ДМИТРИЙ ОСИПОВ

bhv®

Базы данных и Delphi

Теория и практика

+ ПРОБНЫЕ ВЕРСИИ ПО

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

- ВВЕДЕНИЕ В ТЕОРИЮ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ
- ПРОЕКТИРОВАНИЕ ЛОКАЛЬНЫХ И КЛИЕНТ-СЕРВЕРНЫХ БАЗ ДАННЫХ
- СТРУКТУРИРОВАННЫЙ ЯЗЫК ЗАПРОСОВ SQL
- РАСШИРЯЕМЫЙ ЯЗЫК РАЗМЕТКИ XML
- ТЕХНОЛОГИИ DBEXPRESS, INTERBASE EXPRESS И ADO
- МНОГОУРОВНЕВЫЕ БАЗЫ ДАННЫХ НА ОСНОВЕ DATASNAP
- НЕСТАНДАРТНЫЕ ПРИЕМЫ РАЗРАБОТКИ ПРИЛОЖЕНИЙ БАЗ ДАННЫХ

УДК 681.3.06
ББК 32.973.26-018.2
О-74

Осипов Д. Л.

О-74 Базы данных и Delphi. Теория и практика. — СПб.: БХВ-Петербург, 2011. — 752 с.: ил. + DVD — (Профессиональное программирование)

ISBN 978-5-9775-0659-5

Книга основана на материалах лекций и практических занятий, разработанных автором, и объединяет теоретические основы и практические аспекты разработки реляционных баз данных. В первой части рассмотрена концепция реляционных баз данных: реляционная модель данных, жизненный цикл информационной системы, концептуальное и логическое моделирование БД, нормализация отношений, обеспечение многопользовательского доступа к данным, вопросы обеспечения безопасности БД, языки SQL и XML и др. Во второй части описаны возможности современных версий Delphi в области разработки приложений баз данных: подробное описание технологий dbExpress, Interbase Express и ADO, особенности использования компонентов управления данными визуальной библиотеки Delphi, механизм разработки многоуровневых приложений на основе технологии DataSnap, порядок создания отчетов для печати и многое другое. На DVD размещены дополнительные главы, а также материалы и пробные версии ПО компании Embarcadero, включая Delphi XE.

Для студентов и программистов

УДК 681.3.06
ББК 32.973.26-018.2

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Игорь Шишигин*
Зав. редакцией	*Григорий Добин*
Редактор	*Леонид Кочин*
Компьютерная верстка	*Натальи Смирновой*
Корректор	*Виктория Пиотровская*
Дизайн серии	*Инны Тачиной*
Оформление обложки	*Елены Беляевой*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 31.01.11.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 60,63.
Тираж 1500 экз. Заказ №
"БХВ-Петербург", 190005, Санкт-Петербург, Измайловский пр., 29.

Санитарно-эпидемиологическое заключение на продукцию № 77.99.60.953.Д.005770.05.09
от 26.05.2009 г. выдано Федеральной службой по надзору
в сфере защиты прав потребителей и благополучия человека.

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

ISBN 978-5-9775-0659-5

Оглавление

МАТЕРИАЛ НА ДИСКЕ

Генератор отчетов своими руками

Диаграммы и графики

ОТЧЕТЫ В MICROSOFT OFFICE

Глава 4

Реляционная модель данных

Создателем реляционной модели считается математик Эдгар Фрэнк Кодд (Edgar Frank Codd, 1923–2003). Однако работа талантливого ученого начиналась не с чистого листа. Научный базис теории отношений, положенной Коддом в основу реляционной модели, закладывался еще на рубеже XIX и XX веков. Авторами основ будущей реляционной теории являются два исследователя: Чарльз Содерс Пирс (1839–1914) и Эрнст Шредер (1841–1902). Несмотря на столь глубокие корни реляционной теории, первая реляционная база данных появилась на свет спустя семьдесят лет. Датой рождения реляционной БД можно считать июнь 1970 года. Именно тогда Кодд (на тот момент времени сотрудник одной из лабораторий корпорации IBM) опубликовал свою знаменитую статью "Реляционная модель данных для больших совместно используемых банков данных" (Codd E. F., A Relational Model of Data for Large Shared Data Banks. CACM 13: 6), в которой впервые прозвучал столь популярный сегодня термин "реляционная модель".

Замечание

За свой вклад в науку Эдгар Кодд был удостоен премии имени Тьюринга. Это знаменательное событие произошло в 1981 году.

Первопричиной возникновения нового по тем временам подхода к проектированию баз данных послужили существенные ограничения предыдущих моделей. Ни сетевая, ни иерархическая модели не были способны просто и доступно описывать подлежащие учету данные. Кодд сумел объединить, на первый взгляд, несовместимые вещи — с одной стороны, реляционная модель опиралась на математические выкладки, а с другой стороны, была понятна рядовому пользователю, состоящему в конфронтации даже с таблицей умножения.

Замечание

Модель данных представляет собой логически взаимоувязанную совокупность описаний объектов данных и операторов, управляющих этими объектами. Применяя термин "модель", мы акцентируем внимание на то, что это абстрактная конструкция. Чтобы модель превратилась в действующую базу данных, она должна быть физически реализована средствами какой-либо СУБД. Нетрудно догадаться, что реляционная база данных может получиться только из реляционной модели.

Модель данных не представляет существенного интереса для обычного пользователя, но без нее не обойтись разработчику БД. Ведь это не что иное, как чертеж будущей БД. Ни один строитель не начнет строить дом без чертежа, так и ни один программист не начнет создание физической БД без ее подробной схемы. Фундамент реляционной модели построен на понятии сущности и связей между сущностями.

Реляционной модели данных посвящено много фундаментальных трудов, в которых подробно изложены все ключевые аспекты. Обязательно рекомендую читателю хотя бы в обзорном порядке ознакомиться с работами ведущих специалистов в этой области [7, 11, 17]. Моя задача несколько прозаичнее, далее я попытаюсь сформулировать тот минимум информации, без которого понять реляционную модель будет просто невозможно.

Любая область знаний, будь то математика, радиоэлектроника, физика, химия или информатика по мере своего развития формирует свой набор терминов и определений. Без этого никак не обойтись, ведь в своих ученых беседах специалисты оперируют понятиями, не имеющими аналогов в окружающем мире. Зачастую размерность специфичных понятий разрастается до такой степени, что существенная часть времени отводимого на изучение интересующего нас предмета затрачивается только на формирование словарного запаса. Это утверждение подтвердят студенты, выбравшие в качестве своей профессии медицинскую стезю. Вы никогда не присутствовали при общении двух эскулапов? Даже если они говорят о банальном насморке, у стороннего наблюдателя неизбежно создается впечатление, что перед ним два иностранца с неподдающейся идентификации языковой принадлежностью.

С точки зрения сложности семантики, системы управления базами данных (в общем) и реляционная модель (в частности) не являются исключением из правил. Эта сравнительно новая область знаний практически мгновенно успела обзавестись специфичным набором слов. Многие термины перекочевали в БД из словарного запаса программистов, часть определений позаимствована из теории множеств, в тезаурусе разработчиков моделей данных даже нашлось место для понятий, обычно встречающихся в справочниках по философии.

Сущность и атрибуты

Любой более-менее подготовленный пользователь твердо знает, что базы данных предназначены для хранения и обработки информации. Если в руки такому человеку попадется база городских телефонов, то, даже не запуская программу, он сможет предположить, что в ней он обнаружит телефонные номера граждан и организаций. База данных городской библиотеки обязана содержать информацию об авторах и их бессмертных произведениях. Столкнувшись с базой данных авиакомпании с высокой степенью вероятности, можно утверждать, что в ней мы найдем информацию о расписании полетов, марках самолетов и городах, в которые они рассчитывают доставить пассажиров.

То, что кажется элементарным для стороннего наблюдателя, для профессионального разработчика БД может стать очень непростым делом. На самом первом этапе проектирования будущей БД он ломает голову над двумя ключевыми вопросами:

1. Что необходимо хранить в БД?
2. Какие связи между данными следует поддерживать в БД?

Для получения ответа на первый вопрос разработчик должен составить первоначальный перечень типов сущностей. *Тип сущности* (entity) — это класс однотипных объектов, о которых следует хранить информацию в БД. Немного забегая вперед, проговорюсь, что на заключительной стадии проектирования типы сущности превратятся в реляционные таблицы. В 90% случаев сущность представляет собой какой-то объект, отвечающий на вопрос "Кто?" или "Что?". В базе данных библиотеки на роль сущностей претендуют такие одушевленные и неодушевленные объекты, как автор, книга, издательство, читатель. Совсем не обязательно, чтобы сущность соответствовала объектам реального мира, напротив, она может моделировать и отвлеченные понятия, допустим, выраженные в особых отношениях между сущностями. Именно в таком случае сущность ловко маскируется под глагол. Как правило, это происходит, когда учету подлежат некоторые события. Например, когда читатель получает (это и есть глагол) книгу из библиотечного фонда, логика базы данных требует сохранить информацию о том, кому, когда и какая из книг была выдана. При более подробном рассмотрении выясняется, что при построении первоначального списка мы просто не выявили такой тип сущности, как читательская библиотечная карточка, в которую и делаются все перечисленные пометки. Библиотечная карточка поддерживает двустороннюю связь между сущностями "читатель" и "книга".

Если тип сущности — это класс однотипных объектов, то как же величать отдельный объект, входящий в этот класс? В реляционной модели такой объект называют *экземпляром сущности*. Экземпляр сущности (или просто сущность) — это конкретная реализация объекта. Для сущности "автор" это Уильям Шекспир и Лев Толстой, для сущности "книга" это "Ромео и Джульетта" и "Война и мир".

Замечание

Экземпляр сущности — это то, о чем (или о ком) следует хранить информацию в базе данных.

У любой сущности есть свои характеристики. У сущности "самолет" из базы данных авиакомпании таких характеристик миллион. Это марка, максимальная скорость полета, число посадочных мест, грузоподъемность, дата выпуска, цвет фюзеляжа и многое-многое другое. У сущности "читатель" отличительных признаков не меньше, а может быть и больше, чем у летательного аппарата. В счет может пойти все, начиная с цвета глаз и заканчивая размером обуви. Однако в библиотечной базе данных вряд ли стоит учитывать рост, вес и отпечатки пальцев клиента. Гораздо полезнее знать имя, домашний адрес и номер телефона. Отличительные свойства сущности называют *атрибутами*.

Атрибуты могут быть *простыми* (например, название города, в который летит самолет), а может быть и *составными* (т. е. включать в себя несколько простых атрибутов). В качестве составного атрибута выступает адрес читателя. Здесь найдется место почтовому индексу, городу, улице, дому и номеру квартиры. Атрибут может быть *производным*, т. е. полученным в результате проведения какой-то операции над другими атрибутами. Подобные атрибуты часто встречаются при построении различных бухгалтерских систем, например, значение взимаемого с сотрудников предприятия налога составляет определенный процент от заработной платы этих служащих. Можно столкнуться с *многозначными* атрибутами, они описывают те свойства сущности, в которых одновременно хранится несколько значений. Например, у одного человека может быть несколько адресов проживания, несколько контактных телефонных номеров и т. п. Есть и особая разновидность атрибутов, без которых невозможно построение реляционной модели, — атрибуты, однозначно идентифицирующие экземпляр сущности. На роль такого атрибута в состоянии претендовать индивидуальный номер налогоплательщика или уникальный заводской номер самолета. Позднее подобные атрибуты превратятся в ключевые поля реляционных таблиц.

Построив исчерпывающий список типов сущностей и подготовив перечень их атрибутов, разработчик модели базы данных должен задуматься над особенностями реализации типов сущностей. Для этого в первую очередь следует разобраться с особенностями физического представления каждого из атрибутов типа сущности, для этого предназначены тип данных и домен.

Тип данных и домен

Отправной точкой процесса построения любой системы хранения и обработки данных по праву считается выбор *типа данных*. Независимо от того, какой язык программирования вы изучали, при построении новой программы самым первым шагом становится рассмотрение типов данных, имеющихся в распоряжении системы. Программистов Delphi, Си или поклонников любого другого языка программирования не удивишь типами данных `INTEGER`, `REAL` или `CHAR`. Указанные типы данных специализируются соответственно на обслуживании целочисленных, вещественных и символьных значений.

Типы данных определяют порядок хранения данных в памяти компьютера. Физическая концепция хранения данных зависит от особенностей конкретной платформы, на базе которой функционирует программное обеспечение. Поэтому в разных операционных системах, в разных системах управления базами данных число байтов, отводимых для описания целого или вещественного числа, может отличаться. Например, в 32-разрядных ОС размерность целого числа `INTEGER` равняется 4 байтам. В 64-разрядных системах это утверждение может оказаться ошибочным, ведь здесь проще адресовать 8-байтовую переменную.

Типизация хранимых значений не просто указывает на размерность в байтах, которую должна выделить система для размещения в памяти того или иного значения. Преследуемая цель еще более значима. Типизация определяет, какие операции могут быть осуществлены с теми или иными данными. Система программирования не позволит новичку передать результаты деления в целочисленную переменную, ведь в этом случае есть риск утерять дробную часть результата. Система станет отчаянно сопротивляться, если мы попробуем просуммировать символьный и вещественный тип данных. Пусть даже в символьной переменной хранится числовое значение. В последнем случае перед проведением математической операции необходимо повести преобразование данных. Точно так же СУБД не допустит ввода текстовых данных в поля таблиц, предназначенных для хранения числовой информации.

Подытожим все сказанное. Понятие "тип данных" интегрирует в себе три компонента:

1. Ограничение множества значений, принадлежащих типу.
2. Дефиниция набора операций, применяемых к типу.
3. Определение способа отображения (внешнего представления) значений типа.

Как показала практика, при обработке данных в современных информационных системах возможностей обычной типизации становится недостаточно. Примеров тому великое множество. Формально, при определении атрибута, хранящего дату рождения сотрудников какого-нибудь предприятия, следует ограничиться типом данных `DATETIME`. Но такой подход не способен в полной мере отразить задачи отдела кадров. Стандартизированный в SQL тип данных `DATETIME` допускает ввод даты в диапазоне от 1/01/0000 г. до 12/31/9999 г. Вряд ли начальник нынешнего отдела кадров примет на работу сотрудника в несовершеннолетнем возрасте или, наоборот, ровесника Ивана Грозного. Поэтому атрибут "дата_рождения" должен уметь сопротивляться данным, некорректным, с точки зрения пользователя, но абсолютно верным с точки зрения типа данных `DATETIME`. Есть и другой пример. Допустим, что в базе данных хранится почтовый индекс. Каждый поклонник эпистолярного жанра знает, что это комбинация из шести цифр, отражающая региональную принадлежность почтового отделения. Порядок описания индекса не сложен для человека — нужно заполнить все шесть значений, не четыре, не пять и не семь, а только шесть. Теперь попробуйте подыскать стандартный тип данных, способный однозначно решить эту простейшую задачу. Тип `INTEGER` не подходит, потому что обладает существенно бóльшим диапазоном значений. Строка из шести элементов `CHAR` также не вполне адекватна, ведь она позволяет пользователю вводить любые (не только цифровые) символы. Рассмотрим еще один пример. Формально номер телефона можно типизировать как целое число — `INTEGER`. Целые числа поддерживают все основные арифметические операции. Однако никому в здравом уме и в доброй памяти не придет в голову идея суммировать значения телефонных номеров или вычитать из одного номера другой, хотя формально это обычные целочисленные значения. Просто очень сомнительна практическая ценность полученного результата. В подобных и во многих других схожих ситуациях одна лишь типизация бессильна — она в недостаточной степени поддерживает че-

ловеческую логику. Если я вас убедил, что возможностей типа данных недостаточно для описания характеристик атрибутов, закончим с примерами и поговорим о том, как это ограничение можно преодолеть.

Фундаментальная для современных языков программирования идея типизации в реляционной модели получила дальнейшее развитие. На сцену вышло новое понятие — *домен*. Домен ни в коем случае не подменяет понятие типизации, а выступает в качестве дополнительного ограничителя информации, обрабатываемой в базе данных. Ограничение накладывается за счет введения дополнительных логических правил. Так, принимая нового сотрудника на работу, при вводе даты рождения можно контролировать его возраст, путем элементарного сравнения текущей системной даты компьютера и даты рождения. Обслуживая атрибут почтового индекса, описываемого шестью символами типа `CHAR`, наложим дополнительное ограничение на диапазон приемлемых значений — наш домен должен пропускать только цифровые знаки. Надо сразу оговориться, что логические правила являются не столько описанием отдельного домена, сколько средством поддержания целостности данных в масштабах всей базы данных. Допустим, вы вводите почтовый индекс города Москвы, а в атрибут "город" передаете Монреаль. В этом случае разумная база данных должна намекнуть нам, что мы сегодня перетрудились и следует прерваться на чашечку кофе… Но для того, чтобы система смогла выявить подобную логическую ошибку, разработчик БД должен построить не только строгий набор доменных ограничений, но и реализовать развитую систему поддержания корпоративной целостности.

Замечание

Тип данных определяет особенности физического хранения данных. Домен, опираясь на концепцию типа данных, несет дополнительную нагрузку — отвечает за поддержание логических правил описания данных. Благодаря доменным ограничениям в БД реализуется одно очень важное качество — доменная целостность данных.

В профессиональных СУБД доменные ограничения могут накладываться на несколько атрибутов одновременно, таким образом, одновременно контролируя непротиворечивость хранящихся в них данных.

Связь

Реляционная модель призвана не только описать перечень типов сущностей, но и отразить особенности взаимодействия между ними. Взаимодействие возникает там, где между типами сущностей проявляются отношения, выраженные в виде глагола, например, продавец оформил заказ. Здесь "продавец" и "заказ" — это две сущности, а связь между ними просматривается в глаголе "оформил".

Различают три типа связи между сущностями: "один к одному", "один ко многим" и "многие ко многим". Появление в реляционной БД связи "один к одному" — крайне редкий случай. Чаще всего он свидетельствует о том, что разработчик не

вполне корректно определился с атрибутами и вместо одного из атрибутов создал лишний тип сущности. На практике наиболее распространена связь "один ко многим", например, автор написал много книг, завод выпустил много автомобилей, в одном отделе работает много сотрудников (рис. 4.1).

"Сотрудник"		
Ключ_сотрудник	Фамилия	Ключ_отдела
1	Григорьев	2
2	Арбузов	2
3	Самойлов	1
4	Костюков	3
5	Симонова	1

"Отделы предприятия"	
Ключ_отдела	Название
1	Отдел кадров
2	Бухгалтерия
3	Отдел планирования

Рис. 4.1. Отношение "один ко многим"

Связь "многие ко многим" встречается реже: офисы разных компаний размещены в разных городах, множество издательств опубликовало произведения различных авторов и т. п. Реляционная модель напрямую не поддерживает отношение "многие ко многим", поэтому его приходится разбивать на два отношения "один ко многим", но об этом поговорим позднее.

Есть еще один классификационный признак связи — *степень* (размерность). При выявлении связей между сущностями разработчик обычно встречается с *унарными* (unary relationship) и *двойными связями* (binary relationship), но в реальной жизни имеют место связи и большей кратности. Например, выражение "авиакомпания доставляет пассажира в город" подразумевает *тернарную связь* (ternary relationship). Здесь мы сталкиваемся с тремя существительными-сущностями "авиакомпания", "пассажир" и "город" и всего одним глаголом "доставляет". Дальнейшее развитие модели такой базы данных во многом зависит от опыта и интуиции разработчика, ведь разные пассажиры вправе летать в различные города, пользуясь услугами разных авиакомпаний. Ограничившись лишь двойными связями, можно утратить суть сохраняемой информации — какая из авиакомпаний, в какой именно из городов доставила конкретного пассажира. К вопросу построения связей класса "многие ко многим" мы вернемся в *главе 5* при описании процесса ER-моделирования, а пока обсудим еще одну особенность связи.

Реляционная модель позволяет связывать сущность саму с собой — задавать *рекурсивную связь*. Такой, на первый взгляд, несколько необычный способ взаимодействия оказывается весьма полезным при "выращивании" древообразных конструкций, например, описывающих организационную структуру предприятия, завода или, скажем, университета. Здесь четко просматривается иерархия, в которой одна и та же сущность может быть главной по отношению к подчиненным элементам и одновременно находиться в подчиненном состоянии к более высокому по рангу узлу. Так, дочерними элементами по отношению к деканату факультета выступают кафедры этого факультета, в свою очередь деканат подчиняется ректорату учебного заведения.

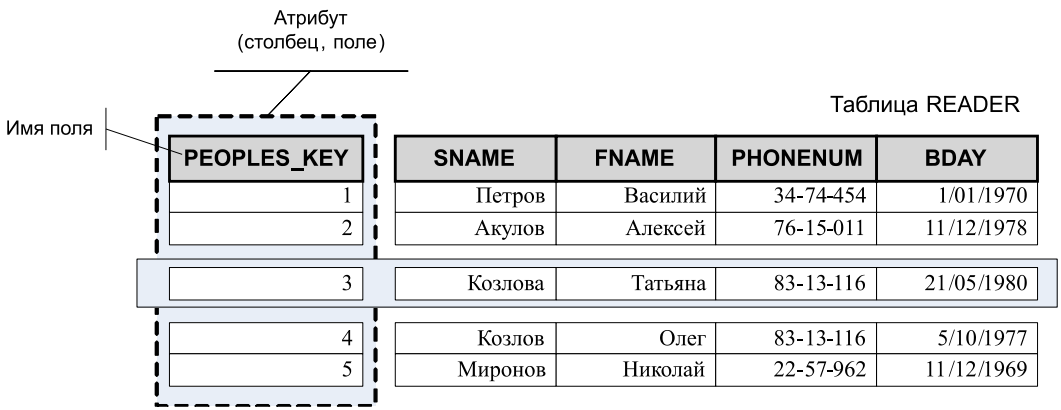
Реляционная таблица

Пользователь рассматривает всю информацию в реляционной БД как совокупность взаимосвязанных двумерных таблиц (отношений), в каждой из которых хранятся данные о строго определенном типе сущностей.

Реляционная таблица состоит из строк и столбцов. Каждая строка содержит информацию о конкретной сущности, так в таблице "Авторы" из библиотечной базы данных мы увидим строки с данными о А. С. Пушкине, М. Ю. Лермонтове, Ф. М. Достоевском и других известных (и не очень) поэтах и писателях. Но в таблице "Авторы" не должно оказаться ни единой строки о клиентах библиотеки — читателях, ведь это другой тип сущности. Каждый столбец реляционной таблицы предназначен для хранения определенного атрибута сущности. Для атрибута "Фамилия" разработчик таблицы применит текстовый тип данных размерностью в 15–20 символов. А для атрибута "Номер телефона" (которому найдется место в таблице читателей) будут наложены дополнительные доменные ограничения, запрещающие передавать в столбец нецифровые символы.

Замечание

Терминология реляционных баз данных складывалась из нескольких областей знаний, поэтому в качестве синонимов к понятию "строка" можно услышать слова "ряд" и "кортеж". У термина "столбец" также есть два тождественных понятия: "атрибут" и "поле". Ведя речь о таблице, допустимо применять термин "отношение" (рис. 4.2).



PEOPLES_KEY	SNAME	FNAME	PHONENUM	BDAY
1	Петров	Василий	34-74-454	1/01/1970
2	Акулов	Алексей	76-15-011	11/12/1978
3	Козлова	Татьяна	83-13-116	21/05/1980
4	Козлов	Олег	83-13-116	5/10/1977
5	Миронов	Николай	22-57-962	11/12/1969

Рис. 4.2. Реляционная таблица

Для того чтобы таблица (отношение) получила высокую честь называться "реляционной", необходимо ее соответствие ряду строгих требований:

- уникальность имен столбцов внутри таблицы;
- каждая строка хранит информацию об отдельной сущности, строки таблицы отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку такой таблицы;