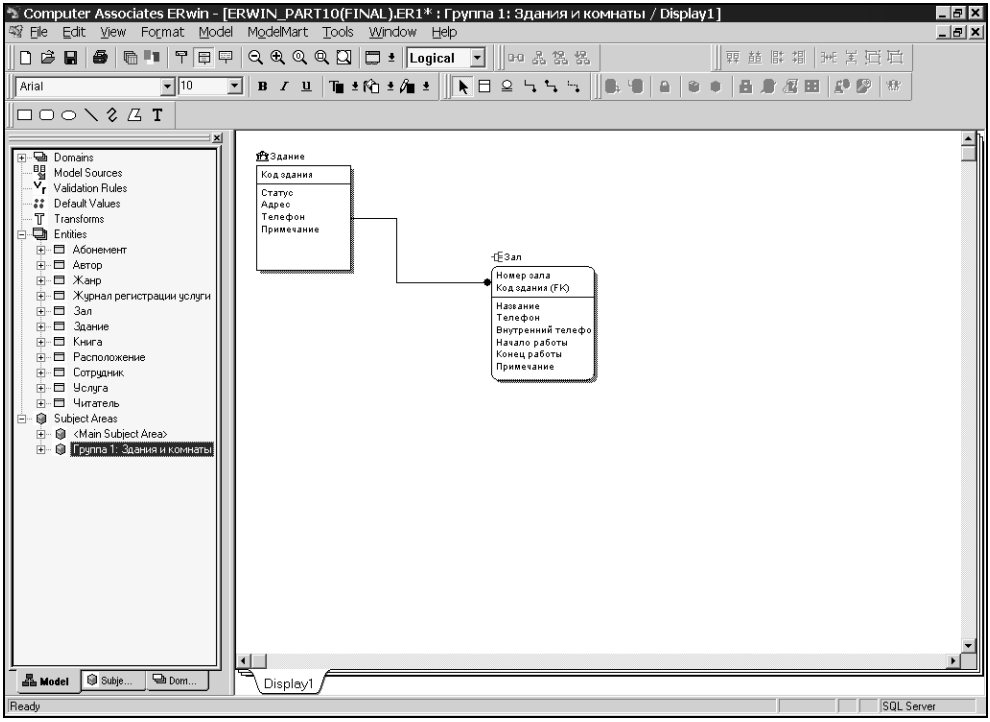


Рис. 10.82. Вид области
с добавленными сущностями "Здание" и "Зал"

На экране остались только выбранные нами сущности, а в инспекторе модели в разделе **Subject Area** стала доступна новая область под названием "Группа 1: Здания и комнаты".

Для того чтобы переключаться между областями необходимо щелкнуть правой кнопкой по ее названию и в выпадающем меню выбрать пункт **Go To**.

Предлагаю вернуться к области **Main Subject Area**.

10.10. Stored Display

Иногда возникает необходимость в том, чтобы изменить вид отображаемой информации, например, мы занимаемся написанием инструкций по базам данных и нас не интересует, из каких атрибутов состоит каждая сущность, мы просто хотим видеть, какие сущности у нас имеются. Для этого случая можно воспользоваться хранимыми отображениями (Stored Display).

Выбираем пункт меню **Format\Stored Display Settings**, на экране появится окно **Stored Display**, принцип работы которого аналогичен с окном **Subject Areas**.

Нажимаем кнопку **New**. Вводим название хранимого отображения — "Entity". Нажимаем **OK**. После этого переключаемся на вкладку **Logical** (рис. 10.83).

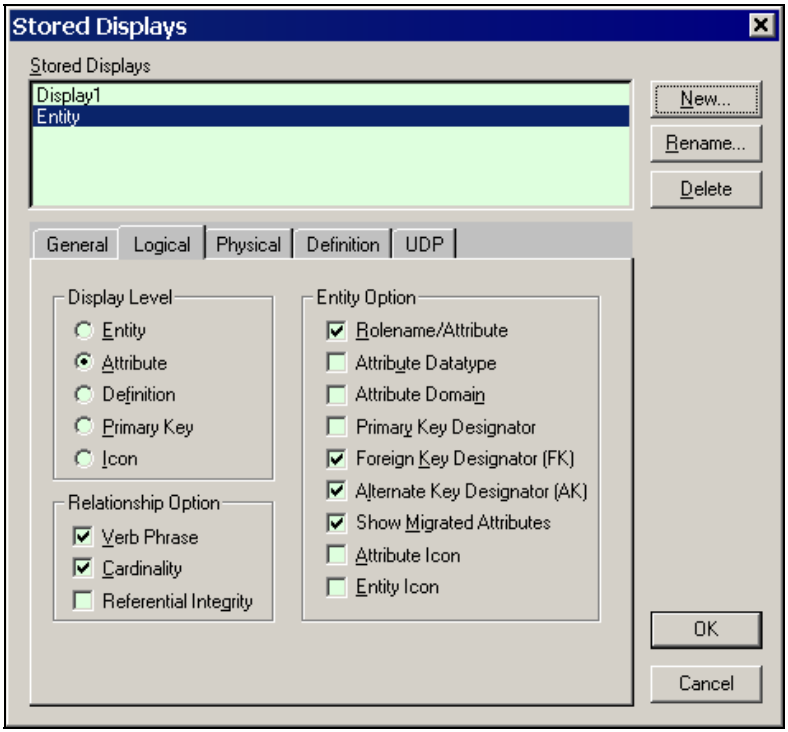


Рис. 10.83. Вкладка **Logical** окна **Stored Displays**

На этой вкладке мы можем указывать, какая именно информация будет видна в логической модели при активизации хранимого отображения (это достигается с помощью установки соответствующих флагов и переключателей). Для

того чтобы отображались только сущности, необходимо установить переключатель **Entity** в группе элементов **Display Level** и установить флаг **Entity Icon** в группе элементов **Entity Options** для отображения иконок сущностей.

Для настройки отображаемой информации в физической модели необходимо перейти на вкладку **Physical**, предлагаю согласиться с установками по умолчанию (они будут дублировать настройки для логической модели). Нажимаем **OK**. После этого в нижней части экрана появится дополнительная вкладка **Entity**, именно она и отвечает за созданное хранимое отображение. Если сделать ее активной, то можно будет увидеть информацию, как на рис. 10.84.

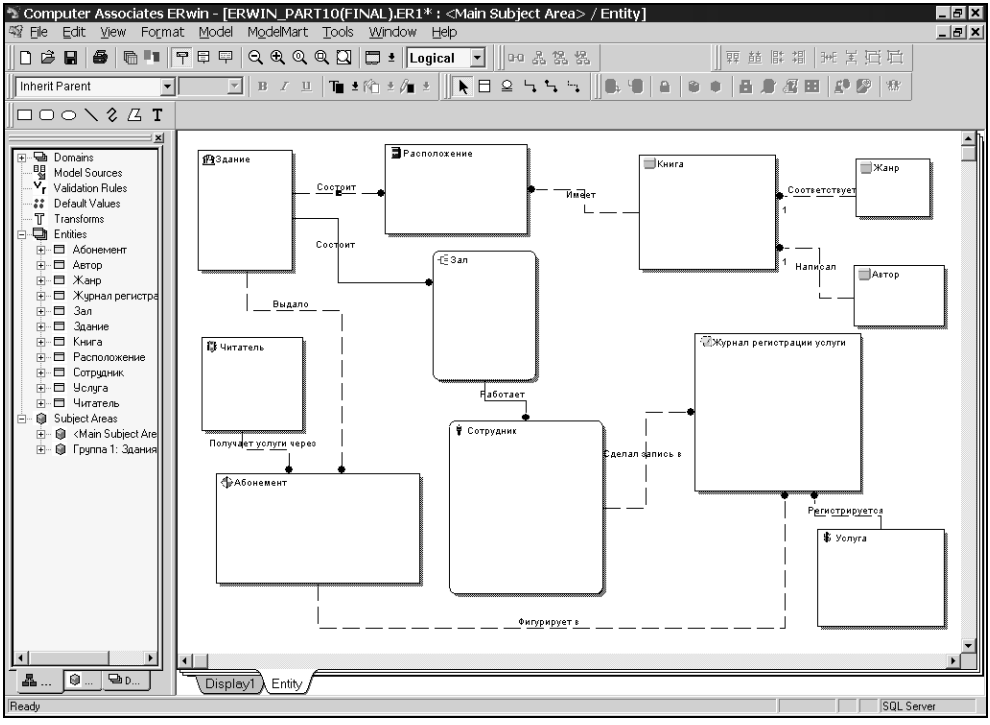


Рис. 10.84. Вид вкладки **Entity**

10.11. ERwin — MS SQL Server 2000

Модель у нас разработана, остается ее только создать на сервере. Эту работу за нас сделает ERwin. Переключаемся в физический режим. Проверяем, чтобы названия таблиц и полей были на английском.

Выбираем пункт меню **Tools\Forward Engineer/Schema Generation**. На экране появится окно, как на рис. 10.85.

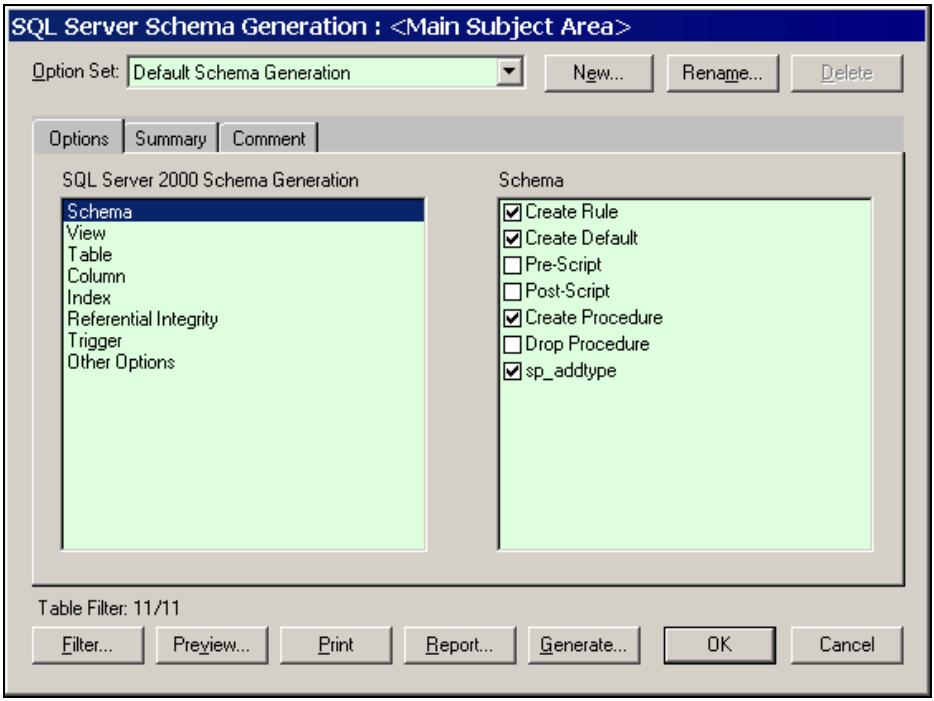


Рис. 10.85. Окно **SQL Server Schema Generation**

В данном окне в левой части расположены категории, а в правой — параметры для выбранной категории. Например, если сделать активной категорию **Table**, то можно увидеть, что установлен флаг для опции **Create Table**. Это означает, что для каждой таблицы, которая присутствует в модели, будет передана на сервер команда `CREATE TABLE`.

Замечание

Предлагаю оставить установки, предложенные по умолчанию, так как этого достаточно для создания полноценной базы данных по разработанной нами модели.

Если нажать на кнопку **Preview**, то можно будет увидеть скрипт, составленный с помощью команд SQL для создания базы данных (рис. 10.86).

Можно сохранить его и выполнить средствами СУБД. Предлагаю закрыть это окно.

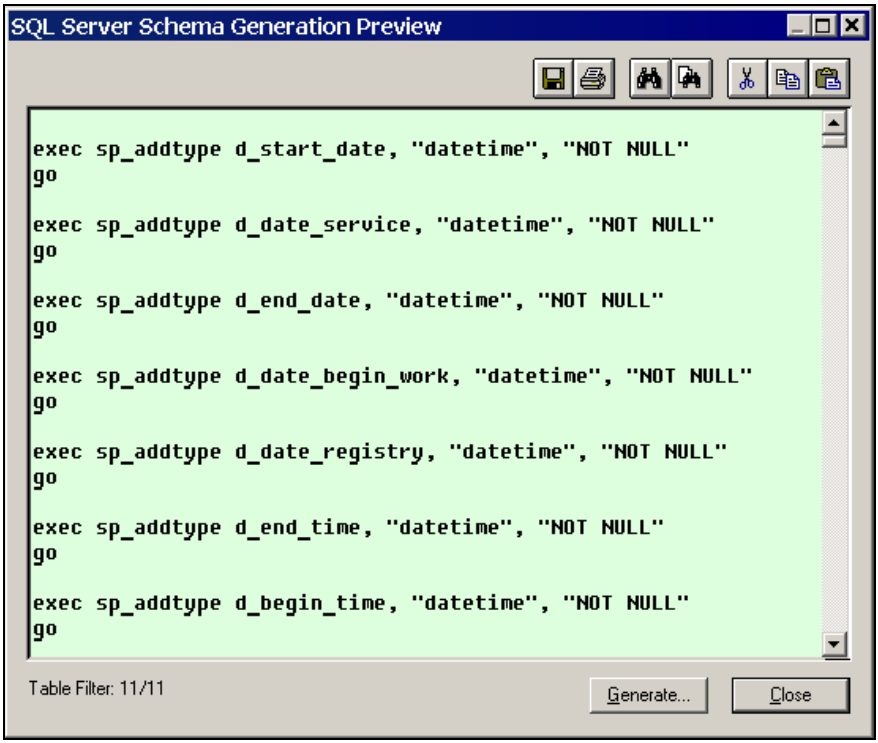


Рис. 10.86. Скрипт для создания базы данных "Библиотекарь"

Для создания базы данных напрямую, то есть когда ERwin самостоятельно подключается к серверу и передает необходимые SQL-команды для создания базы, необходимо нажать кнопку **Generate**. На экране появится окно, в котором необходимо указать (рис. 10.87):

- В поле **User Name** — имя пользователя, под которым будет производиться операция, необходимо при этом, чтобы у указанного пользователя было достаточно прав на проводимые действия;
- В поле **Password** — пароль пользователя;
- В поле **Database** — имя существующей базы данных, предлагаю на сервере создать новую базу под данный эксперимент (у меня база называется Library);
- В поле **Server Name** — имя сервера.

Нажимаем **Connect**, и если все параметры указаны верно и база данных существует, то появится окно **Generate Database Shema** и начнется процесс

создания базы. Все команды, которые будут отправляться на сервер, можно увидеть в данном окне (рис. 10.88).

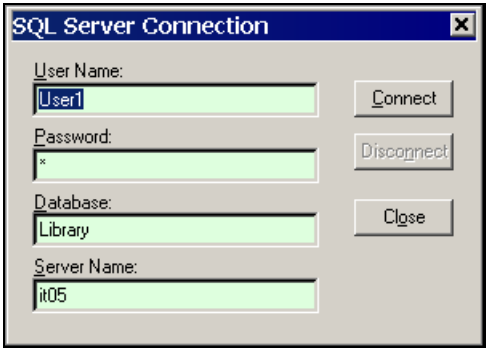


Рис. 10.87. Параметры доступа к серверу MS SQL Server 2000

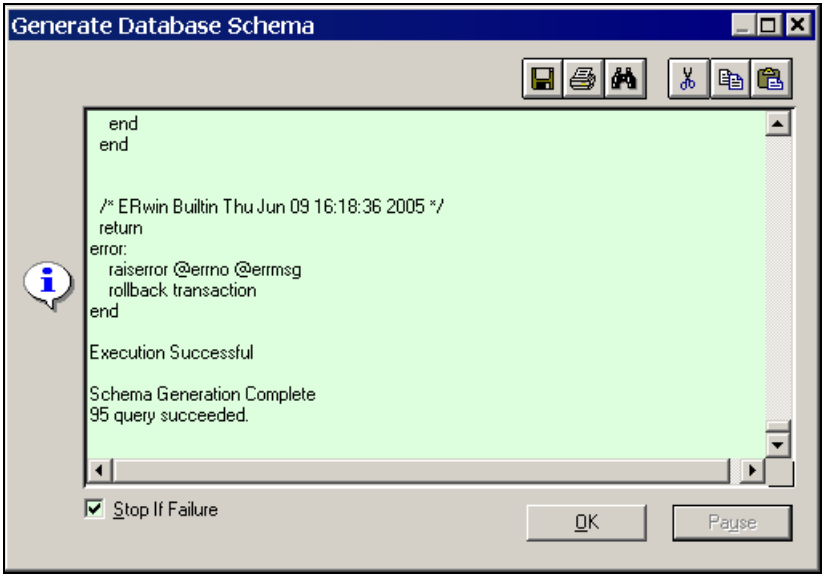


Рис. 10.88. Команды, которые будут отправляться на сервер

После того как база создана, необходимо нажать кнопку **OK**.

Запускаем менеджер SQL Server. Подключаемся к только что созданной базе. Для того чтобы проверить, насколько точно ERwin создал базу, воспользуемся диаграммой. Щелкаем правой кнопкой мыши на пункте **Diagrams** и в выпадающем меню выбираем пункт **New Diagram** (рис. 10.89).

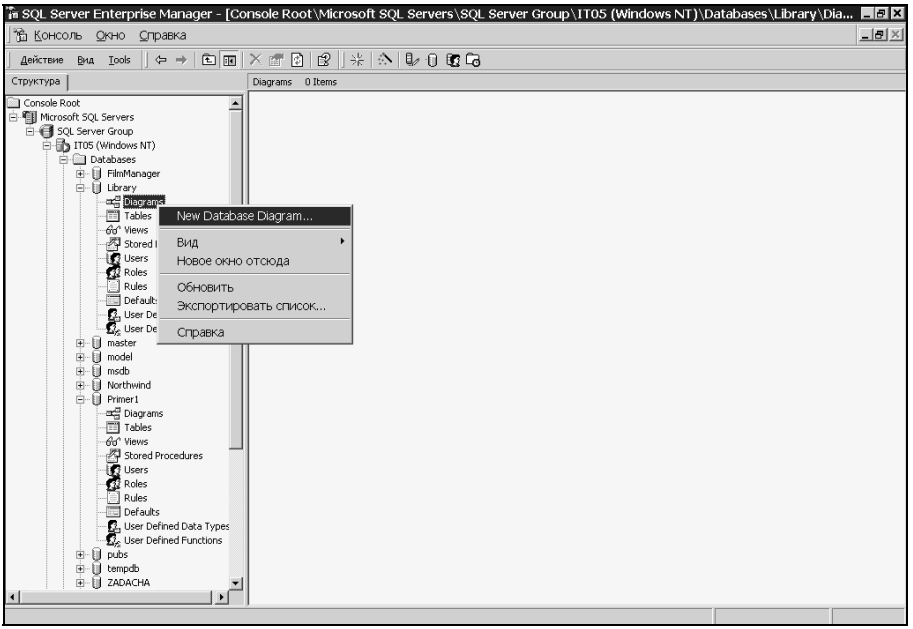


Рис. 10.89. Выбор пункта **New Database Diagram...**

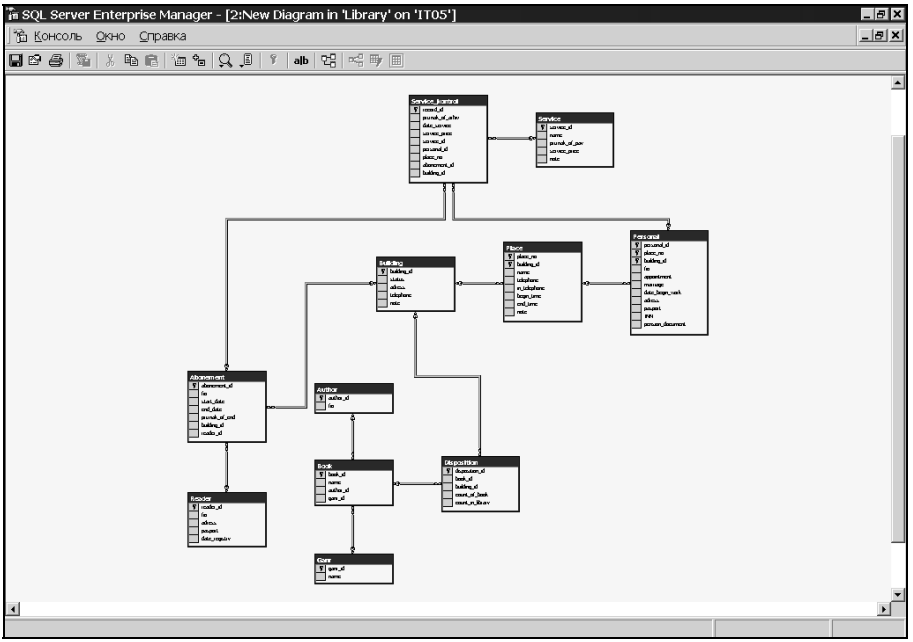


Рис. 10.90. Диаграмма в MS SQL Server для базы данных "Библиотекарь"

Включаем в диаграмму все несистемные таблицы и смотрим, что получилось. У меня диаграмма выглядит как на рис. 10.90.

Средствами менеджера SQL Server необходимо присвоить требуемым полям автоинкрементный тип данных, после чего можно использовать базу данных в своих целях. Например, можно разработать для нее клиентское приложение.

10.12. Печать модели средствами ERwin

Для печати модели необходимо выбрать пункт **File\Print**, на экране появится окно **Print** (рис. 10.91).

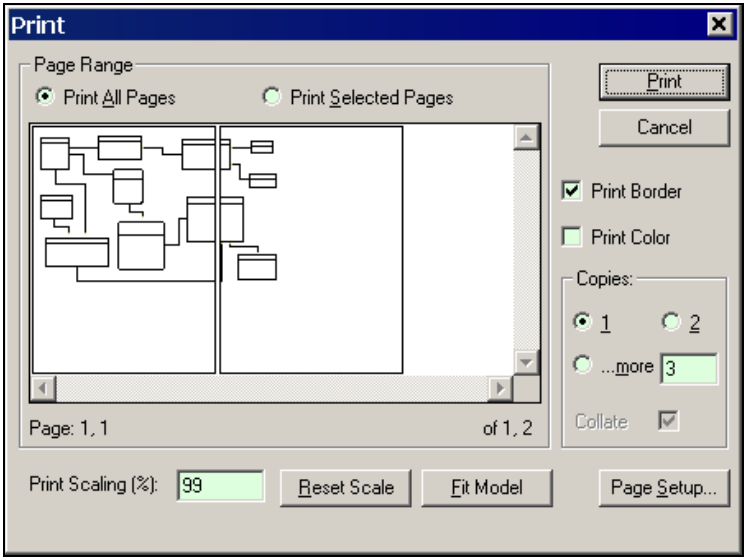


Рис. 10.91. Окно **Print**

Рассмотрим доступные в нем элементы:

- переключатель **Print All Pages** — печать всех страниц;
- переключатель **Print Selected Pages** — печать выбранной страницы, причем выбор осуществляется в этом же окне, где страницы отображаются в виде прямоугольников;
- поле **Print Scaling** — задает масштаб печати;

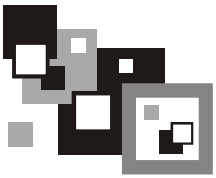
- кнопка **Reset Scale** — задать масштаб 100%;
- кнопка **Fit Model** — задать такой масштаб, чтобы вся схема поместилась на одном листе;
- группа элементов **Copies** — количество копий модели, которое необходимо напечатать.

На компакт-диске в каталоге ГЛАВА 10\SOURCE можно увидеть модель базы данных, разработанной в данной главе.

В каталоге ГЛАВА 10\PICTURE расположены цветные версии рисунков.

В каталоге ГЛАВА 10\VIDEO можно посмотреть видеоролики, посвященные теме, рассмотренной данной в главе.

Глава 11



Обзор дополнительных компонентов

В данной главе рассмотрены дополнительные компоненты, которые позволяют упростить процесс написания приложений. Все рассмотренные компоненты являются бесплатными.

11.1. Ehlib

Содержит компоненты, направленные на расширение возможностей клиентской части программы базы данных при взаимодействии пользователя с приложением (заменяют многие компоненты вкладки **DataControl**).

Официальный сайт: **http://www.farpost.com/personal/dmitryb/RUS/**. Предназначен для версий: 4, 5, 6, 7, 9 (Delphi 2005). В комплект поставки входит демонстрационное приложение, которое показывает возможности данной библиотеки.

11.1.1. Установка

Для установки в Delphi 7 необходимо: скачать дистрибутив с официального сайта. После этого запустить его, и компоненты автоматически установятся в систему.

Для установки в Delphi 2005 необходимо скачать полную версию с исходниками (файл ehlibrus.exe), после этого распаковать архив (пароль "ФЕВРАЛЬФЕВРАЛЬ", об этом сказано на сайте). Далее скопировать содержимое папки Common в папку Delphi9.Vcl, запустить Delphi, выбрать **File\Open** и открыть файл EhLib90.dpk, далее в менеджере проекта щелкнуть правой кнопкой на имени файл и в выпадающем меню выбрать **Compile** (рис. 11.1).

Далее необходимо выбрать **File\Open** и открыть файл DclEhLib90.dpk, после чего в менеджере проекта щелкнуть правой кнопкой на имени файл и в выпадающем меню выбрать **Compile**. После того как операция пройдет, еще раз щелкнуть правой кнопкой мыши на имени файла и выбрать **Install** (рис. 11.2).

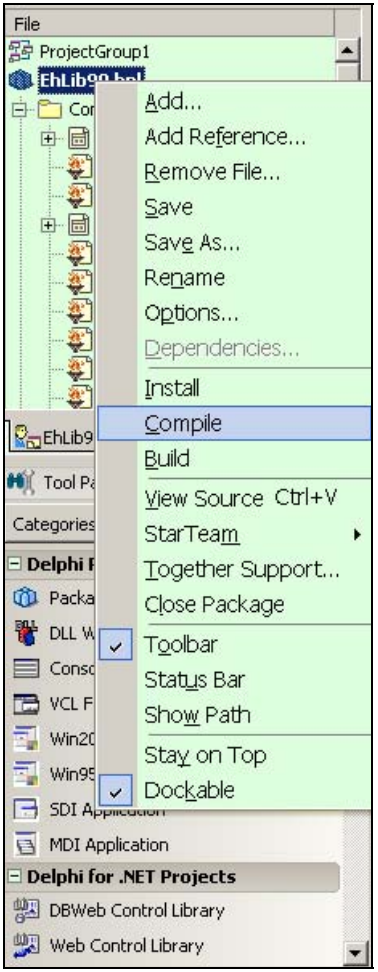


Рис. 11.1. Компиляция компонентов **Ehlib**

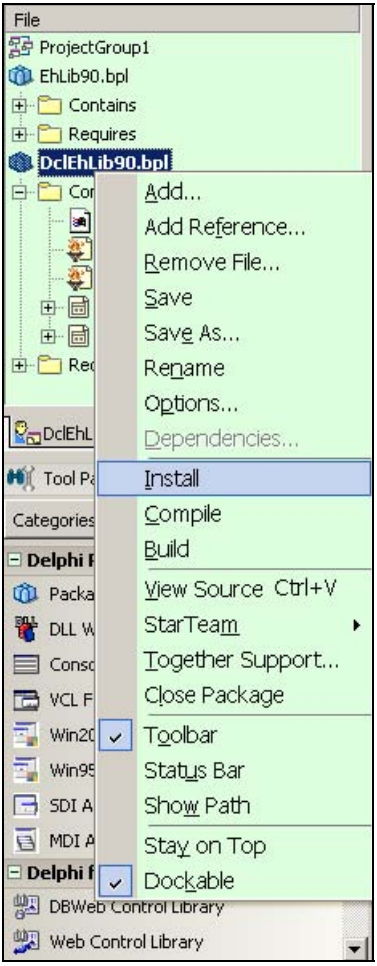


Рис. 11.2. Установка компонентов **Ehlib**

После установки в палитре компонентов появится вкладка **EhLib**.

11.1.2. Работа с компонентами

Быстрый поиск

Размещаем на форме компонент `DBGridEh`, настраиваем его свойство `DataSource`. Открываем свойство `OptionEh` и подсвойства `dghIncSearch` и `dghPreferIncSearch` устанавливаем в `True`.

После этого запускаем приложение, и при вводе символов с клавиатуры в активном поле будет происходить поиск введенного значения (рис. 11.3). Причем найденная запись будет выделена желтым цветом.

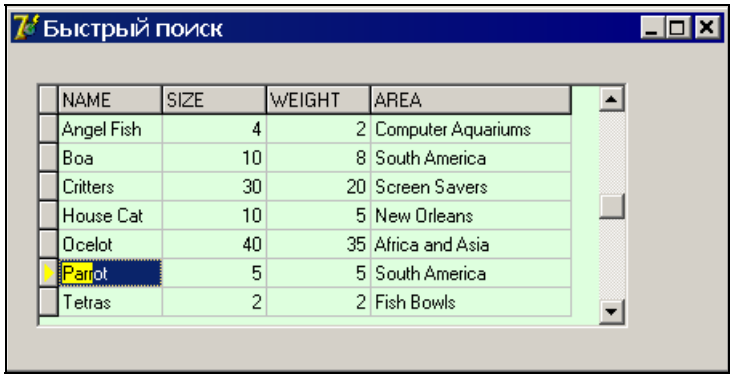


Рис. 11.3. Быстрый поиск

Сортировка по нескольким полям

Размещаем на форме компонент `DBGridEh`, настраиваем его свойство `DataSource`. Открываем свойство `OptionsEh`. Подсвойства `dghAutoSortMarking` и `dghMultiSortMarking` устанавливаем в `True`.

Далее необходимо скопировать в папку проекта файл EhLib[инт].pas, где [инт] — это используемый интерфейс доступа к данным (ADO, BDE и т. д.). Эти файлы поставляются в пакете EhLib (подкаталог DataService).

Подсоединяем файл EhLib[инт].pas в качестве модуля в нужном файле, например:

```
Interface
uses ..
     EhLib[инт];
```

В табл. 11.1 представлена информация о том, какой файл необходимо подключать при использовании в различных технологиях доступа к данным.

Таблица 11.1. *Информация об используемых файлах*

Технология доступа	Компоненты, для которых можно организовать сортировку описанным выше способом	Имя файла для подключения
BDE	`Query`	EhLibBDE.pas
ADO	`ADOQuery`	EhLibADO.pas
ClientDataSet	`ClientDataSet`	EhLibCDS.pas
DBExpress	`SQLQuery`	EhLibDBX.pas
InterBase Express	`IBQuery`	EhLibIBX.pas

Далее необходимо открыть свойство `ColumnDefValuew`, в нем открыть подсвойство `Title` и установить свойство `TitleButton` в `True`.

Теперь можно запускать приложение, и с помощью щелчка левой кнопкой мыши на заголовке полей `DBGridEh` будет произведена сортировка; если при этом удерживать кнопку <Ctrl>, то можно организовать сортировку сразу по нескольким полям (рис. 11.4).

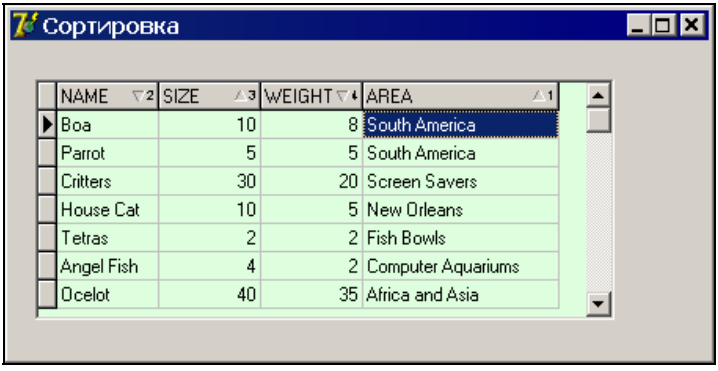


Рис. 11.4. Сортировка

Работа с датой — компонет *DBDateTimeEditEh*

Размещаем на форме компонент `DBDateTimeEditEh`, настраиваем свойство `DataSource` и свойство `DataField`. После этого можно более удобно работать с полями, предназначенными для хранения даты.

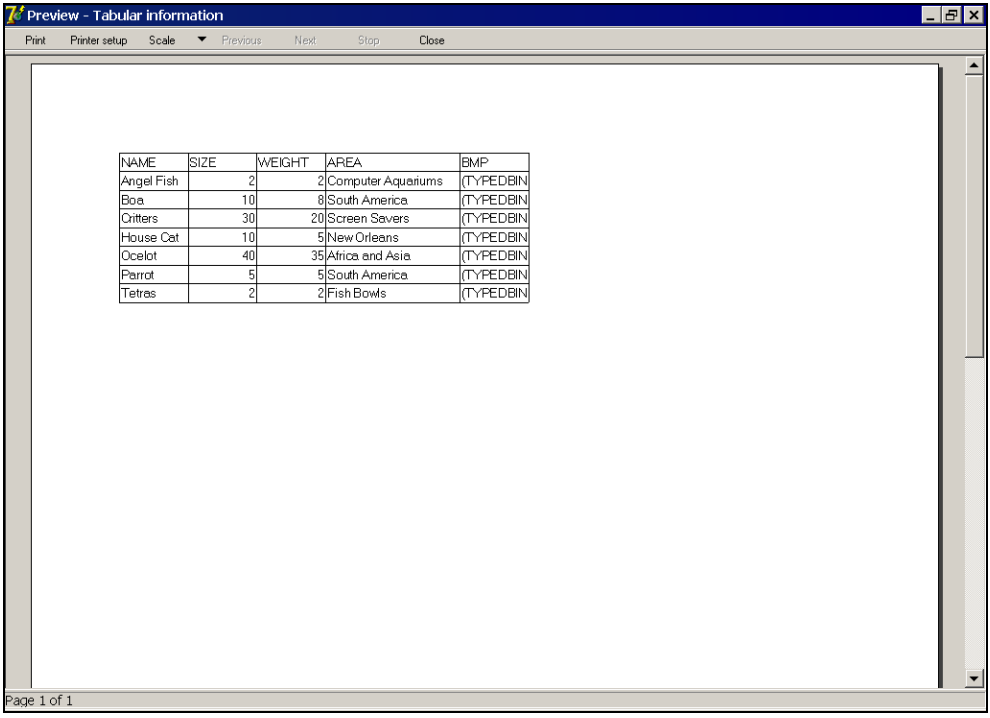
Быстрая печать набора данных

Размещаем на форме компонент `PrintDBGridEh`, в свойстве `DBGridEh` выбираем имя компонента `DBGridEh`, данные которого мы собираемся печатать.

Добавляем кнопку `Button` и пишем для нее код печати набора данных:

```
PrintDBGridEh1.Preview;
```

После запуска программы и нажатия на кнопку, для которой мы писали код, можно будет увидеть результат, как на рис. 11.5.



NAME	SIZE	WEIGHT	AREA	BMP
Angel Fish	2	2	Computer Aquariums	(TYPEDBIN
Boa	10	8	South America	(TYPEDBIN
Critters	30	20	Screen Savers	(TYPEDBIN
House Cat	10	5	New Orleans	(TYPEDBIN
Ocelot	40	35	Africa and Asia	(TYPEDBIN
Parrot	5	5	South America	(TYPEDBIN
Tetras	2	2	Fish Bowls	(TYPEDBIN

Рис. 11.5. Результат предварительного просмотра

11.2. scExcelExport

Позволяет очень просто конвертировать данные из набора данных в Excel.

Официальный сайт: **http://www.scip.be** (ссылка на компонент: **http://www.scip.be/index.php?Page=ComponentsExcelExport&Lang=EN**). Предназначен для версий: 5, 6, 7, 2005. В комплект поставки входит демонстрационное приложение, которое показывает возможности данного компонента.

11.2.1. Установка

Необходимо откомпилировать пакет ExcelExportPack7.dpr (для Delphi 7) или ExcelExportPack9.dpr (для Delphi 2005). После этого надо откомпилировать и установить пакет dclExcelExportPack7.dpr (для Delphi 7) или dclExcelExportPack9.dpr (для Delphi 2005). После установки в палитре компонентов появится вкладка **SC**, содержащая единственный компонент `scExcelExport`. MS Excel должен быть установлен, перед тем как начать работать с компонентом.

11.2.2. Работа с компонентом

Быстрый экспорт данных в Excel

Размещаем компонент `ADOTable`, настраиваем его на работу с данными. Свойство `Active` в `True`. Далее размещаем компонент `scExcelExport`, его свойство `DataSet` настраиваем на `ADOTable`. Добавляем на форму кнопку `Button` и пишем для нее следующий код для быстрого экспорта в Excel:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
 {Задаем название листа в Excel}
 scExcelExport1.WorksheetName := 'Попытка №1';
 {Экспортируем данные}
 scExcelExport1.ExportDataset;
 {Завершаем соединение с Excel}
 scExcelExport1.Disconnect;

end;
```

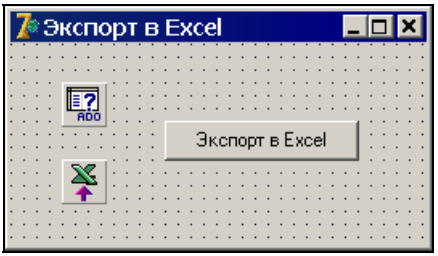


Рис. 11.6. Быстрый экспорт в Excel — вид формы

Если запустить программу и нажать на кнопку, то можно будет увидеть результат, как на рис. 11.7.

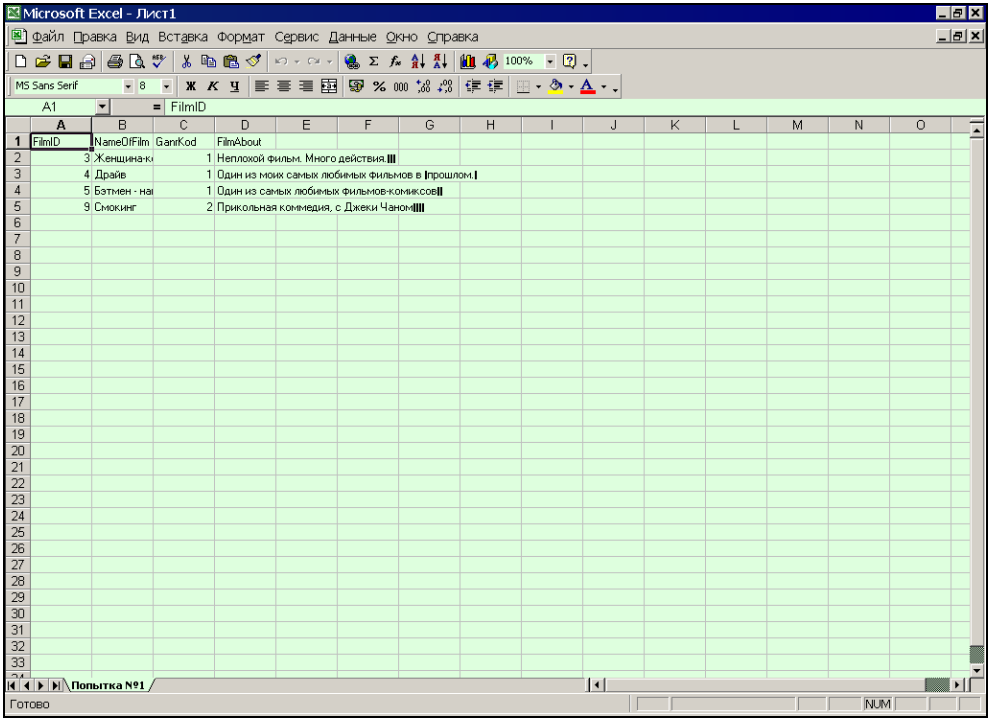


Рис. 11.7. Результат быстрого экспорта в Excel

Экспорт данных в Excel с оформлением ячеек

Предположим, что есть набор данных, в котором есть поле `Summ`, содержащее числовые значения. Нам нужно, чтобы записи, у которых в поле Summ значения больше 10, выделялись красным цветом. Для этого нужно воспользоваться событием `OnGetCellStyleEvent`:

```
procedure TForm1.scExcelExport1GetCellStyleEvent(Sender: TObject;
  Field: TField; var ColorBackground: TColor; FontCell: TxlFont);
begin
 {Если в Excel передаются значения поля Summ, то}
 if Field.FieldName = 'Summ' then
  begin
    {Если значение больше 10, то}
    if Field.Value > 10 then
    begin
      {Фон красный, шрифт Times New Roman, размер 14}
      FontCell.Color := clRed;
```

```
      FontCell.Name := 'Times New Roman';
      FontCell.Size := 14;
    end;
  end;
end;
```

Этот код задает оформление ячеек в Excel; для того, чтобы экспортировать данные, нужно использовать кнопку `Button` со следующим обработчиком события `OnClick`:

```
{Экспортируем данные}
scExcelExport1.ExportDataset;
{Завершаем соединение с Excel}
scExcelExport1.Disconnect;
```

11.3. Trend v.2.03.

Предназначен для построения диаграмм и графиков. Позволяет строить их в реальном времени.

Ссылка для скачивания компонента: **http://www.torry.net/vcl/charts/charts/trend2003.zip**. Предназначен для версий: 6, 7, 2005. В комплект поставки входит демонстрационное приложение, которое показывает возможности данного компонента.

11.3.1. Установка

Необходимо откомпилировать пакет JBKTrendD7.dpr (для Delphi 7) или JBKTrendD9.dpr (для Delphi 2005). После этого необходимо откомпилировать и установить пакет DclJBKTrendD7.dpr (для Delphi 7) или DclJBKTrend9.dpr (для Delphi 2005). После установки в палитре компонентов появится вкладка **Beispiele**, содержащая единственный компонент `JBKTrend`.

Основные свойства:

- `MinValue` — минимальное значение по вертикали;
- `MaxValue` — максимальное значение по вертикали;
- `GridXStep` — ширина шага (количество делений) по оси *X*;
- `GridYStep` — ширина шага (количество делений) по оси *Y*;

- `GridStyle` — стиль сетки;
- `Style` — стиль диаграммы.

11.3.2. Работа с компонентом

Простой пример использования компонента

Помещаем на форму компонент `ListBox` и в свойство `Items` заносим следующие значения:

```
10
30
80
70
20
50
10
90
```

Располагаем компонент `JBKTrend`. Помещаем кнопку `Button` и для нее пишем следующий код:

```
procedure TForm1.Button1Click(Sender: TObject);
var
 i:byte;
begin
 {От первого до последнего элемента ListBox производим
 следующее действие}
 for i:=0 to ListBox1.Items.Count-1 do
   {Добавляем значения элемента ListBox в диаграмму методом Add}
   JBKTrend1.Add(StrToInt(ListBox1.Items[i]));
end;
```

Можно запускать приложение и наслаждаться результатом (рис. 11.8).

Диаграмма в режиме реального времени

Располагаем на этой же форме еще один компонент `JBKTrend`. Далее компонент `Timer` и для события `OnTimer` пишем следующий код:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
{Диаграмма будет строиться на основе
случайных значений в диапазоне от 0 до 100}
```

```
  JBKTrend2.Add(Random(100));
end;
```

Вид формы представлен на рис. 11.9.

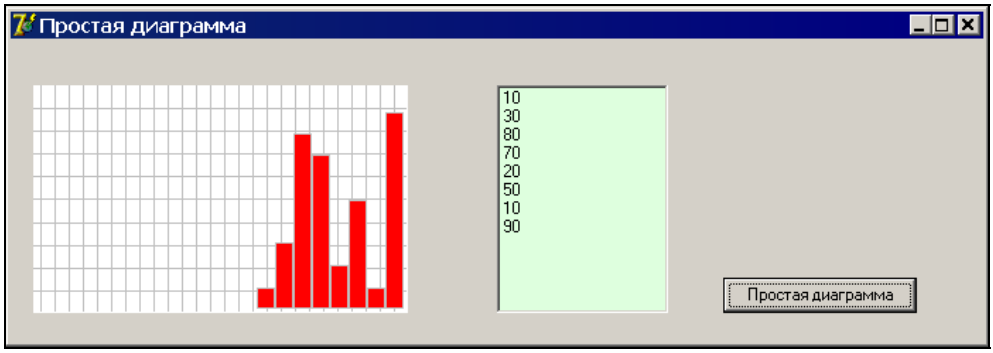


Рис. 11.8. Вид формы **Простая диаграмма**

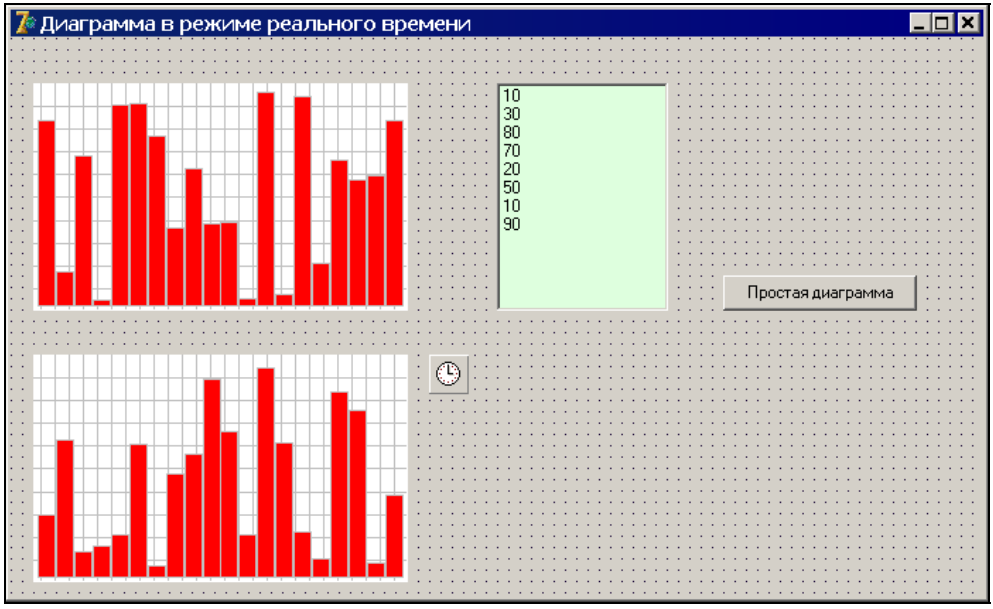


Рис. 11.9. Вид формы **Диаграмма в режиме реального времени**

Если изменить свойство `Style` компонента `JBKTrend2` на `3D`, то можно увидеть следующий результат (рис. 11.10).

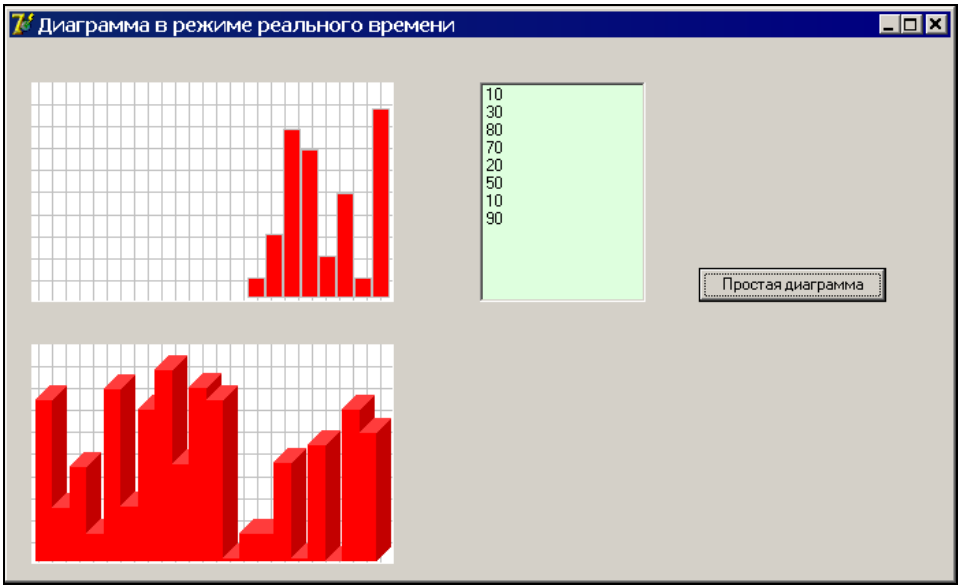


Рис. 11.10. Вид формы **Диаграмма в режиме реального времени** со стилем 3D

11.4. DBGridBelang v.1.0.

Предназначен для отображения различным цветом строк в гриде (является альтернативой стандартному `DBGrid`).

Ссылка для скачивания компонента: **http://www.torry.net/db/visible/db_grids/MyDBGridBelang.zip**. Предназначен для 7 версии. В комплект поставки входит демонстрационное приложение, которое показывает возможности данного компонента.

11.4.1. Установка

Необходимо выбрать пункт меню **Component\Install Component** и установить модуль KytDBGrid.pas. После установки в палитре компонентов появится вкладка **Kiyat**, содержащая единственный компонент `KytDBGrid`.

Основные свойства:

- `WarnaBarisGanjil` и `WarnaBarisGenap` — у этих двух свойств задаются цвета, которые будут повторятся через одну строку (сначала цвет, заданный в `WarnaBarisGanjil`, потом цвет, заданный в `WarnaBarisGenap`, потом опять цвет, заданный в `WarnaBarisGanjil`, и т. д.);
- `WarnaBarisPilih` — в данном свойстве задается цвет активной записи.

Причем вид грида в режиме разработки и в режиме выполнения программы разный. В режиме выполнения цвета отображаются корректно, в режиме разработки — нет, поэтому не пугайтесь (рис. 11.11).

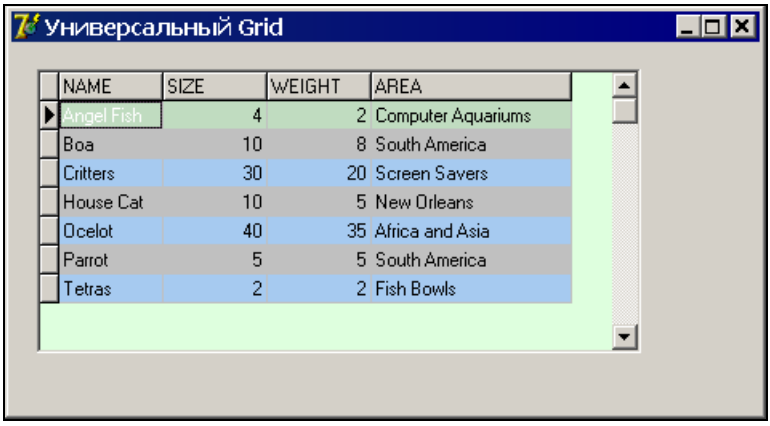


Рис. 11.11. Пример использования `KytDBGrid`

11.5. JanH DBGrid v.1.0

Грид, в котором реализована возможность использования скроллинга мыши. Ссылка для скачивания компонента: **http://www.torry.net/db/visible/db_grids/jh_dbgrid.zip.** Предназначен для версий: 4, 5, 6, 7. В комплект поставки входит демонстрационное приложение, которое показывает возможности данного компонента.

11.5.1. Установка

Необходимо установить пакет JaHDBGridPkgD7.dpk. После установки на вкладке **DBControl** появится компонент `JanHDBGrid`.

11.5.2. Работа с компонентом

Основные свойства:

- свойство `MoveByRows` задает, на какое количество записей произойдет переход при использовании скроллинга мыши;
- в событиях `OnMouseWheelDown` и `OnMouseWheelUp` можно задать собственный обработчик, который будет вызываться при использовании скроллинга мыши.

Для работы с компонентом его достаточно разместить на форме.

11.6. Little Report v.1.0

Компонент для быстрого экспорта набора данных в формат HTML или TXT. Ссылка для скачивания компонента: **http://www.torry.net/vcl/reports/htmlreports/LittleReport.zip**. Предназначен для версий: 5, 6, 7. В комплект поставки входит демонстрационное приложение, которое показывает возможности данного компонента.

11.6.1. Установка

Необходимо выбрать пункт меню **Component\Install Component** и установить модуль LittleReport.pas. После установки в палитре компонентов появится вкладка **Simone DI Cicco**, содержащая единственный компонент `LittleReport`.

11.6.2. Работа с компонентом

Располагаем на форме компонент `LittleReport`, подключаем его к набору данных посредством свойства `DataSet`. Рядом располагаем две кнопки `Button`, в их свойства `Caption` пишем "Экспорт в HTML" и "Экспорт в TXT" (рис. 11.12).

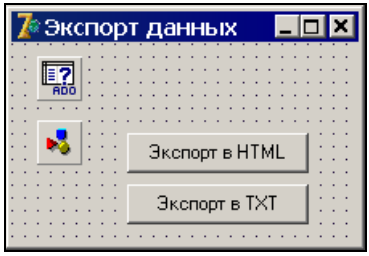


Рис. 11.12. Форма **Экспорт данных**

После чего пишем для кнопок код. Для кнопки **Экспорт в HTML**:

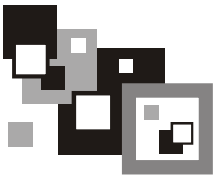
```
LittleReport1.CreateReportHTML('c:\MyRep.html');
```

Для кнопки **Экспорт в TXT**:

```
LittleReport1.CreateReportTXT('c:\MyRep.txt');
```

В каталоге ГЛАВА 11\PICTURE расположены цветные версии рисунков.

Глава 12



Ответы на часто задаваемые вопросы

Большинство идей по поводу вопросов было почерпнуто с форума **VR-ONLINE** (**http://www.vr-online.ru/forum.php**).

- **Вопрос № 1**

  Есть большая база ~800 тыс. человек. Я выгружаю в XML табличку, а потом в новой базе затягиваю ее. Размер получившейся базы больше, чем исходной (примерно: было 170 Мбайт — стало 230 Мбайт). В чем может быть дело?

  Ответ: Возможно, не очищен журнал транзакций. Необходимо использовать метод `MergeChangeLog`.

- **Вопрос № 2**

  Разработано приложение, работающее с СУБД MS SQL Server 2000. Все работает нормально, но каждый раз при запуске программа выводит окно ввода имени пользователя и его пароля. Как убрать это?

  Ответ: У компонента `ADOConnection` свойство `LoginPromt` необходимо установить в `False`.

- **Вопрос № 3**

  Меня интересует следующий вопрос. В Delphi с базами данных можно работать посредством стандартных компонентов `Table`, `Source` и т. п. А можно работать с компонентами с вкладки ADO. В чем между ними разница? Какие компоненты лучше использовать?

  Ответ № 1: Все зависит от того, какую СУБД необходимо использовать. Если DBF или PARADOX, то используйте BDE. Для Access и SQL Server лучше ADO, а для Oracle, Sybase или DB2 лучше dbExpress. Каждая тех-

нология лучше всего работает с определенными базами, хотя некоторые, типа ADO, могут работать с любыми базами через ODBC.

Ответ № 2: Для SQL Server лучше всего использовать компоненты SDAC для Oracle лучше ODAC (**www.crlab.com**).

❐ Вопрос № 4

Возникла необходимость отображать в одном `DBGrid` данные из разных таблиц. Например, название организации и телефон хранятся в таблице Person, а Город в таблице Town. Как это можно организовать?

Ответ: Предположим, структура таблиц следующая:

Таблица Person (IDPerson — ключ, Name — имя организации, Phone — телефон, IDTown — ключ из таблицы Town)

Таблица Town (IDTown — ключ, Town — название города)

Необходимо использовать запрос:

```
SELECT T1.Name, T1.Phone, T2.Town
FROM Person T1 INNER JOIN Town T2 ON T1.IDTown=T2.IDTown
```

❐ Вопрос № 5

Есть таблица, которая содержит 3 столбца: Фамилия, Имя, Номер. Как организовать с помощью ADO сортировку столбцов (по возрастанию, убыванию)?

Ответ: Самый простой способ: использовать `DBGrindEh` из библиотеки EhLib.

❐ Вопрос № 6

Необходимо отобразить весь список людей (поле Name), возвращенный компонентом `ADOTable`, в компоненте `ComboBox`. Как это сделать?

Ответ:

```
ADOTable1.First; // переходим на первую запись
// перебираем все записи
for i:= 0 to ADOTable1.RecordCount - 1 do
begin
 ComboBox1.Items.Add(ADOTable1.FieldValues['Name'].AsString);
 // добавить значение поля 'Name' текущей записи
 // в виде строкового значения в ComboBox
 ADOTable1.Next; // переход на следующую запись
end;
```

❒ Вопрос № 7

Как сохранить картинку в таблицу MS SQL Server?

Ответ: Необходимо сделать тип поля, где будет храниться картинка — `Image`. На форму помещаем `ADOTable`, настраиваем его соответствующим образом. Располагаем еще `DBImage`, `OpenDialog` и `Button`. `DBImage` необходимо настроить на то поле, которое будет хранить картинку. Описанным способом можно работать только с картинками в формате BMP.

Для кнопки пишем код

```
if OpenDialog1.Execute then
begin
 ADOTable1.Edit;
 {Предполагается, что поле, в котором хранится изображение,
 имеет имя Foto}
 ADOTable1Foto.LoadFromFile(OpenDialog1.FileName);
 ADOTable1.Post;
end;
```

❒ Вопрос № 8

Как узнать `FieldName` той колонки в `DBGrid`, по ячейке которой был произведен щелчок мышкой?

Ответ: У компонента `DBGrid` есть обработчик события `OnCellClick`, в нем необходимо написать следующий код:

```
procedure TForm1.DBGrid1CellClick(Column: TColumn);
begin
 ShowMessage(Column.FieldName);
end;
```

❒ Вопрос № 9

Есть БД, предназначенная для конференции. В ней имеется информация об именах участников, их место жительства и т. д. Также есть поле с фотографиями участников. Но проблема в следующем: каждый участник имеет несколько докладов. Как в связанной таблице (и не только в связанной) вставить гиперссылку на внешний файл (DOC из папки source), то есть при щелчке на одной из записей открывать документ в Microsoft Word? При использовании компонента `DBRichEdit` невозможно загрузить рисунки с докладов (доклад при необходимости надо "послать" на принтер).

Ответ: Просто создаем дополнительное поле в нужной таблице. В этом поле будет храниться путь к файлу. Далее создаем подкаталог. Там будут храниться все доклады участников в формате DOC. Помещаем на форму

кнопку, называем ее "Загрузить документ" и пишем для нее код, после выполнения которого загрузится документ, путь к которому находится в переменной AWordDoc. Необходимо будет подключить модуль ComObj.

```
procedure TForm1.Button1Click(Sender: TObject);
var
 MSWord:OLEVariant;
 AWordDoc:string;
Begin
 AWordDoc:='D:\Обращение.doc';

 Try
  MSWord:=CreateOleObject('Word.Application');
 Except
  ShowMessage('Невозможно обратиться к Word');
  Exit;
 end;

 try
  MSWord.Documents.Open(AWordDoc);
  MSWord.Visible:=true;

 Except
  ShowMessage('Ошибка при загрузке документа');
  MSWord.Quit;
 end;

end;
```

❐ Вопрос № 10

Имеются две однотипные таблицы. Из одной таблицы я хочу скопировать данные в другую таблицу. Как лучше это осуществить?

Ответ № 1:

Один из самых простых вариантов — это написать запрос вида:

```
INSERT INTO TableNameToCopy(Field1,Field2,Field3)
SELECT FField1,FField2,FField3 FROM TableNameFromCopy
```

где:

- TableNameToCopy — имя таблицы, в которую будет производиться копирование;

- `Field1,Field2,Field3` — имена атрибутов (столбцов) таблицы, в которые будет производиться вставка данных;
- `TableNameFromCopy` — имя таблицы, из которой будет производиться копирование;
- `FField1,FField2,FField3` — имена атрибутов, из которых будут браться данные.

Ответ № 2: Предположим, используется ADO.

```
ADOTable2.insert;
ADOTable2.fieldvalues['namefield']:=
          ADOTable2.fieldvalues['namefield'];
ADOTable2.post;
ADOTable1.next;
```

❐ Вопрос № 11

Предположим, есть таблица:

```
------------------------
Код |  Имя    | Зарплата |
------------------------
 50 | Иванов |   100    |
 38 | Петров |   300    |
 72 | Рикко  |   250    |
```

Как средствами MS SQL Server составить запрос, который выведет:

```
------------------------
п/п |  Имя    | Зарплата |
------------------------
 1  | Иванов |   100    |
 2  | Рикко  |   250    |
 3  | Петров |   300    |
```

То есть не только выведет записи, но и пронумерует их в порядке следования.

Ответ:

```
CREATE TABLE #Table (
[ID] [int] IDENTITY (1, 1) NOT NULL ,
[Name] [char] (50) COLLATE Cyrillic_General_BIN NOT NULL ,
[Summa] [int]
)
```

```
INSERT INTO #Table (Name, Summa)
SELECT [Имя], [Зарплата] FROM YourTable

SELECT * FROM  #Table

DROP TABLE #Table
```

❐ Вопрос № 12

Нужно организовать работу с базой данных в Delphi таким образом, чтобы внизу окошка была сетка таблицы с данными, и чтобы при нажатии на определенную строчку этой таблицы в специальном окошке этой программы высвечивался рисунок. Например, база данных сотрудников фирмы. Нажимаю на определенную фамилию в таблице, и в окошке появляется фотография. Как это сделать?

Ответ: Необходимо создать дополнительную форму (назовем ее `FormPhoto`), разместить на ней `DBImage`, подключить его к базе и к нужному полю. Потом для обработчика события `OnCellClick` компонента `DBGrid`, по нажатию на запись которого должна высвечиваться форма с фотографией, необходимо написать код:

```
{У компонента DBGrid есть такое событие, как OnCellClick,
в котором передается номер колонки,
в которой щелкнули мышкой
(обратите внимание, что нумерация начинается с нуля).
Например, таблица состоит из трех полей:
Номер, Название, Фотография.}
If Column.Index=2 then
FomShowPhoto.Show;
```

❐ Вопрос № 13

Необходимо, чтобы по нажатию на кнопку в таблицу заносилась дата из компонента `MonthCalendar`.

Ответ: Для кнопки надо написать следующий код (предположим, что поле, в которое будет происходить добавление даты, называется Data):

```
Table.Edit; {Переводим в режим редактирования набор данных}
Table.FieldByName('Data').Value := MonthCalendar.Date;
Table.Post; {Сохраняем изменения}
```

❐ Вопрос № 14

Проблема такая: как можно создать в Excel шаблон и забить его значениями из базы данных?

Ответ: Создаем в Excel шаблон. Создаем в Delphi документ на основе шаблона:

```
app:=createoleobject('Excel.Application');
app.WorkBooks.Add(ExtractFilePath(Application.ExeName)+
'шаблон.xlt');
```

Так обращаемся к ячейке:

```
App.WorkBooks[1].WorkSheets[1].Cells[7,4].Value:=данные из базы
```

Так закрываем Excel:

```
App.ActiveWorkBook.Close(false);
App.Quit;
```

Необходимо будет подключить модуль `ComObj` и объявить переменную `app` типа `variant`.

❐ Вопрос № 15

Почему не работает следующая строка кода:

```
AdoTable1.fieldvalues['test']:='test';
```

Выдает ошибку `"... not edit or insert mode"`.

Ответ: Потому что, перед тем как внести в таблицу изменения, надо перевести ее в режим редактирования:

```
ADOTable1.Edit; {При редактировании записи}
ADOTable1.Insert; {При добавлении записи}
```

❐ Вопрос № 16

Есть MySql Server, нужно к нему подключиться. Как лучше это сделать?

Ответ: На вкладке **dbExpress** есть компонент `sqlConnection`, у него есть свойства `connectionName` и `driverName`. Их значения нужно выставить в `mysql`, а дальше все как обычно.

❐ Вопрос № 17

Необходимо организовать поиск в ADOTable. Как это можно сделать?

Ответ: Надо использовать метод Locate.

Для поиска по точному совпадению:

```
Locate('Имя поля',Искомое значение,[]);
```

Для поиска по приблизительному совпадению:

```
Locate('Имя поля', Искомое значение,
[loCaseInsensitive, loPartialKey]);
```

Пример:

```
ADOQuery1.Locate('Name',Edit1.Text,[])
```

❐ Вопрос № 18

Подскажите, пожалуйста, как можно создать фильтр (без колдовства с sql), используя ADO?

Ответ: Свойство Filter задает критерий фильтрации. Набор данных (ADOTable, ADOQuery) будет отфильтрован по условию, если их свойство Filtered установлено в True. В условиях фильтра можно использовать название атрибутов таблицы, а также логические операции (OR, AND, NOT), но нельзя использовать переменные.

Примеры:

```
Name='Иванов'
Pay>100
```

Нельзя использовать: Pay>y.

❐ Вопрос № 19

Существует SQL-сервер, на нем база данных, доступ осуществляется через компонент ADOQuery, необходимо отсортировать данные в клиентском приложении без использования в запросе секции ORDER.

Ответ:

```
ADOQuery1.Sort:='LastName ASC, DataSale DESC'
```

Здесь будет осуществлена сортировка для поля LastName по возрастанию и для поля DataSale по убыванию.

❐ Вопрос № 20

Имеется база данных MS SQL Server. В ней имеется таблица, в которую надо помещать изображения (фото). В таблице имеется поле типа `Image`. Как поместить картинку в это поле, можно ли это сделать с помощью `ADOQuery`?

Ответ: Необходимо использовать следующий код.

```
var MemoryStream:TMemoryStream;

MemoryStream:=TMemoryStream.Create;
Image1.Picture.Graphic.SaveToStream(MemoryStream);
MemoryStream.Position:=0;
Query1.Close;
Query1.SQL.Clear;
Query1.SQL.Add('insert into фото (вид) ');
Query1.SQL.Add('values :Vid');
Query1.Parameters.ParamByName('Vid').LoadFromStream(MemoryStream,
ftBlob);
Query1.ExecSQL;
MemoryStream.Free;
```

❐ Вопрос № 21

Имеется MS SQL Server плюс приложение, которое его использует. Необходимо посмотреть, какие запросы выполняло приложение на сервере. Есть ли какой-нибудь режим в MS SQL, который все запросы к нему пишет в лог-файл?

Ответ: Есть приложение под названием Profiler.

❐ Вопрос № 22

Хочу разобраться с `BeforeAction`. Осуществляется навигация по БД. Кнопкой `nbPrior` перехожу к началу таблицы. Предпоследнее нажатие: курсор — на первой записи набора данных, кнопка активна; последнее нажатие: курсор, естественно, — на том же месте (куда ему деваться), кнопка становится неактивной. Есть идея осуществить, так сказать, предпросмотр положения курсора и, если он переходит с предпоследней записи на последнюю, сделать кнопку `nbPrior` неактивной. То есть исключить непонятность типа: кнопка `nbPrior` (`nbNext`) показывает, что текущее положение курсора не есть еще начало (конец) набора данных, но при нажатии на эту кнопку курсор никуда не переходит. Как это можно реализовать?

Ответ № 1: Предлагаю такой вариант:

```
procedure TForm1.DBNavigator1Click(Sender: TObject; Button:
TNavigateBtn);
begin
 if (Button=nbPrior) or (Button=nbFirst) then
 begin
  Table1.Prior;

  if Table1.Bof then
   DBNavigator1.VisibleButtons:=[nbNext,nbLast]
  else
   Table1.Next;

 end;

 if (Button=nbNext) or (Button=nbLast) then
 begin
  Table1.Next;

  if Table1.Eof then
   DBNavigator1.VisibleButtons:=[nbFirst,nbPrior]
  else
   Table1.Prior;

 end;
end;
```

Ответ № 2: Можно доработать предложенный вариант таким образом, чтобы кнопки `DBNavigator` не пропадали, а просто становились заблокированными. Для этого вместо:

```
DBNavigator1.VisibleButtons:=[nbFirst,nbPrior]
```

пишем:

```
DBNavigator1.Controls[Ord(nbFirst)].Enabled:=True;
DBNavigator1.Controls[Ord(nbPrior)].Enabled:=True;
```

❐ Вопрос № 23

Подсоединяюсь через `ADOConnection` — курсор серверный, `ADODataSet` — курсор серверный, тип динамический, добавляю `DBGrid`, появляется

ошибка: "Dataset does not support bookmarks? Which are required for multi-record data controls".

Ответ: Предлагаю следующий выход — у `ADODataSet` установить следующие свойства:

```
CursorLocatiojn:=clUseServer;
CursorType:=ctKeyset;
LockType:=ltBatchOptimistic;
```

Таким образом, будет осуществляться пакетная обработка данных, то есть все изменения в пакете либо подтверждаются, либо откатываются.

Теперь в `DBGrid` у вас все отобразится, но вы не сможете видеть сразу изменения, внесенные другими пользователями. Для этого служит оператор `Requery`. Помещаем на форму кнопку **Сохранить изменения** и пишем туда код (после нажатия данные сохранятся и набор данных обновится):

```
try
 ADODataSet1.UpdateBatch();
 ADODataSet1.Requery();
except
 ADODataSet1.CancelBatch();
 ADODataSet1.Requery();
end;
```

❒ Вопрос № 24

Как можно получить отдельно часы, минуты и секунды, а также день, месяц и год?

Ответ № 1: Получение часов, минут, секунд и миллисекунд.

```
procedure TForm1.Button1Click(Sender: TObject);
var
 Hour, Min, Sec, MSec: Word;
begin
 DecodeTime(Time, Hour, Min, Sec, MSec);
 D:=FormatDateTime('dd',Date));
 M:=FormatDateTime('mm',Date));
 Y:=FormatDateTime('yy',Date));
End;
```

Ответ № 2: Получение дня, месяца и года.

```
var
 D,M,Y: string;
begin
 D:=FormatDateTime('dd',Date));
 M:=FormatDateTime('mm',Date));
 Y:=FormatDateTime('yy',Date));
end;
```

❒ Вопрос № 25

Есть базы DBF DOS-кодировки. Можно их перегнать в Windows-кодировку (1251), чтобы текст читался нормально, а не каракулями?

Ответ: Как вариант можно читать из базы, прогонять через функцию `OemToChar()` и записывать изменения в базу.

Заключение

Я надеюсь, что вы нашли для себя в данной книге полезную и интересную информацию. Все ваши пожелания и комментарии рад буду увидеть на форуме **http://www.vr-online** в разделе "Книги от LittleBudda", там же можно всегда со мной пообщаться и задать любой интересующий вас вопрос. Я буду благодарен за любые ваши замечания, их я смогу в дальнейшем использовать в своей работе.

Приложение

Описание компакт-диска

Исходные коды примеров, приведенных в данной книге, можно загрузить с прилагаемого компакт-диска. Также компакт-диск содержит цветные версии рисунков и видеоролики, посвященные темам, рассматриваемым в данной книге. Структура компакт-диска приведена в табл. П1.

Таблица П1. *Структура компакт-диска*

Папка	Содержание
\Глава 3	Содержит исходные коды (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для третьей главы книги
\Глава 4	Содержит исходные коды (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для четвертой главы книги
\Глава 5	Содержит исходные коды (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для пятой главы книги
\Глава 6	Содержит исходные коды (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для шестой главы книги
\Глава 7	Содержит исходные коды (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для седьмой главы книги
\Глава 8	Содержит исходные коды (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для восьмой главы книги
\Глава 9	Содержит исходные коды (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для девятой главы книги

Таблица П1 *(окончание)*

Папка	Содержание
\Глава 10	Содержит модель базы данных (подкаталог SOURCE), цветные версии рисунков (подкаталог PICTURE), видеоролики (подкаталог VIDEO) для десятой главы книги
\Глава 11	Содержит цветные версии рисунков (подкаталог PICTURE) для одиннадцатой главы книги
\Programm\ EhLib	Компоненты для Delphi для работы с БД. Размещена с разрешения Дмитрия Большакова (**http://www.ehlib.com**). Пароль для распаковки версии xUSSR (**http://www.ehlib.com/RUS/ehlibrus.exe**): ФЕВРАЛЬФЕВРАЛЬ
\Programm\ FastReport	Демонстрационные версии генератора отчетов FastReport 2.5 и 3.0, а также документация размещены с разрешения компании FastReport (**http://www.fast-report.com**). Для получения 10% скидки на приобретение лицензии зайдите на страницу **http://www.fastreport.ru/ru/purchase/readers.php** и при оформлении заказа введите пароль — БХВ-СПБ
\Programm\ IBExpert	Инструменты для работы с БД размещены с разрешения компании IBExpert (**www. IBExpert.com**). Trial-версия IBExpert абсолютно полнофункциональна и не имеет никаких временных ограничений, если на компьютере пользователя выбрана национальная настройка "Россия" ("Russian"). Другими словами, для российских пользователей продукт абсолютно бесплатен и триальные ограничения не работают

Все видеоролики, расположенные в подкаталогах Video, упакованы архиватором WinRar. Перечень файлов, доступный после распаковки, приведен в табл. П2.

Таблица П2. *Файлы, доступные после распаковки архивов*

Папка	Архив	Какие файлы будут доступны после распаковки
\Глава 3\ Video	part1.rar	Install_Interbase_6_5.avi (39,5 Мбайтайт) Install_Interbase_7_5.avi (48,1 Мбайт)
	part2.rar	InterBase_Manager.avi (17,8 Мбайт)
	part3.rar	Create_base.avi (269,1 Мбайт)
	part4.rar	Client_D7.avi (1825,3 Мбайт) Client_D2005.avi (610,2 Мбайт)

Таблица П2 *(окончание)*

Папка	Архив	Какие файлы будут доступны после распаковки
\Глава 4\ Video	part1.rar	Install__MSSQL.avi (55,7 Мбайт)
	part2.rar	Create_base.avi (281,4 Мбайт)
	part3.rar	Client_D7.avi (1 196,9 Мбайт) Client_D2005.avi (881,8 Мбайт)
\Глава 5\ Video	part1.rar	Install_Firebird.avi (19,6 Мбайт)
	part2.rar	Create_base.avi (577,1 Мбайт)
	part3.rar	Client_D7.avi (1859 Мбайт) Client_D2005.avi (1363,3 Мбайт)
\Глава 6\ Video	part1.rar	Install_QReport_D7.avi (13,2 Мбайт) Install_QReport_D2005.avi (43,6 Мбайт)
	part2.rar	Client_D7.avi (354,5 Мбайт) Client_D2005.avi (397,4 Мбайт)
\Глава 7\ Video	part1.rar	Client_D7.avi (423,9 Мбайт) Client_D2005.avi (148,3 Мбайт)
\Глава 8\ Video	part1.rar	Install_FastReport_2_5.avi (9,9 Мбайт) Install_FastReport_3_14.avi (17,7 Мбайт)
	part2.rar	Client_D7.avi (252,4 Мбайт) Client_D2005.avi (212 Мбайт)
\Глава 9\ Video	part1.rar	Client_D7.avi (358,1 Мбайт) Client_D2005.avi (471,2 Мбайт)
\Глава 10\ Video	part1.rar	Install_ERwin_40.avi (30,5 Мбайт)
	part2.rar	ERwin_develop_Library.avi (976,1 Мбайт)

Список используемой литературы

1. Хоторн Р. Разработка баз данных Microsoft SQL Server 2000 на примерах. — Издательство "Вильямс", 2001. — 464 с.
2. Кандзюба С. П. Delphi. Базы данных и приложения. Эффективный самоучитель. — Издательство "Диасофт", 2005. — 576 с.
3. Build Reports using QuickReport 3 — QBS Software Ltd.
4. Rave Reports. Borland Edition 5.0. Руководство разработчика. — Nevrona Designs
5. Архангельский А. Я. Программирование в Delphi 6. — Издательство "Бином", 2003. — 1120 с.
6. Фаронов В. В. Программирование баз данных в Delphi 6. Учебный курс. — Издательство "Питер", 2002. — 352 с.
7. Ковязин А. Н., Востриков С. М. Мир InterBase. Архитектура, администрирование и разработка приложения баз данных в InterBase/Firebird/ Yaffil. — Издательство "Питер", 2005. — 496 с.

Предметный указатель

Г

Д

З

И

К

Л

М

Н

О

П

Р

С

Т

У