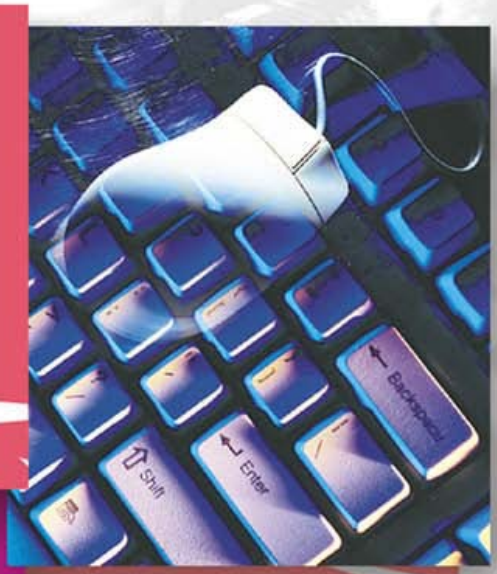
АНДРЕЙ ШКРЫЛЬ

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

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.

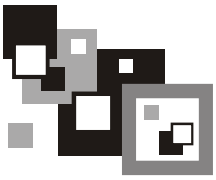
Отпечатано с готовых диапозитивов
в ГУП "Типография "Наука"
199034, Санкт-Петербург, 9 линия, 12

ISBN 5-94157-761-3

© Шкрыль А. А., 2006
© Оформление, издательство "БХВ-Петербург", 2006

Оглавление

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.

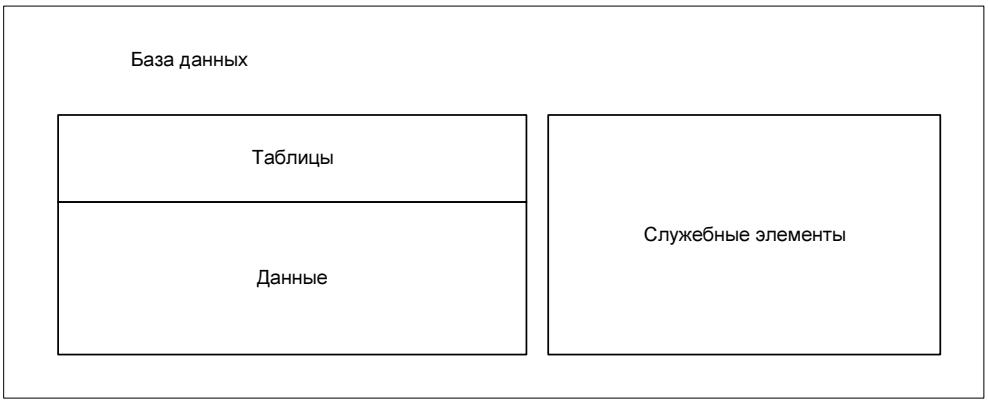


Рис. 1.1. Структура базы данных

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.

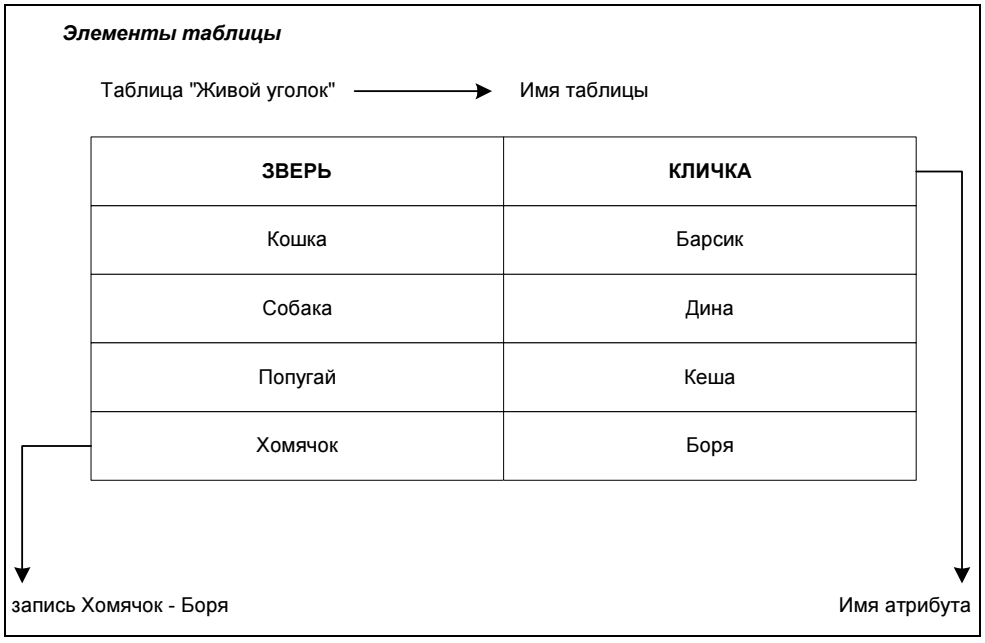


Рис. 1.2. Элементы таблицы

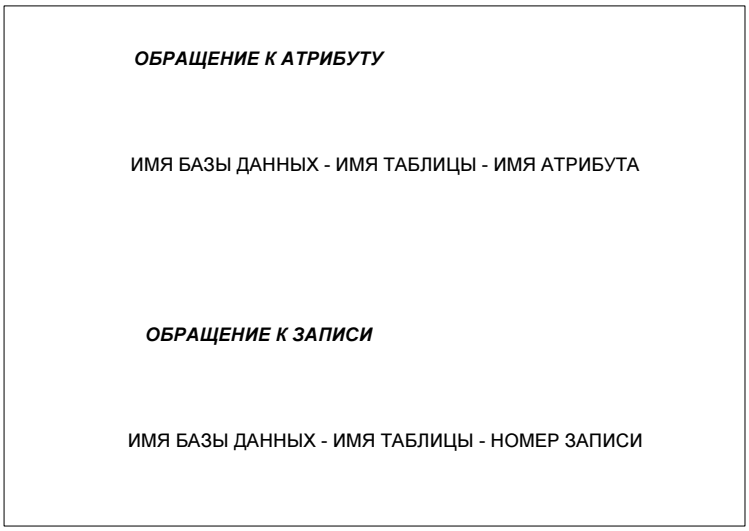


Рис. 1.3. Обращение к элементам таблицы

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*

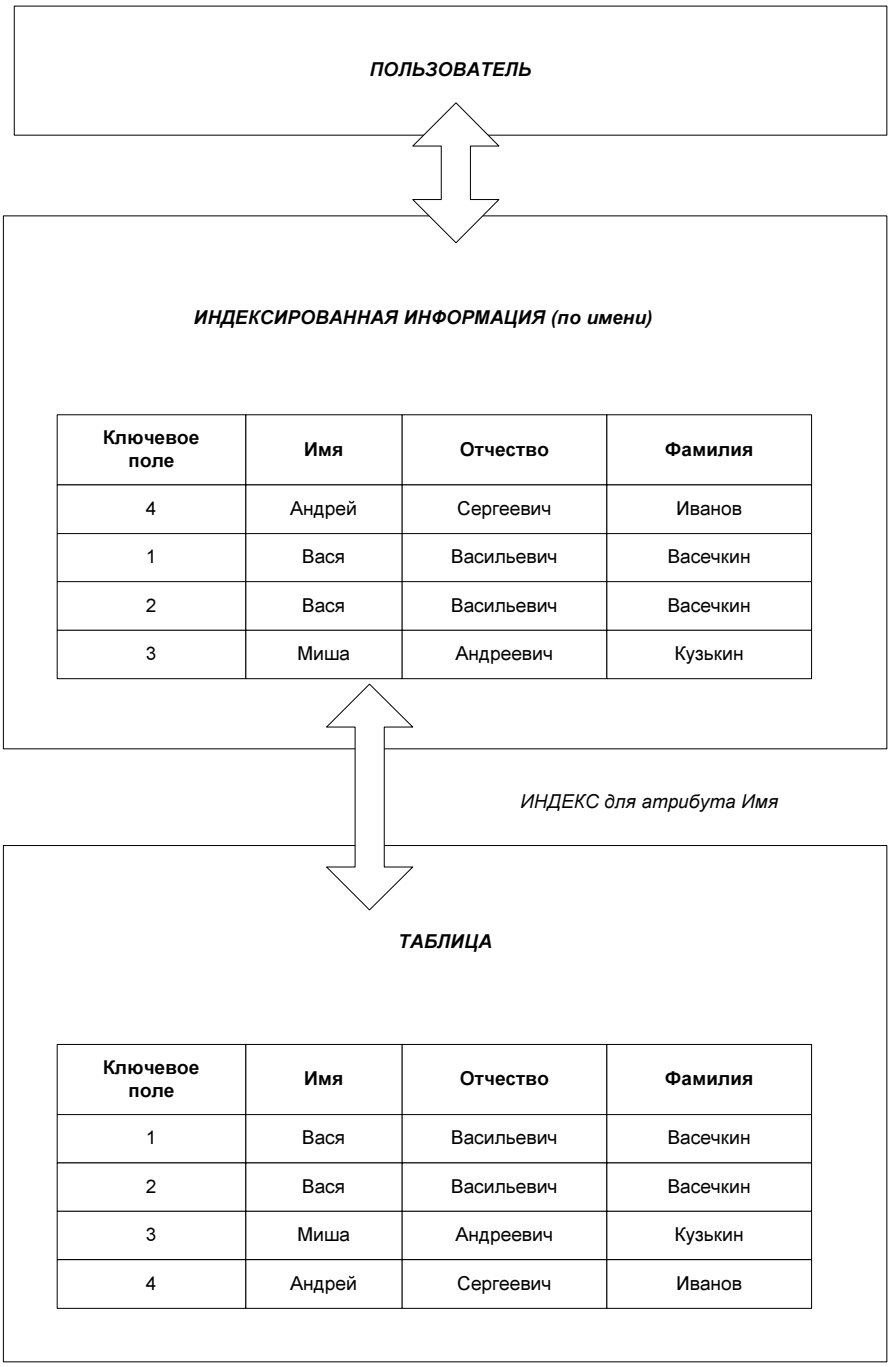
Ключевое поле	Имя	Отчество	Фамилия
1	Вася	Васильевич	Васечкин
2	Вася	Васильевич	Васечкин
3	Миша	Андреевич	Кузькин
4	Андрей	Сергеевич	Иванов

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).

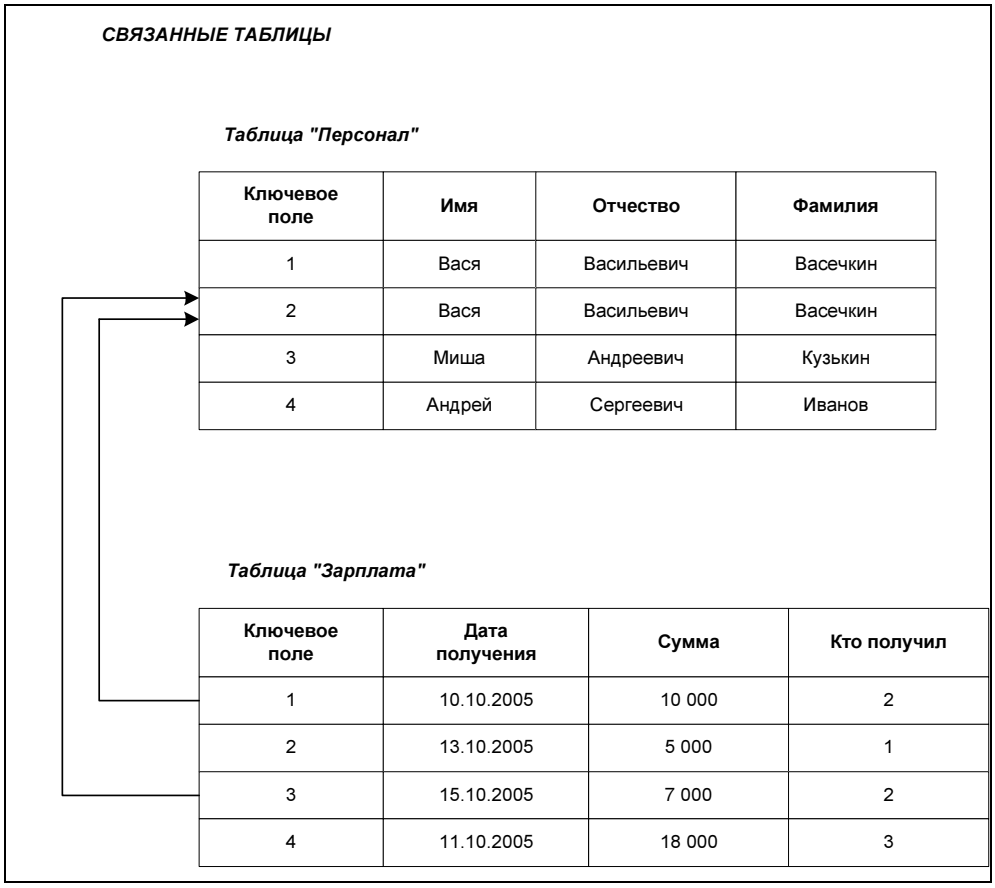


Рис. 1.5. Связанные таблицы

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).

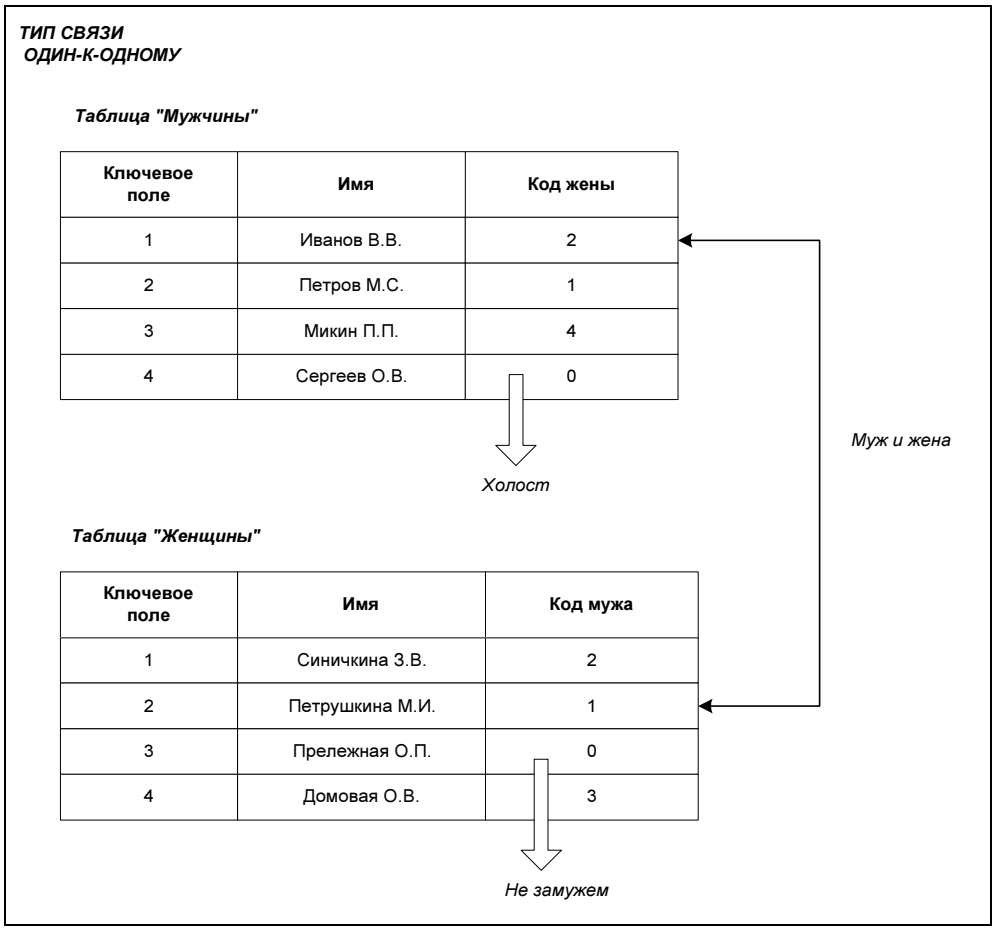


Рис. 1.6. Тип связи один-к-одному

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).

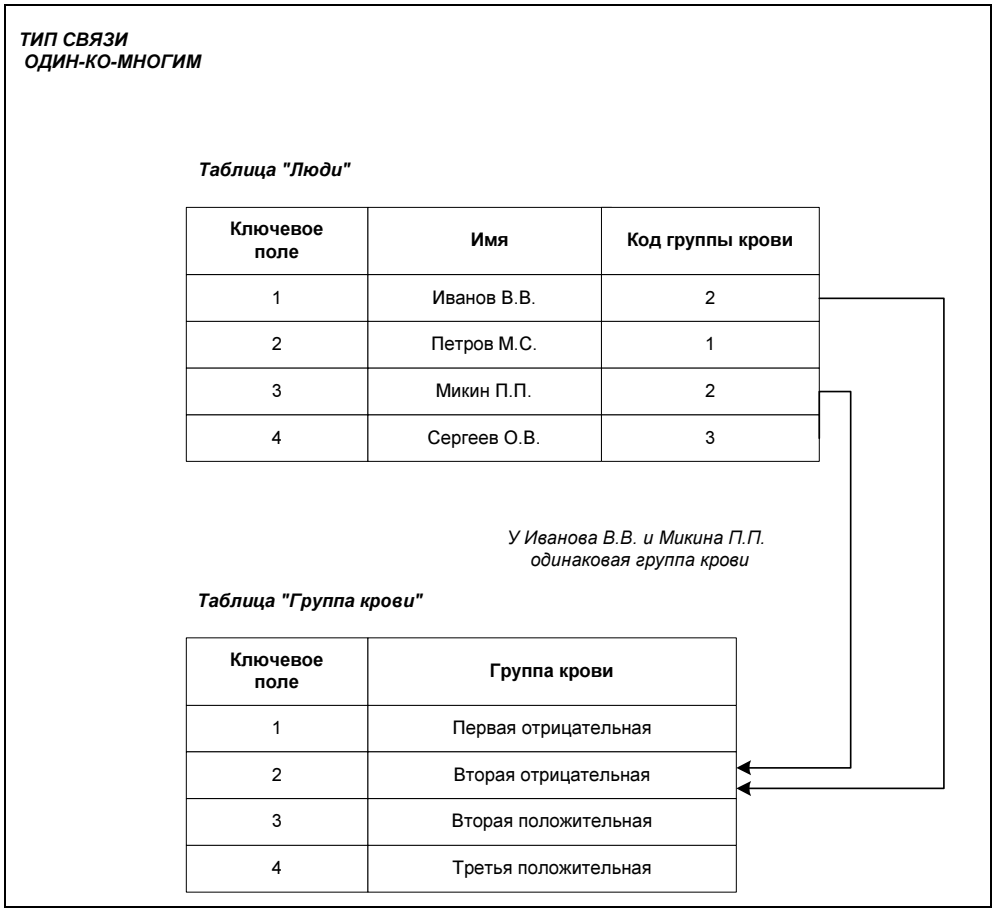


Рис. 1.7. Тип связи один-ко-многим

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).

ТИП СВЯЗИ
МНОГИЕ-КО-МНОГИМ

Таблица "Врачи"

Ключевое поле	Имя	Специальность
1	Бубнов К.М.	Окулист
2	Семенов А.А.	Невропатолог
3	Важный О.Г.	Лор
4	Говорун П.С.	Кардиолог

Таблица "Отношения"

Ключевое поле	Код врача	Код пациента
1	1	4
2	1	2
3	1	3
4	2	3
5	4	1

Таблица "Пациенты"

Ключевое поле	Имя
1	Иванов В.В.
2	Сергеев О.В.
3	Микин П.П.
4	Петров М.С.

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.

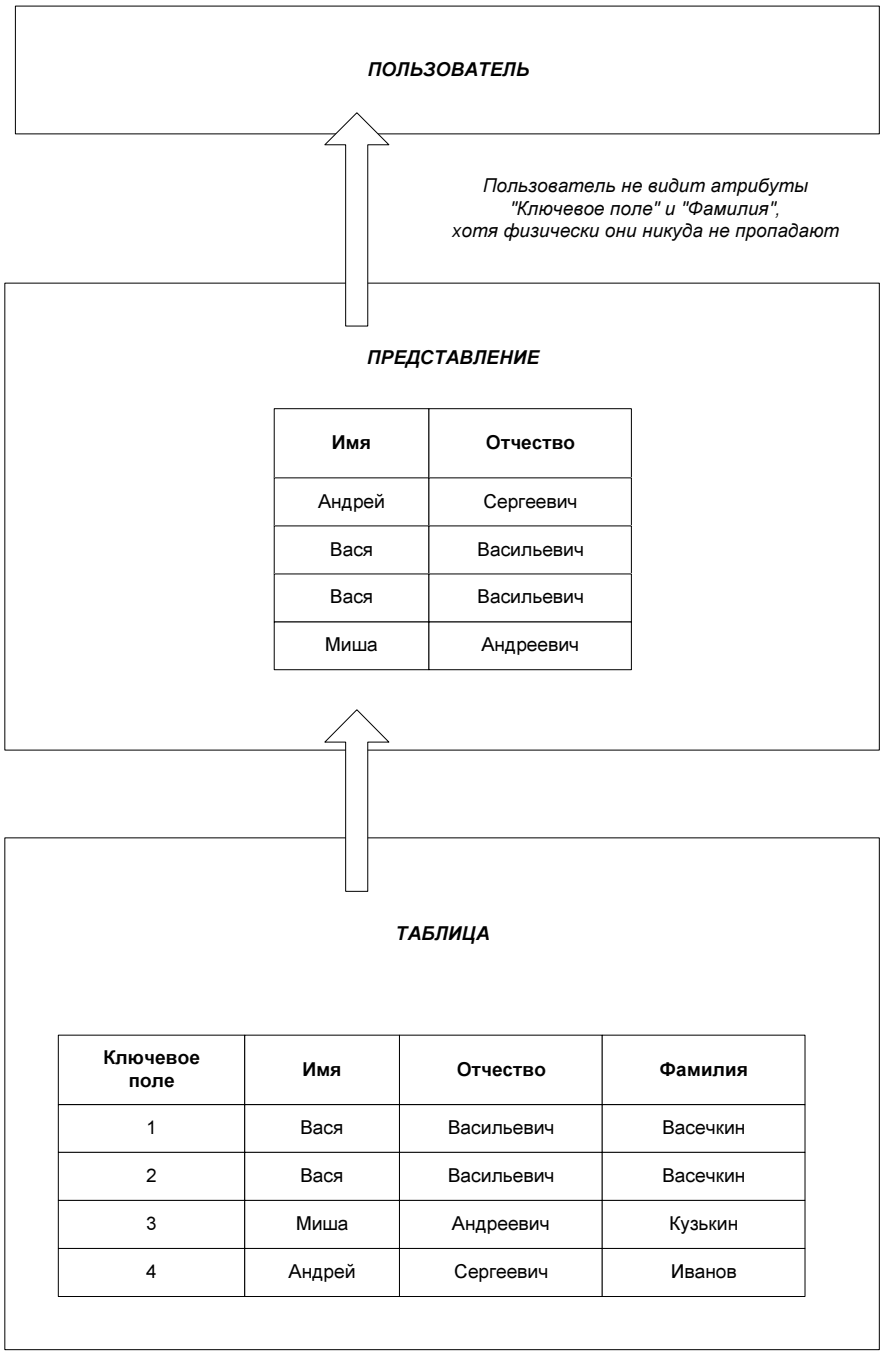


Рис. 1.9. Принцип работы представления

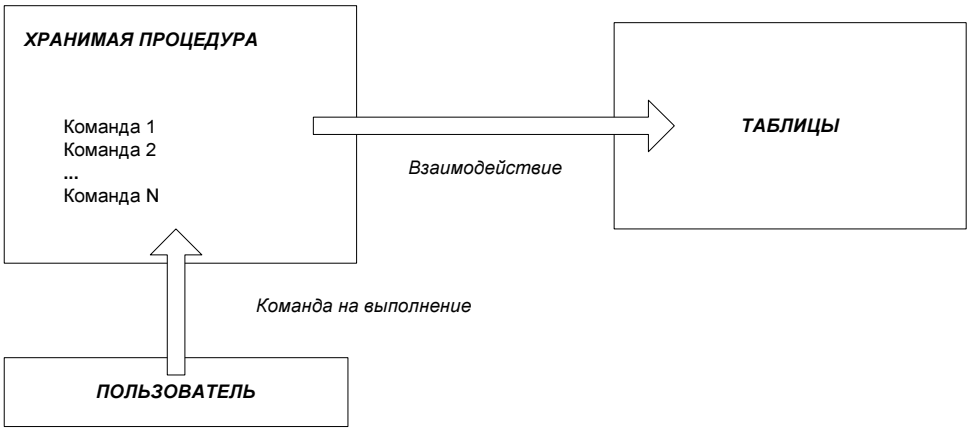


Рис. 1.10. Принцип работы хранимой процедуры

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.

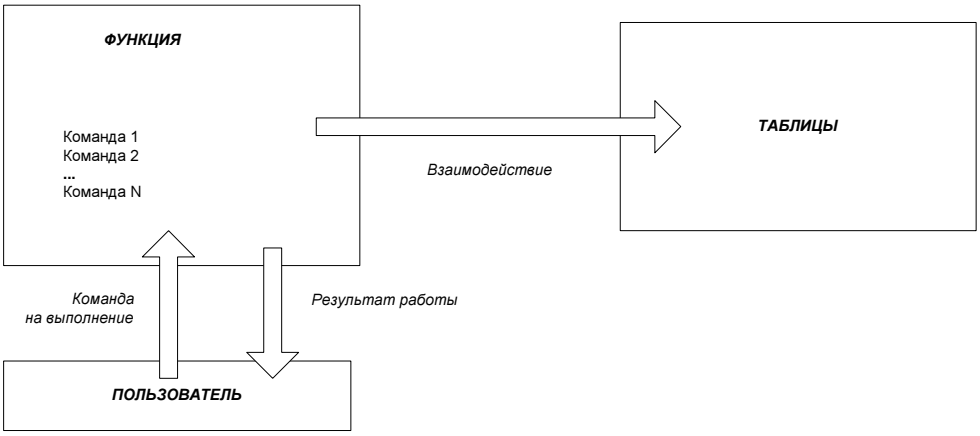


Рис. 1.11. Принцип работы функции

- 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