ГАЛИНА МИРОШНИЧЕНКО

bhv

РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ:

ПРАКТИЧЕСКИЕ ПРИЕМЫ ОПТИМАЛЬНЫХ РЕШЕНИЙ

CASE-СРЕДСТВА – МОДЕЛИРОВАНИЕ РЕЛЯЦИОННЫХ ДАННЫХ

НОРМАЛИЗАЦИЯ БАЗЫ ДАННЫХ – РАЗУМНЫЙ УРОВЕНЬ

ИНДЕКСЫ В БАЗЕ ДАННЫХ – ПОЛЬЗА И ВРЕД

ОПТИМИЗАТОР ЗАПРОСОВ SQL-СЕРВЕРА – ПОМОГАЕТ ИЛИ ВВОДИТ В ЗАБЛУЖДЕНИЕ

ПРОИЗВОДИТЕЛЬНОСТЬ ВЫПОЛНЕНИЯ ТРАНЗАКЦИЙ – О НЕЙ НАДО ПОМНИТЬ ВСЕГДА

PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ

+CD

Галина Мирошниченко

РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ:

ПРАКТИЧЕСКИЕ ПРИЕМЫ ОПТИМАЛЬНЫХ РЕШЕНИЙ

Санкт-Петербург
«БХВ-Петербург»
2005

УДК 681.3.06
ББК 32.973.26
М64

Мирошниченко Г. А.

М64 Реляционные базы данных: практические приемы оптимальных решений. — СПб.: БХВ-Петербург, 2005. — 400 с.: ил.

ISBN 5-94157-551-3

Рассмотрены практические вопросы увеличения производительности функционирования клиент-серверных приложений путем использования различных технических приемов на этапах проектирования, разработки и сопровождения реляционной базы данных. Описаны программные средства и преимущества использования CASE-технологий при проектировании БД. Изложены вопросы логического и физического моделирования. Подробно рассмотрено использование индексов. Большое внимание уделено вопросам программирования: понятию об оптимизаторе и плане выполнения запросов; измерению времени выполнения запросов; использованию хранимых процедур, триггеров и многому другому. Приведены рекомендации по разработке транзакций, мониторингу активности пользователя и выявлению критических мест программы. Книга сопровождается многочисленными примерами. При этом на стадии проектирования используется CASE-средство Erwin, а на стадии реализации — MS SQL Server 2000.

CD содержит рассмотренные в книге примеры моделей, учебную базу данных, а также текстовые файлы с листингами программ на T-SQL.

Для программистов, студентов, бизнес-аналитиков и администраторов, работающих с клиент-серверными технологиями

УДК 681.3.06
ББК 32.973.26

Группа подготовки издания:

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

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 27.06.05.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 32,25.
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 5-94157-551-3

Оглавление

Введение

На сегодняшний день реляционные базы данных уверенно являются неотъемлемой частью большинства информационных систем. В этой области прикладного программирования стремительно развиваются новые технологии, платформы реализации и среды разработки приложений. Но тем не менее остается классическая часть и общепризнанный подход к процессу проектирования, разработки и сопровождения реляционной базы данных, разбитый на последовательные этапы, которые обычно приходится реализовывать.

С проблемой производительности так или иначе сталкиваются все разработчики программного обеспечения. Конечно, хорошо, если выбранная целевая система управления базами данных (СУБД) вместе с аппаратными средствами ее реализации предоставляет существенный запас по критерию производительности для конкретной прикладной задачи, так что над этими вопросами можно особенно не задумываться. Однако очевидным правилом является следующее: чем выше этот запас, тем сама СУБД вместе с ее аппаратной конфигурацией становится все более дорогостоящей. Поэтому работа на предельных нагрузках по различным параметрам производительности вполне возможна, и этими вопросами с неизбежностью приходится заниматься.

Основная идея данной книги состоит в том, чтобы рассмотреть решения, принимаемые на всех этапах процесса разработки клиент-серверного приложения с оперативной обработкой транзакций, иначе называемого OLTP-приложением (On-Line Transaction Processing) реляционной базы данных, расположенной на одном сервере, с точки зрения того, к какому уровню производительности эти решения приведут. Для каждого из этих этапов предлагаются стандартные приемы, позволяющие оптимизировать те или иные критерии, основным из которых является производительность информационной системы. Выбор предлагаемых решений объясняется, в необходимых случаях выполняется сравнение альтернативных вариантов, рассматриваются их достоинства и недостатки.

Проблемы, рассматриваемые в книге, актуальны для всех реляционных СУБД, реализующих тот или иной диалект структурного языка запросов SQL (Structured Query Language), и решаются в большинстве случаев аналогично. В нашем случае в качестве среды реализации выбрана СУБД под названием MS SQL-сервер 2000, который является одной из наиболее популярных клиент-серверных СУБД, обладающей развитыми средствами настройки оптимального функционирования приложения. Данная СУБД интенсивно развивается, анонсирована ее новая версия под названием Yucon, которую Microsoft планирует выпустить в самое ближайшее время. Однако изложение материала преследует, в первую очередь, задачу демонстрации концепций, общих для всех реляционных клиент-серверных СУБД, — ведь проблемы всюду одни и те же.

Книга построена достаточно просто. По порядку рассматривается процесс разработки клиент-серверного OLTP-приложения реляционной базы данных, разбитый на общепринятые этапы. Для каждого из этих этапов приводятся и анализируются рекомендации, выполнение которых, в конечном счете, может оказаться полезным для повышения производительности приложения. В качестве примера, проходящего через всю книгу, рассматривается простейшая база данных, предназначенная для классической задачи материального учета.

Глава 1 посвящена проектированию реляционной базы данных и построению ее информационных моделей. В этой главе достаточно подробно изложена общая теория реляционной модели данных, приведено и описано несколько ее наиболее распространенных графических нотаций, пригодных для всех фаз семантического моделирования, — от концептуальной до физической. Далее разбирается достаточно большое количество примеров, связанных с созданием логической модели реляционной базы данных. Для одного и того же описания предметной области обычно приводится несколько вариантов корректной модели. Эти варианты анализируются, объясняется, к каким преимуществам и проблемам на последующих стадиях разработки базы данных приведет тот или иной вариант проектного решения.

Для изображения моделей выбраны стандартизованные нотации IDEF1X (Integration DEFinition for Information Modeling, интегрированная среда определений для информационного моделирования) и IE (Information Engineering, информационная инженерия), предлагаемые CASE-средством (Computer-Aided Software Engineering, автоматизированное проектирование и создание программ) Erwin. Поэтому в *главе 1* книги даются минимальные сведения по работе с этой программой, которых хватает только для того, чтобы научиться изображать и понимать нужную модель. Конечно, для полноценной и эффективной работы с Erwin этого совершенно не достаточно — более подробно ознакомиться с возможностями этого популярного CASE-средства можно в работах [14] и [15] *(см. список литературы)*.

В *главе 2* рассматривается применение основного приема оптимизации проектного решения для OLTP-приложения реляционной базы данных, основанного на таком подходе, как нормализация.

Если материал *глав 1 и 2* в минимальной степени привязан собственно к SQL-серверу, то в *главе 3* рассматриваются вопросы, связанные с физической реализацией проектного решения в контексте выбранной целевой СУБД, которой в нашем случае является SQL-сервер. Здесь, кроме общих рекомендаций по реализации проектного решения в рамках файловой системы, собраны воедино рекомендации по настройкам SQL-сервера, влияющим на те или иные аспекты производительности.

Глава 4 посвящена такому объекту, который имеется в любой реляционной СУБД, как индексы. Индексам уделено особое внимание, потому что именно они являются основным средством, правильное использование которого позволяет существенно повысить быстродействие выполнения главных для базы данных реляционных операций. Описывается механизм физической реализации индексов разных типов и суть основанной на них оптимизации запросов. Рассматриваются команды создания индексов для SQL-сервера, разбираются параметры этих команд, объясняется, в какой ситуации какие значения этих параметров наиболее предпочтительны.

Глава 5 посвящена собственно программированию на SQL-сервере. В связи с этим заметим, что данная книга не является учебным пособием по SQL,-серверу в том смысле, что в ней не описывается базовый синтаксис языка T-SQL (Transact-SQL), являющегося языком программирования для MS SQL-сервера. Для того чтобы в полной мере разобраться в материале *глав 5 и 6*, желательно знание языка T-SQL, для чего можно порекомендовать работы [16], [17] и [18].

Основное внимание в *главе 5* уделяется оптимизации выполнения запросов на выборку данных, т. е. выполнению такого стандартного оператора языка SQL, как `SELECT`. При этом широко используются и анализируются планы выполнения запросов, разбираются и сравниваются по быстродействию альтернативные варианты решения одной и той же типичной задачи. Изложение начинается с простых запросов и заканчивается достаточно сложными SQL-конструкциями.

Далее в *главе 5* подробно рассматривается ряд полезных и интересных примеров практического программирования на T-SQL. Например, приводятся листинги программ автогенерации данных для базовых таблиц, включая способы заполнения данными столбца таблицы, содержащего рисунки.

Глава 6 посвящена вопросам оптимального проектирования транзакций. Сначала приведены сведения по общетеоретическим аспектам этого вопроса. Да-

лее подробно разбираются средства управления транзакциями собственно на SQL-сервере.

Книга ориентирована на бизнес-аналитиков, программистов приложений и администраторов, работающих с клиент-серверными технологиями реляционных баз данных, а также на преподавателей и студентов по соответствующим специальностям. При этом для чтения глав по проектированию баз данных какой-либо специальной подготовки не требуется. Однако для понимания идеологии оптимизации запросов необходимо знание языка SQL, хотя бы в объеме стандарта SQL-92.

Описание компакт-диска

Описание компакт-диска, прилагаемого к книге, приведено в табл. 1.

Таблица 1. *Описание компакт-диска*

Папки	Описание	Комментарии
\БД	Базы данных в формате MS SQL сервер 2000	
	MAT_УЧЕТ – БД для задачи материального учета	MAT_УЧЕТ_Data.MDF
	MAT_UCHET – БД для примеров	mat_uchet_Data.MDF
\PART1	Модели (проектные решения) для соответствующих рисунков главы 1 в виде файлов в формате Erwin 3.5.2 (*.er1). Листинги программ на T-SQL – текстовые файлы	mod_ris1_*.er1 Номер (номера), упомянутые после "1_" соответствуют номерам рисунков в тексте книги
\PART2… \PART6	Листинги программ на T-SQL – текстовые файлы	Номер, указанный в названии файла, соответствует номеру листинга в тексте книги. В начале каждого файла с листингом указано, в контексте какой БД нужно данную программу запускать. Если этой информации нет, то данные для соответствующей программы нужно задать согласно пояснениям в тексте книги
\PART5 \ГРАФИКА	Папка с графическими файлами	

ГЛАВА 1

Моделирование реляционных данных

База данных в самом общем смысле — это унифицированная совокупность данных, совместно используемых широким кругом пользователей, динамически изменяющаяся в процессе своего функционирования.

Если унифицированным форматом представления данных являются таблицы, то речь идет о реляционной базе данных.

1.1. Общая характеристика реляционной модели данных

Наиболее распространенная трактовка *реляционной модели данных* на сегодняшний день принадлежит К. Дейту, согласно которой эта модель состоит из трех частей.

- *Структурная часть* описывает, какие объекты рассматриваются реляционной моделью. Постулируется, что реляционной базой данных называется база данных, состоящая из набора отношений. Схемой реляционной базы данных называется набор заголовков отношений, входящих в базу данных.
- *Целостная часть* описывает ограничения специального вида, которые должны выполняться для любых отношений в любых реляционных базах данных. Это *целостность сущностей* и *целостность внешних ключей*.
- *Манипуляционная часть* описывает два эквивалентных способа манипулирования реляционными данными — реляционную алгебру и реляционное исчисление. С практической точки зрения, важным является вывод о реляционной полноте структурного языка запросов SQL в том или ином стан-

дарте, который и реализует манипуляционную часть реляционной модели в реальных СУБД.

1.1.1. Структурная часть реляционной базы данных

Основы реляционной модели данных были впервые изложены в статье Е. Кодда в 1970 г., которая послужила стимулом для дальнейшего развития реляционной модели данных. Сам термин *реляционное представление данных*, введенный Коддом, происходит от латинского RELATIO, что означает "отношение" или "таблица". Таблица является наиболее привычным и удобным вместилищем для хранения информации, имеет жестко оговоренное количество поименованных и упорядоченных столбцов (структуру), и может неограниченно расти по количеству строк.

Теоретической основой реляционных баз данных являются такие математические дисциплины, как теория множеств и реляционная алгебра или алгебра отношений. Фундаментальным понятием реляционной модели данных является теоретико-множественное определение понятия *отношения*, которое формулируется следующим образом.

Пусть даны N множеств $D1$, $D2$, ..., DN (не обязательно различных), называемых *доменами* отношения R. Тогда R есть отношение над этими множествами, если само множество R есть множество упорядоченных n-кортежей вида $<d1, d2, ..., dn>$, где $d1$ — элемент из $D1$, $d2$ — элемент из $D2$, ..., dn — элемент из DN.

Отношение R, определенное на множестве доменов $D1$, $D2$, ..., DN, имеет две части: заголовок и тело.

Заголовок отношения содержит фиксированное количество названий атрибутов (столбцов) отношения. Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов. Заголовок статичен, он не меняется во время работы с базой данных. Если в отношении изменены, добавлены или удалены атрибуты, то в результате получим уже другое отношение (пусть даже с прежним именем).

Тело отношения представляет собой набор кортежей, т. е. подмножество декартового произведения доменов, что собственно и является отношением в математическом смысле слова. Тело отношения может модифицироваться во время работы с базой данных — кортежи могут изменяться, добавляться и удаляться.

Понятие отношения фактически лежит в основе теории реляционных баз данных. В практической реализации любой реляционной СУБД отношения являются математическим аналогом *таблиц*. В табл. 1.1 приведены термины, которые употребляются как синонимы.

Таблица 1.1. *Сравнение терминологии*

Реляционный термин	**Соответствующий "табличный" термин, связанный с программной реализацией СУБД**
Отношение	Таблица
Заголовок отношения	Заголовок таблицы
Тело отношения	Тело таблицы
Атрибут отношения	Наименование столбца (поля) таблицы
Кортеж отношения	Строка (запись) таблицы
Степень отношения	Количество столбцов таблицы
Мощность (кардинальность) отношения	Количество строк таблицы
Домен	Базовый или пользовательский тип данных

Свойства отношений непосредственно следуют из приведенного выше определения. Принципиальными здесь являются следующие моменты.

1. Все элементы отношения есть однотипные кортежи, что позволяет считать их аналогами строк в простой таблице, т. е. в такой таблице, в которой все строки состоят из одинакового числа ячеек и в соответствующих ячейках содержатся одинаковые типы данных.

2. За исключением крайнего случая отношение включает в себя не все возможные кортежи из декартового произведения доменов, на которых оно определено. Это значит, что для каждого отношения имеется критерий, позволяющий определить, какие кортежи входят в отношение, а какие нет. Этот критерий определяет смысл, или семантику, отношения. Он является логическим выражением и называется *предикатом отношения R*.

3. В отношении нет одинаковых кортежей, и кортежи не упорядочены сверху вниз, т. к. само отношение есть множество кортежей, и, следовательно, неупорядочено. Кроме того, как всякое множество отношение не может содержать неразличимые элементы. Поэтому одно и то же отношение может быть *изображено* разными таблицами, в которых *строки идут в различном порядке*.

1.1.2. Целостная часть реляционной базы данных

Домены

В реляционной теории принято считать, что все значения атрибутов отношения атомарны. Это следует из трактовки понятия домена. Домен можно рассматривать как подмножество значений некоторого типа данных, имеющих определенный смысл. Реляционная модель требует, чтобы типы используемых данных были простыми (скалярными), т. е. не обладающими внутренней структурой.

Конечно, понятие атомарности является весьма относительным. Так, строковый тип данных можно рассматривать как одномерный массив символов, а целый тип данных — как набор битов. Важно лишь то, что при переходе на такой низкий уровень теряется *семантика (смысл) данных*. Если строку, выражающую, например, фамилию сотрудника, разложить в массив символов, то при этом теряется смысл такой строки как единого целого.

Итак, *домен* — это семантическое понятие, которое несет определенную смысловую нагрузку. Домен имеет уникальное имя в пределах базы данных, определен на простом типе данных или на другом домене. Возможно наличие некоторого логического условия, позволяющего описать подмножество данных, допустимых для данного домена. Например, домен D, имеющий смысл "возраст сотрудника", можно описать как следующее подмножество множества натуральных чисел: ($D = n \in N$: $n \geq 18$ и $n \leq 60$).

Собственно для реляционной модели данных тип используемых данных не важен. Требование того, чтобы тип данных был *простым*, нужно понимать так, что *в реляционных операциях не должна учитываться внутренняя структура данных*. Конечно, должны быть описаны действия, которые можно производить с данными как с единым целым, например, данные числового типа можно складывать, для строк разрешается операция конкатенации и т. д. С этой точки зрения, если рассматривать массив как единое целое и не использовать поэлементных операций, то его можно считать простым типом данных. Более того, допустимо создать свой, сколь угодно сложный тип данных, описать возможные действия с ним, и если в операциях не требуется знание внутренней структуры данных, то такой тип данных также будет простым с точки зрения реляционной теории.

Основное назначение доменов состоит в том, что они *ограничивают сравнения*. Некорректно, с логической точки зрения, сравнивать значения из различных доменов, даже если они имеют одинаковый тип. Синтаксически правильный запрос "выдать список всех товаров, у которых учетная цена больше имеющегося на складе количества", не соответствует смыслу понятий "количество" и "цена". Таким образом, понятие домена помогает правильно *моделировать* предметную область.

Перейдем к введению понятий, определяющих целостную часть реляционной базы данных. Основными компонентами целостности являются, как уже было сказано, два ограничения, которые должны выполняться в *любой* реляционной базе данных: целостность сущностей и целостность внешних ключей.

Целостность сущностей

Под *сущностью* понимается некоторый объект, представляющий интерес для описания предметной области при создании модели базы данных. Понятие сущности возникает из методологии семантического моделирования данных, которая подробно рассматривается в *главе 1*. Финальной целью этого процесса является представление сущностей в виде отношений, поэтому для введения понятий, связанных с описанием целостной части реляционной базы данных, достаточно понимания сущности как синонима понятия отношения или таблицы.

Вводятся следующие понятия.

Ключ (возможный ключ, потенциальный ключ) отношения — это минимальный набор атрибутов, который может быть использован для однозначной идентификации любого кортежа отношения. Ключи отношений бывают *атомарными* и *составными*. Атомарный ключ состоит из единственного атрибута. Составной ключ представляет собой набор атрибутов. Ключ не должен содержать лишних атрибутов. Это означает, что если любой атрибут исключить из ключевого набора, то оставшихся атрибутов будет уже недостаточно для идентификации кортежа.

Любое отношение имеет по крайней мере один потенциальный ключ. Действительно, если никакой атрибут или группа атрибутов не являются потенциальным ключом, то в силу уникальности кортежей все атрибуты вместе образуют потенциальный ключ.

Отношение может иметь несколько потенциальных ключей. Традиционно один из них объявляется *первичным ключом* (PK, Primary Key), а остальные — *альтернативными*. Различия между первичными и альтернативными ключами могут быть важны в конкретной реализации реляционной СУБД. Далее этому вопросу будет уделено должное внимание.

Понятие потенциального ключа является *семантическим* и отражает введенную определенным образом трактовку понятий из конкретной предметной области. Например, рассмотрим отношение "Сотрудники", представленное табл. 1.2.

На первый взгляд может показаться, что в этом отношении имеются три потенциальных ключа — в каждой колонке таблицы содержатся уникальные данные. Однако среди сотрудников могут быть однофамильцы и сотрудники

с одинаковой зарплатой. Табельный же номер по сути своей уникален для каждого сотрудника, и именно *понимание семантики данных* привело к утверждению, что в данном отношении имеется только один потенциальный ключ — "Табельный номер".

Таблица 1.2. *Отношение "Сотрудники"*

Табельный номер	Фамилия	Зарплата
1002	Иванов	1000
1003	Петров	2000
1004	Сидоров	3000

Если представить отношение "Сотрудники" в виде табл. 1.2, не сообщив смысл наименований атрибутов, то окажется невозможным судить, может или не может в этом отношении появиться, например, кортеж (1002, Петров, 3000). Если бы такой кортеж появился, что не противоречит уникальности кортежей, то можно точно сказать, что *не является* альтернативным ключом — ни один из атрибутов по отдельности. Но нельзя сказать, что же *является* ключом отношения.

Потенциальные ключи служат единственным *средством адресации на уровне кортежей* в отношении. Только знание значения потенциального ключа кортежа позволяет точно указать этот кортеж.

С точки зрения семантического моделирования данных, потенциальные ключи служат *средством идентификации* объектов предметной области — экземпляров сущностей, данные о которых хранятся в отношении. Поскольку эти экземпляры должны быть различимы по определению, их идентификаторы не могут содержать неизвестные значения.

Обычно для ситуации наличия неизвестных или неполных данных используются типы данных, пополненные так называемым *NULL-значением*.

NULL-значение — это, собственно, не значение, а некий маркер, показывающий, что значение неизвестно. За кажущейся естественностью такого подхода скрываются весьма неоднозначные проблемы, наиболее явной из которых является необходимость использования трехзначной логики при оперировании NULL-значениями.

Проблема использования NULL-значения в теории реляционных баз данных окончательно не решена. Практически все реализации современных реляционных СУБД позволяют использовать NULL-значения, несмотря на их недостаточную теоретическую обоснованность. Безусловно, следует избегать появления NULL-значений, обусловленных некорректным проектированием базы

данных, но в реальной ситуации, когда какие-то данные об экземпляре сущности просто неизвестны, без NULL-значений не обойтись.

Правило целостности сущностей. Каждая сущность, реализованная тем или иным отношением, должна иметь по крайней мере один потенциальный ключ, входящие в состав которого атрибуты не могут принимать NULL-значений. Этот потенциальный ключ лучше всего объявлять первичным ключом таблицы, соответствующей данной сущности.

Следует отметить, что большинство СУБД вполне позволяют создавать таблицы и без первичных ключей. Однако нарушение правила целостности сущностей на практике сразу дает о себе знать. Например, для СУБД MS SQL-сервер станет невозможным доступ к данным по технологии OLE DB Provider, что сразу отчетливо видно при работе с приложением SQL Server Enterprise Manager — ни вставить, ни удалить записи визуальными средствами в таблице не удастся.

Целостность внешних ключей

Различные объекты предметной области, информация о которых хранится в базе данных, всегда взаимосвязаны. Наиболее типичный способ подобной связи между отношениями описывается ограничением внешнего ключа (FK, Foreign Key). Дадим точное определение.

Пусть дано отношение *R*. Подмножество атрибутов *FK* отношения *R* будем называть *внешним ключом*, если существует отношение *S* (*S* может совпадать с *R*) с потенциальным ключом *K*, и каждое значение *FK* в отношении *R* всегда совпадает со значением *K* для некоторого кортежа из *S* либо является NULL-значением.

Отношение *S* называется *родительским отношением*, а отношение *R* — *дочерним отношением*. Атрибут *K* часто называют вторичным ключом, а пару атрибутов *K* и *FK* — полями связи отношений *S* и *R*.

Другими словами, для каждого отличного от NULL-значения поля, входящего во внешний ключ дочернего отношения, существует одно и только одно значение поля связи в родительском отношении. На уровне реализации СУБД поле связи в родительском отношении — это чаще всего его первичный ключ, а если нет, то оно должно быть объявлено альтернативным ключом отношения.

Внешний ключ, так же как и потенциальный, может быть простым и составным.

Внешний ключ должен быть определен на тех же доменах, что и соответствующий первичный ключ родительского отношения.

Внешний ключ, как правило, *не обладает свойством уникальности*. Так и должно быть, поскольку в дочернем отношении может быть несколько кортежей, ссылающихся на один и тот же кортеж родительского отношения, что, собственно, и дает тип связи между отношениями *"один-ко-многим"*. Это стандартный тип связи с сохранением ссылочной целостности. Если внешний ключ все-таки обладает свойством уникальности, то связь между отношениями имеет тип "*один-к-одному*".

Хотя каждое значение внешнего ключа обязано совпадать со значениями потенциального ключа в некотором кортеже родительского отношения, обратное, вообще говоря, неверно. В поле связи родительской таблицы могут присутствовать значения, на которые не ссылается ни одно из значений внешнего ключа.

NULL-значения для атрибутов внешнего ключа допустимы только в том случае, когда атрибуты внешнего ключа не входят в состав никакого потенциального ключа.

Поскольку внешние ключи фактически служат ссылками на кортежи в другом (или в том же самом) отношении, то эти ссылки не должны указывать на несуществующие объекты.

Сформулированные выше соображения определяют *правило целостности внешних ключей* или *ссылочной целостности* реляционной базы данных: внешние ключи не должны быть несогласованными, т. е. для каждого значения внешнего ключа должно существовать соответствующее значение в поле связи в родительском отношении.

Примером ограничения внешнего ключа может служить следующая ситуация. Пусть имеются студенты, объединенные в группы. Каждый студент может обучаться только в одной группе. Студенты идентифицируются студенческим билетом (Студбилет), а группы студентов идентифицируются специально введенными уникальными номерами (№_группы). Тогда между множеством значений атрибута Студбилет и множеством значений атрибута №_группы имеет место связь внешнего ключа, причем атрибут №_группы является внешним ключом для атрибута Студбилет.

Замечание

Существуют и другие типы ограничений целостности в реляционной базе данных, которые реализуются разными способами. Но на современном уровне конкретная СУБД считается в полном смысле реляционной, если в ней есть стандартные средства автоматической реализации ограничений целостности сущностей и целостности внешних ключей.

Механизмы реализации ограничений целостности реляционной базы данных будут подробно рассматриваться на протяжении всей книги.

1.1.3. Манипуляционная часть реляционной базы данных

Манипуляционная часть реляционной базы данных описывает средства, с помощью которых, во-первых, из данных, хранящихся в базе данных, можно получать какие-либо выборки или сводные результаты, а во-вторых, изменять сами эти данные или их структуру.

Манипуляционной частью современных промышленных СУБД является специальный язык запросов, называемый SQL, который стал фактическим стандартом доступа к реляционным базам данных. Операторы этого языка по своей сути представляют собой требование к базе данных на выполнение каких-то действий. Язык SQL является реляционно полным. Это означает, что любой оператор реляционной алгебры может быть выражен подходящим оператором языка SQL.

Все СУБД, претендующие на название "реляционные", реализуют тот или иной диалект SQL. Многие нереляционные системы также имеют средства доступа к реляционным данным. Целью стандартизации является возможность переноса приложений между различными СУБД. Однако в настоящее время ни одна из систем не реализует стандарт SQL в полном объеме. Кроме того, во всех диалектах языка имеются возможности, не являющиеся стандартными. Таким образом, можно сказать, что каждый диалект — это надмножество некоторого подмножества стандарта SQL. Подобная ситуация затрудняет перенос приложений между разными СУБД, но скорее всего добиться полной стандартизации невозможно, т. к. динамика развития этой сферы такова, что организации, ведающие установлением и распространением стандартов, просто не успевают за производителями программных новинок.

Диалектом языка SQL, реализованным в СУБД SQL-сервер, является так называемый язык T-SQL, который сочетает в себе операторы языка SQL и традиционные средства структурного программирования. Изучение синтаксиса языка T-SQL выходит за рамки данной книги. Безусловно, его знание является обязательным для понимания материала, изложенного в *главе 5*, которая посвящена программированию на SQL-сервере. Однако начальные главы, посвященные приемам моделирования реляционной базы данных, жестко не связаны с целевой СУБД.

1.2. Технология клиент-сервер

В предыдущем разделе рассмотрены особенности, которые связаны с понятием унифицированного формата данных в определении базы данных. Показано, что принятие формата данных, основанного на таблицах или отношениях,

привело к реляционной модели данных со всеми ее характерными особенностями.

Однако в этом определении имеются еще составные части, которые обязательно необходимо реализовать. Речь идет о фразе "совокупность данных ... совместно используемых широким кругом пользователей, динамически изменяющаяся в процессе своего функционирования", т. е. о том, что информация из базы данных должна быть *одновременно* и *без задержки* предоставлена широкому кругу лиц в необходимой для них форме для анализа или модификации таким образом, чтобы по возможности они не ощущали присутствия друг друга.

Технология клиент-сервер означает такой способ взаимодействия программных компонентов, при котором они образуют единую систему. Как видно из названия, существует некоторый *клиентский процесс*, требующий определенных ресурсов, а также серверный процесс, который эти ресурсы предоставляет. При этом совсем необязательно, чтобы эти процессы выполнялись на одном и том же компьютере. Наоборот, на практике принято размещать серверный процесс на одном узле локальной сети, а клиентские — на других узлах.

В контексте базы данных клиент управляет пользовательским интерфейсом и логикой приложения, действуя как интеллектуальная рабочая станция, на которой выполняются приложения баз данных. Клиент принимает от пользователя запрос, проверяет его синтаксис и генерирует SQL-запрос к серверу на предоставление тех или иных данных. Этот запрос передается на сервер по компьютерной сети. Сервер принимает и обрабатывает запрос к базе данных, затем в зависимости от типа запроса либо передает полученные данные обратно клиенту, либо выполняет модификацию содержимого базы данных. Клиент принимает переданные сервером данные, форматирует их и представляет пользователю.

Обработка сервером запроса включает в себя проверку полномочий клиента, обеспечение требований целостности, собственно выполнение запроса и передачу клиенту необходимых результатов. При этом сервер занимается такими проблемами, как поддержка параллельности работы многих клиентов, которая включает в себя, в первую очередь, согласование данных, одновременно предоставляемых и изменяемых разными клиентами. Подобная архитектура обладает следующими преимуществами:

- обеспечивается более широкий доступ к существующим базам данных;
- повышается общая производительность системы. Поскольку клиенты и сервер находятся на разных компьютерах, их процессы способны выполнять приложения параллельно. При этом настройка производительности

сервера упрощается, если на компьютере, выделенном под сервер, выполняется только работа с базой данных;

- сокращается нагрузка на компьютерную сеть. Это происходит прежде всего за счет того, что в ответ на запрос клиента сервер возвращает ему готовые результаты запроса, а не все данные, необходимые для их получения;
- считается, что существует тенденция к снижению стоимости аппаратного обеспечения, т. к. наиболее ресурсоемкими операциями для любой СУБД является выполнение реляционных запросов. В случае клиент-серверной платформы все они выполняются на сервере, следовательно, высокие аппаратные требования предъявляются теперь в первую очередь к компьютеру, выделенному под сервер. Кроме того, более скромные требования теперь предъявляются и к аппаратным средствам поддержки сети;
- появляется возможность использования специализированного аппаратного обеспечения для сервера, которое может быть сконструировано именно для работы на сервере баз данных. Это очень существенный аспект повышения общей производительности работы системы;
- повышается уровень непротиворечивости данных, поскольку все проверки выполняются в одном месте.

Клиент-серверные платформы приходят на смену так называемым платформам типа *файл-сервер*, уверенно их вытесняя. В подобной ситуации, разумеется, интересными представляются данные о том, насколько более производительными являются клиент-серверные технологии по сравнению с файл-серверными, в чем заключаются их дополнительные преимущества и какую же цену за все это нужно платить.

Рассмотрим вкратце, как работает типичное файл-серверное приложение. Основной принцип его реализации состоит в том, что обработка данных ведется рабочей станцией, а сервер служит просто как дополнительное, доступное всем пользователям дисковое устройство. Это означает, что при выполнении задачи, например при сборке отчета, *вся* база данных или значительная ее часть прокачивается по сети на рабочую станцию и там обрабатывается процессором рабочей станции. Быстродействие такой системы зависит от быстродействия диска сервера, скорости передачи данных по сети, мощности процессора рабочей станции, объема ее оперативного запоминающего устройства и некоторых других факторов. Центральный процессор сервера играет второстепенную роль и должен просто обеспечивать передачу потока данных с сетевого канала на диск и обратно, по возможности не замедляя этот процесс. Главным при таком подходе является то, что практически вся база данных перегоняется по сети на рабочую станцию для последующей обработки. Поэтому если несколько станций одновременно выполняют сборку

отчетов, то на все эти рабочие станции закачивается база данных, и естественно система в целом существенно замедляет работу из-за перегрузки сети.

Когда, с точки зрения передачи данных по сети, выполняются менее накладные операции, например ввод нового документа, объем перекачки данных будет значительно меньше даже с учетом того, что в реальности ввод документа, как правило, сопровождается поиском клиента в справочнике, вычислением его задолженности и т. п. Эти моменты увеличивают объем передачи количества информации с диска сервера на рабочую станцию, но в итоге нагрузка на сеть обычно бывает меньшей, чем при составлении отчетов.

Не следует также забывать о необходимости синхронизации доступа рабочих станций к данным. Поскольку вся обработка ведется на уровне рабочих станций, а файл-сервер просто играет роль разделяемого дискового устройства, задачи синхронизации решаются в таких системах с помощью организации различных файлов блокировок на диске файл-сервера. В эти файлы каждая рабочая станция записывает информацию о данных, которые она модифицирует в данный момент, а при попытке считать данные проверяет, не заняты ли эти данные другой рабочей станцией. Несмотря на ряд недостатков, например таких, как висячие блокировки при аварийном выключении рабочих станций или замедление работы всей системы при модификации большого числа записей, способ управления блокировками, основанный на концепции файл-сервер, вполне работоспособен.

Легко сообразить, что производительность файл-серверной системы напрямую связана с объемом обрабатываемой и, следовательно, передаваемой по сети базы данных с учетом трафика всех одновременных соединений. В ситуации, когда сеть перестает справляться и "торможение" ощутимо дает о себе знать, пользователи системы идут на различные хитрости: закрывают старую базу и открывают новую каждый квартал, пытаются удалить старые данные и т. п. Однако любой бухгалтер может сказать, что данные нужны ему не за квартал, а как минимум за год и предпочтительно в динамике, а не в виде отдельных кусков. Да и долги клиентов иногда тянутся годами.

Временным решением проблемы в такой ситуации может стать увеличение пропускной способности сети за счет установки более мощного и дорогостоящего аппаратного обеспечения. Однако стратегически это не спасет ситуацию. Зачем увеличивать пропускную способность сети, если при росте базы данных все равно наступит момент, когда жесткий диск сервера достигнет предела своей производительности, т. к. все данные, которые уже перегрузили сеть, должны быть прочитаны именно с этого диска? На первый взгляд кажется, что последнее замечание не очень обосновано, — ведь и в клиент-серверных системах данные тоже читаются с того же диска. Однако объем физического чтения в последнем случае существенно уменьшается

благодаря наличию мощного механизма кэширования данных, который является неотъемлемой частью любой клиент-серверной системы.

Не следует забывать и еще об одной важной детали. Это архивирование, которое нужно делать каждый день. И при этом во время архивирования ни один из пользователей с базой данных работать не может.

Теперь рассмотрим клиент-серверную систему. С одной стороны, сами серверы, предназначенные для нее, должны располагать значительно большими ресурсами. С другой стороны, наличие при этом фантастической пропускной способности сети совсем не обязательно. Все дело в том, что в случае клиент-серверной технологии при работе с сервером рабочая станция не качает всю базу данных к себе по сети, а просто передает достаточно компактный запрос на сервер, где запрос и выполняет, после чего результаты запроса передаются обратно на рабочую станцию. Таким образом, трафик по сети значительно снижается. Конечно, если при разработке клиентской части программного обеспечения будут созданы запросы, результатами выполнения которых является вся база данных или значительная ее часть, то основное преимущество клиент-серверной технологии будет утрачено. Но этот недочет будет лежать на совести разработчиков прикладных программ.

Итак, первое, что необходимо понять и запомнить: основным преимуществом клиент-серверных систем не является ускорение выполнения выборок данных. Несколько лет назад журнал PC Magazine проводил сравнительный анализ (в том числе и по быстродействию) СУБД, построенных на основе обычных файл-серверов и с использованием клиент-серверных систем. Естественно, условия испытаний по возможности были нивелированы, т. е. применялись одинаковые объемы баз данных, одинаковые их структуры, один и тот же компьютер в качестве сервера, одинаковое количество рабочих станций и т. д. Во всех случаях средняя скорость выполнения запросов в клиент-серверных системах была ниже, чем у файл-серверных систем. Сейчас об этом эффекте можно прочесть в любой серьезной книге по SQL. Тестировщики не были удивлены полученным эффектом, поскольку понимали причину такого поведения клиент-серверных систем при заданных условиях эксперимента. Ведь задачей эксперимента было сравнение быстродействия двух видов систем, и поэтому были выбраны условия, позволяющие произвести это сравнение. В частности, для тестов использовались базы данных объемом 1,5—2 Гбайт. Ведь если проводить испытания, используя базы данных на порядок большего размера, то клиент-серверные варианты не с чем было бы сравнивать, поскольку обычная файл-серверная система на таких объемах информации просто не имеет возможности функционировать. Вот в этом-то и состоит основное отличие и достоинство клиент-серверных систем: они будут работать с вполне приемлемой скоростью с базами данных такого объема и с таким числом одновременных соединений, с которыми файл-

серверная система работать не сможет в том смысле, что ее функциональность, в том числе и быстродействие, станут неприемлемыми для коммерческих приложений.

К дополнительным преимуществам клиент-серверных систем можно отнести следующие моменты. Системы этого типа имеют встроенный механизм работы с транзакциями, в том числе и их отката. В файл-серверных системах также существует механизм работы с транзакциями, однако способ их реализации принципиально отличается. Это отличие заключается в том, что механизм управления транзакциями представляет собой не что иное, как просто блокировку всей базы данных до завершения выполнения критических по времени операций одной из рабочих станций. Откат возможен только при сохранении работоспособности рабочей станции, инициировавшей транзакцию.

В клиент-серверной системе этот механизм значительно более сложен. Он позволяет получить "слепок" базы данных на момент начала транзакции без блокировки базы данных. И слепков таких может быть много: для каждой рабочей станции свой. В случае "зависания" рабочей станции, открывшей транзакцию, эта транзакция может быть просто откачена, а база данных будет восстановлена в том виде, в каком она была до инициации транзакции. Откат осуществляется либо по запросу рабочей станции (при сохранении ее работоспособности), либо при перезагрузке рабочей станции, либо администратором клиент-серверной системы. Таким образом, выход из строя рабочей станции не столь опасен для целостности базы данных. Кроме того, клиент-серверная система ведет так называемый журнал транзакций. По сути база данных хранится в виде ее начального содержимого и модификаций, записанных в журнал транзакций. Такой способ хранения позволяет производить архивирование базы данных во время работы всей системы: просто состояние базы данных фиксируется на момент начала архивирования, отсекаются незавершенные транзакции, а основная база и часть журнала транзакций, содержащая завершенные транзакции, записываются в архив. Процесс архивирования легко поддается автоматизации, при которой присутствие оператора необязательно, т. к. клиент-серверные платформы обычно для этого имеют достаточно простые и удобные встроенные средства.

1.3. Общий обзор методологии проектирования информационных систем

Информация должна рассматриваться не только как некая принадлежащая предприятию ценность, но и как исходная точка для построения информационной системы, обслуживающей предприятие. *Информационное проектирование* становится наиболее популярной методологией, пронизывающей все

стадии жизненного цикла системы. Главная посылка, на которой строится данная стратегия, состоит в том, что целостность информационных систем определяется степенью охвата информационных элементов объекта соответствующей логической моделью. Информационное проектирование предполагает прежде всего глубокое исследование всех информационных систем, преследующее своей целью поиск ответа на вопрос: каким образом информация используется и разделяется системами. Программные структуры выступают по отношению к информационной модели "надстройкой", определяющей общую информационную инфраструктуру для создания соответствующих систем на предприятии. Таким образом, процедуры логически вытекают из данных, а качество информационных систем определяется качеством построения информационной модели. Информационное обследование организации — трудоемкая и продолжительная работа. Только опытным аналитикам, вооруженным мощным инструментарием и технологией, под силу обеспечить точность и целостность описания информации.

Начальные этапы жизненного цикла приложения базы данных, т. е. собственно ее проектирование, требуют применения адекватных средств автоматизированного создания программ, которые принято называть *CASE-средствами*. В самом широком смысле термин *CASE-инструмент* применим к любым средствам автоматизированного проектирования и создания программ. В частности, CASE-инструменты незаменимы при проектировании реляционной базы данных, т. к. они позволяют существенно повысить производительность и качество подобной работы. Этот позитивный эффект достигается за счет приведенных далее преимуществ.

- *Стандартизация*. CASE-инструменты способствуют расширению использования стандартов на всех этапах жизненного цикла приложения, начиная со стадии его проектирования и заканчивая повседневным обслуживанием в процессе эксплуатации.
- *Интеграция*. CASE-инструменты позволяют сохранять всю генерируемую информацию в специальном хранилище, или *словаре данных*. Таким образом, появляется возможность хранить полный объем данных, собранных на всех этапах жизненного цикла приложения, компоновать их между собой, что является важной предпосылкой успешной интеграции системы в целом.
- *Графические возможности*. Как правило, любые структурированные технологии широко используют диаграммы, которые достаточно трудно создавать и поддерживать вручную. Использование CASE-инструментов позволяет существенно упростить этот процесс, предоставляя возможность создания более корректной и актуальной документации.
- *Непротиворечивость*. Поскольку вся информация в словаре данных взаимосвязана, CASE-инструменты способны обеспечить автоматическую проверку этой информации на непротиворечивость.

- *Инжиниринг*, *реинжиниринг* и *синхронизация*. Современные CASE-инструменты позволяют автоматически преобразовывать фрагменты спецификации проекта в программный код, выполняя тем самым инжиниринг модели, и наоборот, создавать разнообразные модели, используя готовое приложение в качестве входных данных, осуществляя процесс, называемый *реинжинирингом*. Кроме того, если в функционирующее приложение вносятся изменения, то с помощью CASE-инструментов можно выполнять автоматическую синхронизацию словаря данных в соответствии со сделанными изменениями.

Методологии или технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой информационной системы. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают реализацию всех стадий жизненного цикла информационной системы.

Технология проектирования определяется как совокупность трех составляющих:

- *пошаговой процедуры*, определяющей последовательность технологических операций проектирования;
- *критериев и правил*, используемых для оценки результатов выполнения технологических операций;
- *нотаций* (графических и текстовых средств), используемых для описания проектируемой системы.

Моделирование данных является одной из важнейших составляющих проектирования информационной системы в целом. Цель моделирования данных состоит в обеспечении разработчика информационной системы концептуальной схемой базы данных в форме одной модели или нескольких локальных моделей, которые относительно легко могут быть отображены в любую СУБД. Наиболее распространенным средством моделирования данных в случае, когда выбрана реляционная СУБД, является так называемое *семантическое (инфологическое) моделирование*, которое представляет собой моделирование структуры данных основываясь на смысле этих данных. Другим общепринятым названием этого метода является термин *"сущность-связь"*. В качестве инструмента семантического моделирования используются различные варианты *диаграмм "сущность-связь"* (ER, Entity-Relationship или ERD). С их помощью определяются важные для предметной области объекты (сущности), их свойства (атрибуты) и отношения друг с другом (связи).

1.4. Сбор информации для моделирования данных

Перед тем как разрабатывать диаграмму "сущность-связь", необходимо собрать информацию о возможных сущностях предметной области и их атрибутах. Для этого существует множество способов, как формальных, так и неформальных. Укажем на некоторые из них.

Простейший неформальный способ — взять описание предметной области (возможно, полученное в процессе проведения интервью с заказчиком) и выделить в нем существительные и глаголы. Существительные будут кандидатами на роль сущностей, глаголы — кандидатами на роль связей. Естественно, приведенное правило является очень приблизительным, и каждая полученная таким образом сущность-кандидат должна проверяться коллективом аналитиков, возможно, даже голосованием.

Можно получить список сущностей на основе формального описания бизнес-процессов системы, так называемых *диаграмм бизнес-процессов* (business process diagram) или *диаграмм потоков данных* (DFD, Data Flow Diagram). *Подробную информацию см. в [14] и [15] списка литературы*. Два этих вида диаграмм близки по смыслу.

Построение диаграмм бизнес-процессов на начальном этапе проектирования должно служить одной основной цели: выделению сущностей в предметной области. Часто возникает желание смоделировать бизнес-процессы в организации до построения модели данных, т. е. разработать функциональную спецификацию системы, а потом уже определить данные, которых требует каждая функция. Это выглядит естественным, но может привести и к проблемам.

Как уже говорилось, желательна приоритетность информационного проектирования, т. е. построение модели данных до функциональной спецификации. Такой подход дает следующие преимущества:

- многократное использование данных для функций, которые не предвидятся в момент построения модели, и применение ранее выделенных функций в качестве тестовых для модели данных;
- установление устойчивых соглашений об именах и определениях для данных. В противном случае в функциональной спецификации одна функция могла бы быть названа "получить сведения о клиенте", а вторая — "обслужить заказчика", и возникнет вопрос, чем же заказчик отличается от клиента;
- автоматическое получение значительной части функциональной спецификации, т. к. просто посмотрев на построенную модель данных, можно предположить, какие функции необходимы для работы с этими данными;