Моделирование реляционных баз данных

ГЛАВА

2

В этой главе...

Видимые сущности

Идентификация множества сущностей

Реляционные шаблоны

Нормализация

Реляционная алгебра

Во время своего последнего путешествия я читал книгу Одри Ниффенеггер *Жена путешественника во времени*. Не вдаваясь в детали, скажу, что это прекрасная научно-фантастическая романтическая история. Женщина перемещалась по времени по соглашению, в то время как ее муж бесконтрольно бороздил пространственные и временны*е* просторы, возвращаясь к ключевым моментам своей жизни снова и снова. Несмотря на то что сюжет в данном случае более сложный, чем в среднестатистическом романе, мне понравилось, как автор сплела все детали в бурлящий поток. Каждая деталь подчеркивала черты героев и постепенно выстраивала сюжет.

В некотором роде база данных напоминает хороший роман. Фабулой романа является модель базы данных, а данные представляют собой персонажей и их действия. *Нормализация* — это грамматика и структура базы данных.

Когда два писателя рассказывают одну и ту же историю, каждый представляет ее по-своему. Аналогично, нет единственно правильного способа моделирования базы данных, — если база данных содержит всю информацию, необходимую для извлечения рассказа, и следует нормализованным правилам грамматики, то она будет работать. Вывод здесь следующий: как некоторые книги лучше других, так и с некоторыми базами данных работать легче, в то время как запросы к другим вызывают массу сложностей.

Как и при написании романа, основой моделирования данных является внимательное исследование и осознание действительности. Основываясь на своих наблюдениях, проектировщик создает логическую систему — новый виртуальный мир, который моделирует образ реальности. Таким образом, то, насколько ясно проектировщик представляет себе реальность и идентифицирует сущности и их взаимодействие, настолько хорошей будет модель виртуального мира. Как и в постмодернизме, не существует единого безукоризненного и правильного представления — только точка зрения автора или разработчика.

Роль проектировщика баз данных является главной в любом программном проекте, так как он закладывает основы, точнее, структуру виртуальной реальности, которая либо поддержит программный код, либо похоронит его заживо. Любая функция, работающая с данными, должна быть предусмотрена в создаваемой схеме данных.

Жизненный цикл схемы данных обычно переживает несколько поколений кода приложения, и это еще одна причина, почему схема данных столь критична для жизнеспособности проекта. Данные продолжают жить, в то время как приложения приходят и уходят. Если разработчики приложений обычно рассчитывают на их эксплуатацию в течение двух-трех лет, базы данных продолжают существовать десятилетиями.

С точки зрения метафоры автора и проектировщика, эта глава раскроет перед вами основные структуры реляционной базы данных: сущности, нормализацию и реляционные модели.

Моделирование реальности

С точки зрения принципов информационной архитектуры (см. главу 1) целью моделирования данных является удовлетворение потребностей заказчика за счет определения такой структуры данных, которая логически представляет конкретные объекты и события.

Основным элементом данных является их элементарный контейнер. Большинство людей представляют себе этот контейнер как ячейку в электронной таблице — пересечение экземпляра сущности (строки) и его атрибута (столбца). Моделирование данных является искусством помещения этого элементарного контейнера в правильное место среди миллионов ячеек, формирующих всю систему данных. Чтобы моделировать данные, нужно рассматривать каждую ситуацию, сцену и документ с точки зрения элементов данных и целостности реляционной модели.

Моделирование данных включает в себя несколько процессов. Разработку модели данных нельзя рассматривать как обычный последовательный процесс, поскольку каждый шаг в нем зависит от множества других, и открытия ждут проектировщика на протяжении всего процесса. В то же время любой проектировщик обязательно проходит через следующие этапы.

1. Исследования и сбор требований.
2. Логическое представление реальности.
3. Идентификация и проектирование видимых сущностей.
4. Разработка схемы.
5. Проектирование структуры приложения.

Ни одна схема не может совершенно моделировать реальность. Каждая схема разрабатывается для того, чтобы удовлетворить требованиям проекта на данной стадии его жизненного цикла. Умение отсеивать только реальные требования, поддержание баланса между реальностью и моделью, а также умение создавать схемы данных, которые поддерживают текущую версию и оставляют место для роста, являются не столько результатом чтения книг, сколько результатом опыта.

Целью логического проектирования является описание сущностей, их взаимосвязей и общих правил существования. При построении физической схемы за основу берется логическая, а затем она встраивается в структуру конкретной системы управления базами данных (СУБД).

Логическая и физическая модели

Я слышал много дискуссий на тему различий между логической и физической моделями базы данных. В некоторых организациях эти задачи выполняются различными командами разработчиков, которые порой даже не контактируют друг с другом. Некоторые проектировщики

настаивают на том, что их реализация в точности соответствует логической модели. В противоположность этому есть те, которые всегда строят сразу физическую схему.

Целью логического моделирования является документирование сущностей и их взаимосвязей. Здесь главное хорошо понять основные бизнес-правила, и ничего больше. На основе этого понимания команда проектировщиков может построить физическую схему, которая отвечает как бизнес-правилам, так и требованиям производительности.

В главе 17 мы сконцентрируем внимание на физической модели.

Математический подход к моделированию реляционных баз данных включает в себя некоторые правила нормализации — правила, которые проверяют правильность модели базы данных. Разработав за свою жизнь сотни нормализованных моделей баз данных, я нашел эффективный подход к этому процессу, основанный на видимости и поддержке сущностей. *Видимая сущность* представляет собой элемент, который может распознать пользователь. *Поддерживаемой* называют абстрактную сущность, разработанную проектировщиком базы данных для физической поддержки общей логической модели.

Видимые сущности

Видимые сущности обычно представляют объекты, которые могут распознать люди. Многие объекты являются именами собственными, такими как люди, города или местности, а некоторые — нарицательными, такими как контакты или марки автомобилей. Видимые сущности могут представлять также действия, такие как сбор материала или упорядочивание списка. Обычно видимый объект уже представлен где-либо в документе.

Некоторые видимые объекты представляют собой методы, с помощью которых группируются или организуются некоторые другие первичные объекты. Например, покупатели понимают, что продукты делятся на множество категорий: молочные, мясные, хлебо-булочные и т.п. В качестве еще одного примера можно привести заказы на поставки, которые могут иметь несколько категорий срочности.

Каждая строка — это остров

Каждая строка должна представлять некоторый полный логический объект, а каждый атрибут в ней должен относиться именно к этому объекту.

Каждая сущность основана предполагает, что каждая строка представляет некоторый объект или действие. На этой посылке и основано моделирование данных. Ключевым моментом является идентификация этих уникальных и общих объектов и действий. Именно это подразумевает термин *система реляционной базы данных*. Это не значит, что одни сущности связаны с другими каким-то образом — это значит, что все однородные или связанные данные сведены в одну сущность.

Только данные, описывающие объекты, принадлежат этой сущности. Некоторые из возможных атрибутов, даже если они перечислены на листе бумаги для одного объекта, могут на самом деле описывать и другой объект. Стандартная форма заказа обычно включает в себя имя заказчика и его адрес, хотя это имя принадлежит именно заказчику, а не самому заказу. Адрес тоже принадлежит заказчику. В то же время, если в следующем году заказчик сменит свой адрес, в заказе останется старый адрес. Вывод следующий: для моделирования данных критичны основательный анализ объектов и их свойств.

Хорошо спроектированные сущности имеют массу однотипных элементов. Например, одна сущность овощной лавки может содержать массу товаров, вместо того, чтобы яблоко представляло собой одну сущность, арбуз — другую и т.д.

Первичные ключи

Каждая строка в сущности представляет собой единичный уникальный объект в реальном мире. Так же как не могут существовать два идентичных самолета, две строки не могут представлять один и тот же самолет. Для логической гарантии уникальности каждой из строк один из атрибутов назначается первичным ключом — именно по нему обращаются к строкам таблицы команды модификации данных. Если вы обеспечили, чтобы каждому объекту в реальности соответствовал всего один первичный ключ и наоборот, значит, вы сделали свою работу хорошо.

Чтобы привести пример первичного ключа в физической схеме, предположим, что в базе данных компании каждый клиент однозначно определяется своим идентификатором. С помощью специального столбца идентичности SQL Server для строки каждого нового клиента, когда она вставляется в таблицу, автоматически генерирует новое целое число.

Таблицы, строки и столбцы

Реляционные базы данных собирают однообразные или связанные данные в единый список. Например, вся информация о товарах может быть перечислена в одной таблице, а о клиентах — в другой.

Таблицы базы данных в чем-то сходны с электронными таблицами и также состоят из строк и столбцов. Электронные таблицы привлекательны своим стилем обработки информации, облегчающим их добавление и модификацию. На практике менеджеры склонны хранить критичную информацию в электронных таблицах, и многие базы данных берут свое начало именно из них.

Как в электронных таблицах, так и в таблицах баз данных каждая строка представляет некоторый элемент списка, а каждый столбец является некоторым фрагментом данных об этом элементе. В то время как электронные таблицы обычно имеют произвольную структуру, таблицы базы данных могут предъявлять строгие требования к данным в столбцах. Так как целостность строк и столбцов критична в таблице базы данных, то и сама конструкция таблицы критична.

За многие годы разные стили разработки обращались к столбцам таблиц с помощью разных терминов, которые перечислены в табл. 2.1.

Таблица 2.1. Сравнение терминов баз данных

Стиль разработки	Общий список	Элемент в списке	Элемент информации в списке
Электронная таблица	Электронная таблица, рабочий лист, именованный диапазон	Строка	Столбец, ячейка
Старое программное обеспечение	Файл	Запись	Поле
Реляционная алгебра, логическое проектирование	Сущность	Кортеж	Атрибут
SQL, физическое проектирование	Таблица	Строка	Столбец
Объектно-ориентированное проектирование	Класс	Экземпляр объекта	Свойство

Разработчики приложений SQL Server при обсуждении физической схемы обычно называют элементы базы данных таблицами, строками и столбцами, а иногда при обсуждении логической схемы используют термины *сущность*, *кортеж* и *атрибут*. В оставшейся части

книги мы будем использовать термины физической схемы, но поскольку эта глава посвящена теории проектирования, в ней мы будем пользоваться терминами реляционной алгебры (сущность, кортеж и атрибут).

Идентификация множества сущностей

Каждая из сущностей должна быть полной, т.е. содержать все атрибуты, зависящие от первичного ключа, в то же время сама по себе она не способна в достаточной мере отразить модель. Для моделирования любой бизнес-функции или организационной задачи требуется множество сущностей.

Дополнительные сущности отражают множество дополнительных фрагментов информации об основных объектах или их группах и объединяют их в единую систему. При разработке логической модели данных различные типы логических сценариев требуют наличия множества сущностей, в том числе следующих:

- множество объектов;
- отношения между объектами;
- организация или группировка объектов;
- целостность значений;
- сложные объекты.

Иногда разграничение между объектами, их значениями и группировкой нечетки. Однако при поддержке всех вышеперечисленных сущностей модель данных будет полной.

Множество объектов

Иногда оказывается, что один объект на самом деле является списком множества других объектов, как в приведенных ниже примерах.

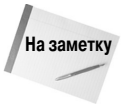
- В базе данных туристического агентства один тур может упоминаться несколько раз, но каждый раз с разными датами.
- В базе данных семей каждый человек может иметь несколько детей.
- Табель рабочего времени сотрудника может содержать множество штампов времени. Этот табель рассматривается как один объект, однако при ближайшем рассмотрении он окажется списком временны*х* событий.
- Календарь сотрудника может содержать множество напоминаний.

Отношения между объектами

Наиболее распространенным назначением существования множества сущностей является описание некоторого типа отношений между двумя разными объектами. Соответствующие примеры приведены ниже.

- В базе данных турагентства клиенты участвуют в турах, а проводят их гиды. Существуют отношения между клиентами и турами, а также между гидами и турами.
- Материал может создаваться из множества других материалов.
- Участковый врач может обслуживать несколько членов семьи; это отношение между ним и этими членами.

- В системе проверки качества программного обеспечения в одной функции может быть обнаружено множество ошибок.



Модель схемы должна предусматривать обеспечение наилучшей производительности для наиболее вероятного сценария, а также обработку исключений. Для примера предположим, что 96% клиентов имеют всего один адрес, 4% имеют еще один адрес, а всего один клиент имеет множество филиалов, разбросанных по всему миру. Добавление в таблицу первых 96% клиентов не потребует дополнительных объединений. Добавление дополнительных адресов потребует наличия дополнительной таблицы, объединенной отношением “один ко многим”. Придется использовать флаг в таблице клиентов для индикации наличия многочисленных офисов, чтобы избежать проверки левым внешним объединением наличия дополнительных офисов у всех клиентов. (Объединения мы детально рассмотрим в главе 9.)

Этот аспект проектирования может возникнуть при разработке логической схемы; физическая схема также может потребовать его для своей оптимизации.

Организация и группировка объектов

Объекты порой группируются в разные категории. Эти категории могут быть перечислены в отдельной сущности, как описано ниже.

- Клиенты могут быть сгруппированы по странам.
- Материалы могут группироваться по своему состоянию (сырье, товары в производстве, готовая продукция).
- В базе данных турагентства туры могут группироваться по странам.

Целостность значений

Атрибуты кортежа иногда требуют обеспечения целостности домена, т.е. вхождения в предопределенный диапазон значений. Соответствующие примеры приведены ниже.

- Тип кредитной карточки, используемой для покупок.
- Регион, которому принадлежат адреса.
- Код подразделения некоторой организации может принимать определенные значения.

Сложные объекты

Некоторые объекты в реальной жизни настолько сложные, что моделировать их одной сущностью невозможно. Информация принимает больше форм, нежели могут содержать один первичный ключ и одна строка таблицы. Чаще всего это касается составных объектов реального мира. Например, в одном заказе может содержаться множество строк. Эти строки являются частью заказа, однако для их моделирования потребуется создание дополнительной сущности.

Реляционные шаблоны

После того как объекты и действия будут организованы, следует определить отношения между ними. Каждое отношение связывает две сущности с помощью их ключей и имеет два следующих основных атрибута.

- **Строгость** — количество объектов, которые могут существовать на каждом конце отношения.
- **Обязательность** — является ли отношение обязательным.

Клиенты или бизнес-аналитики должны быть способны описать отношения между объектами с помощью терминов *включает*, *имеет* или *содержит*. Один клиент может разместить множество заказов. Один заказ может включать в себя множество товаров. Один и тот же товар может содержаться во множестве заказов.

Вторичные сущности и внешние ключи

Когда два объекта связаны отношением, одна из сущностей, как правило, является главной, а вторая — подчиненной, или вторичной. Один объект в главной сущности может быть связан с многочисленными объектами во вторичной, как показано на рис. 2.1.

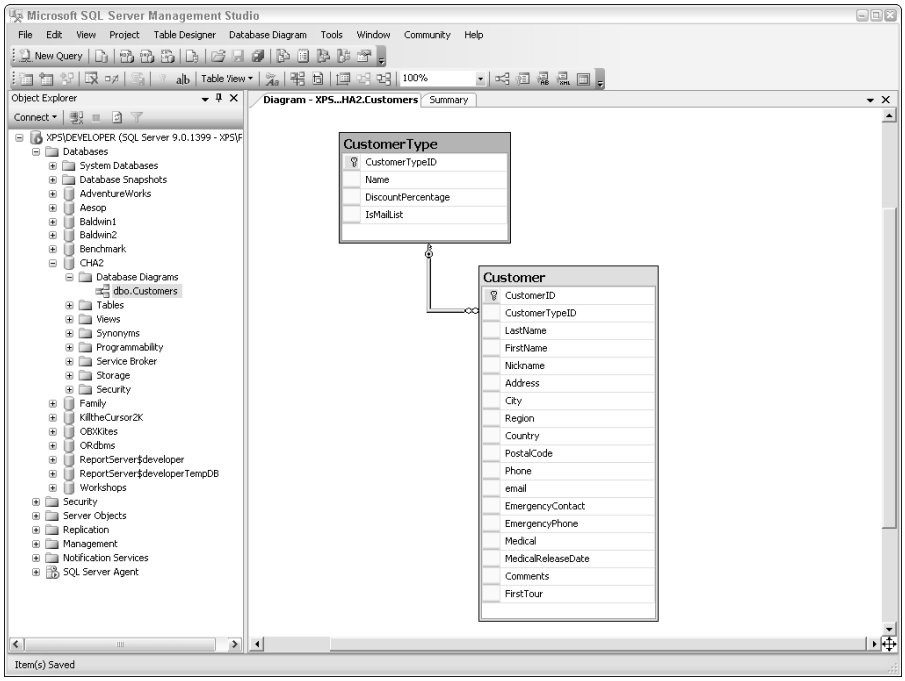


Рис. 2.1. Отношение "один ко многим" состоит из главной и вторичной сущностей. Внешний ключ вторичной сущности связан с первичным ключом главной

Назначение внешнего ключа — хранение значения первичного ключа главной сущности, чтобы можно было установить соответствие между сущностями.

Строгость отношения

Строгость отношения описывает количество элементов с каждой его стороны. Каждая из сторон отношения может допускать как один объект, так и их множество. Тип ключа определяет возможность множества объектов. Первичные ключи допускают наличие только одного объекта, в то время как внешние — их множества.

Существуют различные возможные варианты строгости; все они перечислены в табл. 2.2. В этом разделе мы подробно рассмотрим каждый из этих вариантов.

Таблица 2.2. Строгость отношений

Тип отношения	Ключ первой сущности	Ключ второй сущности
Один к одному	Первичный ключ главной сущности — один объект	Первичный ключ главной сущности — один объект
Один ко многим	Первичный ключ главной сущности — один объект	Внешний ключ вторичной сущности — множество объектов
Многие ко многим	Внешний ключ главной сущности — множество объектов	Внешний ключ вторичной сущности — множество объектов

Обязательность отношений

Вторым свойством отношений является их *обязательность*. Различие между обязательным и необязательным отношением критична для поддержания целостности базы данных.

Некоторые вторичные объекты требуют, чтобы их внешний ключ указывал на первичный. В этом случае вторичный объект будет неполным или бессмысленным без наличия связанного с ним первичного объекта. Например, в следующих ситуациях критично, чтобы отношение имело обязательный тип.

- Строка заказа без самого заказа бессмысленна.
- Заказ без заказчика не имеет смысла.
- В базе данных турагентства событие, не привязанное к некоторому туру, бессмысленно.

В противоположность этому некоторые отношения не обязательны. В этих случаях в реальном мире вторичный объект может существовать без первичного. Приведем два примера.

- Заказчик, не имеющий дисконтной карточки, все равно остается клиентом.
- Заказ может иметь и не иметь код приоритета — в любом случае он остается заказом.

Некоторые разработчики баз данных предпочитают избегать необязательных отношений, поэтому проектируют все отношения как обязательные и создают объекты, которым не нужен вторичный ключ, указывающий на суррогатный объект в главной таблице. Например, вместо того, чтобы разрешить пустые значения в столбце скидки, в отдельную сущность вставляется одна строка с нулевой скидкой, и все клиенты без дисконтной карты указывают на этот элемент.

Существуют две причины, чтобы избегать суррогатных пустых элементов. Во-первых, добавляется работа проектировщику базы данных (дополнительные вставки и проверки внешнего ключа). Во-вторых, гораздо легче работать без лишнего отношения при использовании предложения `WHERE` *`столбец`* `IS NOT NULL`. Пустые элементы являются полезным элементом общей модели, и игнорирование их достоинств только добавляет работы как разработчику, так и самой базе данных.

В некоторых редких ситуациях требуется поставить обязательность в зависимость от некоторого условия. В зависимости от правил отношение может устанавливаться в следующих случаях.

- Если организация иногда продает товары, изготовленные под заказ, которые отсутствуют в основной сущности. В этом случае обязательность отношения зависит от характера товара. Сущность строки заказа для товаров может использовать два атрибута.

Если используется атрибут идентификатор товара, то он должен указывать на корректный первичный ключ сущности товара.

- Если же используется атрибут заказного товара, то атрибут идентификатора товара остается пустым.

Реализация обязательности отношений зависит от конкретной физической схемы. Единственной целью логического проектирования является моделирование объектов организации, их отношений и бизнес-правил.

Диаграмма модели данных

Те, кто занимается моделированием данных для графического отображения создаваемых моделей, используют несколько методов. Популярным является ER-метод Чена, а система Visio Professional содержит еще пять других методов. Метод, который предпочитаю лично я, — самый простой. Он предполагает зарисовку схемы на доске, как показано на рис. 2.2. Строгость отношения отображается одной или тремя линиями (куриная лапа). Если отношение необязательное, то около внешнего ключа помещается окружность.

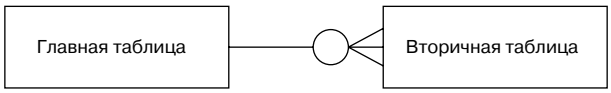


Рис. 2.2. Простой метод отображения на диаграммах логических схем

Еще одним преимуществом этого простого метода построения диаграмм является то, что он не требует установки расширенной версии программы Visio.

Отношения “один ко многим”

На сегодняшний день самым распространенным типом отношений является “один ко многим”. Несколько элементов вторичной сущности связываются с одним элементом главной сущности. Отношение устанавливается между первичным ключом главной сущности и внешним ключом вторичной сущности. Приведем несколько примеров.

- В базе данных турагентства Cape Hatteras Adventures каждый базовый лагерь может иметь несколько туров, которые берут в нем начало. Каждый тур может начинаться только из одного базового лагеря; таким образом, один лагерь связан отношением “один ко многим” с несколькими турами. Отношение устанавливается между первичным ключом сущности `BaseCamp` (базовые лагеря) и внешним ключом сущности `Tour` (туры) (рис. 2.3). Каждый атрибут внешнего ключа таблицы `Tour` содержит копию первичного ключа таблицы `BaseCamp`.

Рис. 2.3. Отношение “один ко многим” связывает первичный ключ с внешним ключом

- Каждый клиент может разместить множество заказов. Каждый заказ имеет свой уникальный первичный ключ `OrderID` (идентификатор заказа); сущность `Order` (заказы) также имеет вторичный ключ `CustomerID` (идентификатор клиента), указывающий

на клиента, разместившего заказ. Сущность `Order` может содержать несколько элементов, имеющих один и тот же идентификатор клиента, что указывает на наличие отношения "один ко многим".

- Благотворительная организация может получать ежегодные взносы своих спонсоров. Как только спонсор вносит ежегодный платеж, он заносится во вторую сущность, которая может хранить данные о взносах за многие годы. Структура этой сущности может быть следующей: `DonorName`, `2001pledge`, `2002pledge`, но возможны и другие варианты.
- Заказ может иметь несколько строк. Первичный ключ сущности `Order` дублируется во вторичном ключе сущности `OrderDetail`. Каждый заказ имеет один элемент в сущности `Order`, но может иметь множество ассоциированных элементов в сущности `OrderDetail`.

Отношение "один к одному"

Отношение "один к одному" связывает две сущности их первичными ключами. Так как первичный ключ должен быть уникальным, с каждой стороны отношения содержится всего один элемент.

Отношения "один к одному" иногда используют для расширения одной сущности за счет создания дополнительной со своими атрибутами. Например, сущность `Employee` (сотрудники) может хранить основную информацию о сотрудниках компании. В то же время более конфиденциальная информация может храниться в отдельной сущности. В этом случае защита может применяться на основе отдельных атрибутов; представление может отображать только избранные атрибуты. Многие организации предпочитают моделировать конфиденциальную информацию с помощью создания отношений "один к одному" между двумя или более сущностями.

Отношения между подтипом и супертипом

Элемент проектирования, использующий отношение "один к одному", на самом деле формирует отношение между супертипом и подтипом. Это отношение связывает одну сущность супертипа с несколькими сущностями подтипов, чтобы расширить состав атрибутов элемента. Сущность супертипа имеет необязательное отношение "один к одному" с каждой из сущностей подтипов.

Такая модель может пригодиться, когда несколько объектов имеют ряд идентичных атрибутов и отличаются только несколькими. К примеру, это может относиться к сущностям клиентов, поставщиков и производителей. Все три сущности имеют одинаковые атрибуты названия и адреса, но каждая из них имеет свой специфический состав остальных атрибутов. Например, только клиенты могут иметь дисконтные карты, и только поставщики имеют атрибуты, связанные с характеристиками договоров на поставку.

Несмотря на то что для поставщиков и клиентов можно использовать отдельные сущности, гораздо лучшее решение предполагает наличие единой сущности `Contact`, где будут храниться общие атрибуты и две отдельные сущности для атрибутов, уникальных для поставщиков и клиентов.

Если партнер является клиентом, то его дополнительная информация заносится в сущность `Customer`; если он поставщик — в сущность `Supplier`. При этом все три сущности (`Contact`, `Customer` и `Supplier`) имеют один и тот же первичный ключ. Один элемент в сущности `Contact` не обязательно может быть связан с одним элементом в сущности `Customer` и(или) с одним элементом в сущности `Supplier` (рис. 2.4).

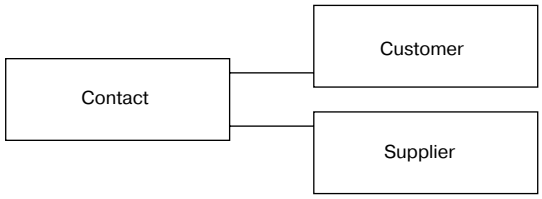


Рис. 2.4. Отношение "один к одному" связывает первичный ключ с другим первичным ключом

Большинство моделей "супертип–подтип" допускает наличие только одного элемента подтипа для каждого элемента супертипа. В то же время в приведенном примере информации о контактах одна и та же организация потенциально может быть одновременно и клиентом, и поставщиком, поэтому элементы могут добавляться одновременно в две сущности.

Модель "супертип–подтип" очень сходна с концепцией наследования классов в объектно-ориентированном анализе. Таким образом, если база данных содержит несколько моделей "супертип–подтип", то база данных может считаться кандидатом на объектно-реляционный стиль вместо реляционного. Объектно-реляционные СУБД для SQL Server можно загрузить с сайта `www.SQLServerBible.com`.

Следует обратить внимание на снижение производительности при использовании такой модели. Операция вставки предполагает добавление элементов в две сущности, а операция извлечения данных должна объединять сущности супертипа и подтипа. Поэтому старайтесь не использовать модели "супертип–подтип" для разделения элементов на категории. Эту модель следует применять только тогда, когда несколько столбцов уникальны для каждого подтипа. При этом при извлечении данных только из одного из подтипов снижается нагрузка на базу данных.

Отношение "многие ко многим"

В отношениях типа "многие ко многим" обе стороны могут быть связаны с множеством элементов противоположной стороны. Отношение "многие ко многим" часто встречается в реальной жизни. Приведу несколько примеров.

- Заказ может содержать несколько товаров, а каждый товар может находиться во множестве заказов.
- В базе данных турфирмы Cape Hatteras Adventures экскурсовод может быть квалифицирован для ведения нескольких туров, а каждый тур может иметь несколько квалифицированных экскурсоводов.
- В базе данных турфирмы клиент может принимать участие в нескольких турах, а каждый тур может иметь множество участников.

Возвращаясь к предыдущему примеру логического моделирования, мы видим, что отношение "многие ко многим" между клиентами и турами моделируется с помощью множественной связи с каждой стороны отношения (рис. 2.5). В данном случае отношение "многие ко многим" необязательно, поскольку как клиенты, так и туры могут существовать независимо друг от друга.

Отношения "один к одному" и "многие ко многим" могут проектироваться из объектов, которые пользователи способны описать и понять. В физической схеме отношение "многие ко многим" не может быть смоделировано с помощью только видимых объектов.

Для разрешения отношения "многие ко многим" используется ассоциированная таблица (рис. 2.6), иногда называемая *стыковочной*. Эта таблица искусственно создает два отношения "один ко многим" между двумя сущностями.

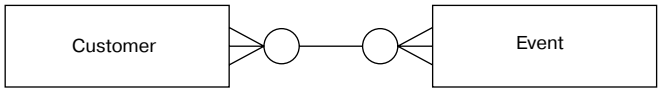


Рис. 2.5. Логическая модель "многие ко многим" содержит множество элементов с каждой из сторон отношения

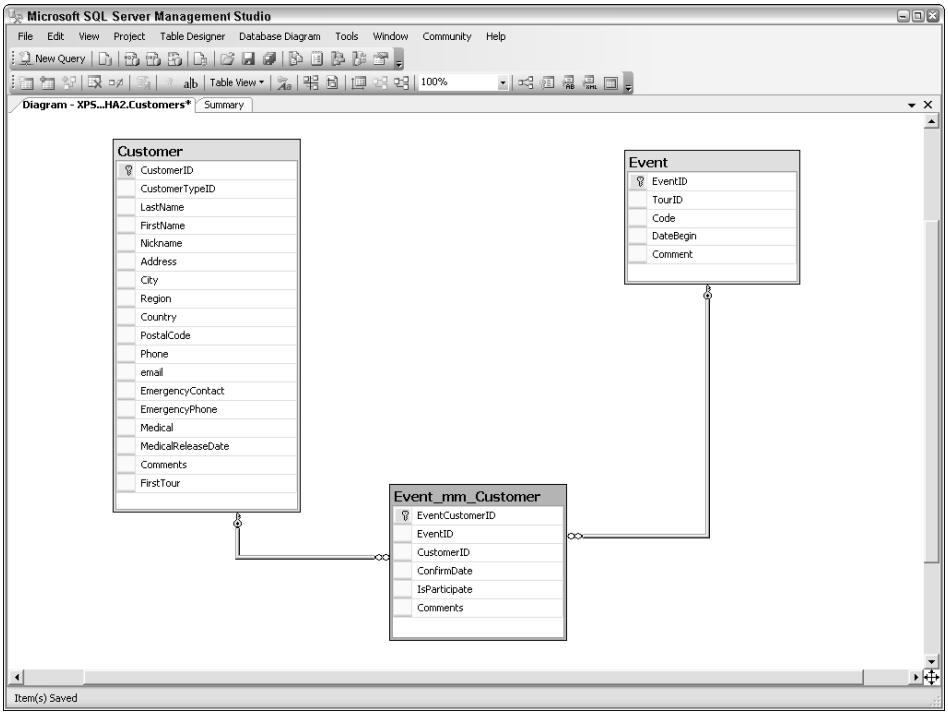


Рис. 2.6. Физическая модель отношения "многие ко многим" включает стыковочную таблицу, имеющую отношения "один ко многим" к обеим сущностям

В некоторых случаях дополнительная информация может описывать отношение "многие ко многим". Такая информация принадлежит к связывающей сущности. Например, в списке материалов отношение одного материала к другому может включать атрибут количества в связывающей сущности, описывающий, какое количество одного материала необходимо для создания второго.

Отношение между каждой основной сущностью и связывающей является обязательным, поскольку это отношение некорректно в отсутствие основного объекта. Если в базе данных турфирмы удалить либо клиента, либо тур, то соответствующий элемент в связывающей сущности станет некорректным.

Сущности категорий

Еще одним типом поддерживающей сущности является сущность *категорий*, иногда называемая *таблицей классификаторов*. Эти сущности обеспечивают целостность домена в смысле способа организации элементов. Отличным примером может служить таблица штатов США. Вместо того чтобы в таблице клиентов атрибут региона мог принимать некорректные

значения (например, `Fl` или `FLA` для штата Флорида), эта таблица связана с таблицей штатов, в которой все эти названия заведомо корректны. Такая целостность данных ускоряет и облегчает поиск и сортировку данных.

Видимые сущности обычно связаны с сущностями категорий отношением "один ко многим"; такое отношение может быть как обязательным, так и нет.

Возвратные отношения

В некоторых случаях отношение устанавливается между двумя элементами одного и того же типа. Приведем несколько примеров.

- Организационная диаграмма представляет сотрудника, отчитывающегося перед другим сотрудником.
- Список материалов детализирует, как один материал получается из других материалов.
- В базе данных семей некоторое лицо имеет отношение к своей матери и отцу.

Все это примеры *возвратных отношений*, чаще всего называемых *рекурсивными*, а иногда *унарными* или *собственными*. Благодаря форме на диаграмме их неформально называют отношениями *слоновьего уха*.

Используя базу данных семей в качестве примера, каждый элемент в сущности `Person` представляет одного человека. Каждый человек имеет (как правило) и мать, и отца, которые также находятся в сущности `Person`. Таким образом, внешние ключи `MotherID` (идентификатор матери) и `FatherID` (идентификатор отца) указывают соответственно на элементы матери и отца в той же сущности.

Так как первичным ключом является `PersonID` (идентификатор человека), а `MotherID` — внешним, устанавливается отношение "один ко многим" (рис. 2.7). Одна мать может иметь несколько детей, однако каждый ребенок имеет одну и только одну мать.

Более сложным примером является список материалов, так как каждый материал может быть создан на основе нескольких исходных и в то же время использоваться для изготовления ряда других материалов на следующей стадии производственного процесса. Это рекурсивное отношение "многие ко многим" проиллюстрировано на рис. 2.8.

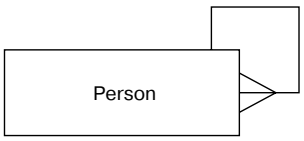


Рис. 2.7. Возвратное, или рекурсивное, отношение представляет собой отношение "один ко многим" в пределах одной сущности

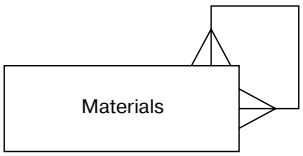


Рис. 2.8. Логическая схема возвратного отношения "многие ко многим" демонстрирует наличие нескольких элементов на каждом конце отношения

Для разрешения отношений "многие ко многим" требуется связывающая сущность. В примере таблицы материалов связывающая сущность `BillOfMaterials` имеет два внешних ключа, каждый из которых указывает на сущность `Material` (рис. 2.9).

Первый внешний ключ указывает на создаваемый материал, а второй — на исходный.

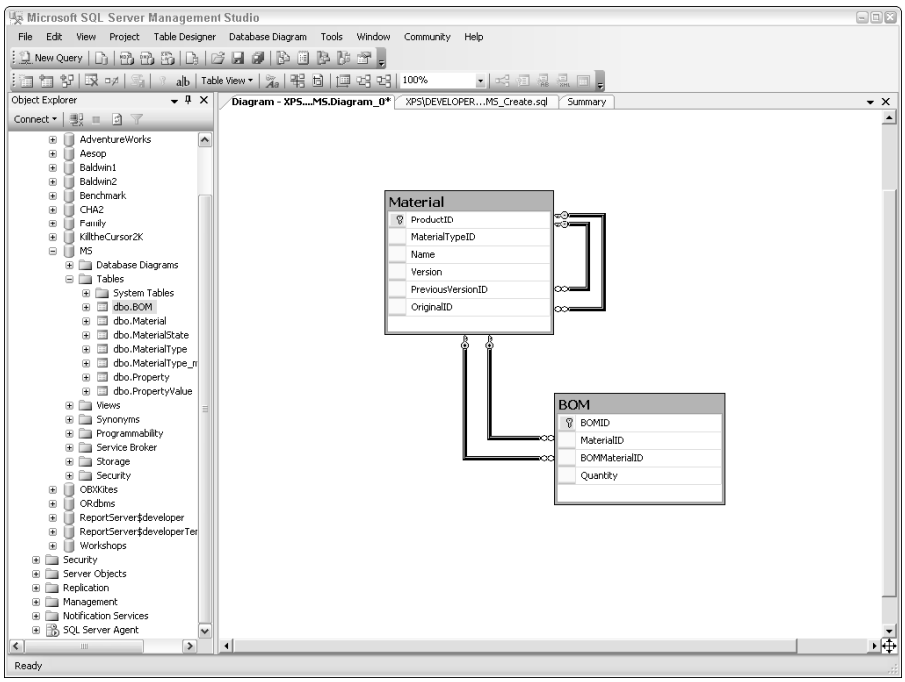


Рис. 2.9. Физическая схема базы данных возвратного отношения "многие ко многим" должна содержать связывающую сущность, как и в обычных отношениях

Нормализация

Самая известная концепция проектирования реляционных баз данных и их нормализации была разработана доктором Эдгаром Коддом. Целью *нормализации* является сокращение или полное устранение аномалий обновления, которые могут возникнуть, если одни и те же фактические данные хранятся в разных частях базы данных.

Нормализация не является ключевым этапом проектирования базы данных, а только средством сокращения аномалий обновления. Как таковая, она больше подходит для баз данных, которые содержат оперативные данные, такие как база данных текущих заказов. Соответственно, базы данных, не содержащие оперативные данные (такие, как база данных отчетов или хранилище данных), не обязательно выиграют от нормализации.

Когда индустрия переходила от модели, основанной на "плоских" файлах, к реляционным моделям данных, нормализацию можно было представить как процесс, начинающийся с "плоского" файла и проходящий постепенно через все нормальные формы, приобретая реляционный вид, в котором максимально сокращены аномалии обновления.

Хотя большинство разработчиков баз данных знакомы с нормализацией, реальные рабочие принципы нормализации зачастую понимаются неверно. Существуют три основных заблуждения относительно нормализации и логической схемы базы данных.

- Наиболее распространенное заблуждение представляет собой мнение о том, что чем больше таблиц, тем выше форма нормализации базы данных. Например, некоторые разработчики верят, что если конкретная схема базы данных в третьей форме нормализации содержит 12 таблиц, то в четвертой она уже будет содержать 16 таблиц.

- Еще одним заблуждением является рассмотрение нормальных форм как последовательных, т.е. мы начинаем с ненормализованной базы, затем корректируем ее для первой нормальной формы, затем для второй и т.д.
- И наконец, заблуждением является уверенность в том, что для каждой задачи существует единственная правильная нормализованная форма. Опытные проектировщики данных способны сразу увидеть несколько возможных конфигураций сущностей и моделей логической схемы.

Принципы проектирования сущностей и атрибутов

Ключевым моментом в проектировании схемы, которая свободна от аномалий обновления, является обеспечение того, чтобы каждый единичный факт из реальной жизни моделировался одной точкой данных в базе. Существуют три принципа, которые определяют единичную точку данных.

- Каждая сущность базы данных должна описывать одну "вещь".
- Между атрибутами и фактами должно существовать отношение "один к одному", т.е. каждый атрибут должен представлять один факт, и каждый факт должен быть представлен одним атрибутом. Например, "универсальный ключ", который шифрует несколько фактов в одном атрибуте, нарушает этот принцип; то же касается моделей, в которых для определения одного факта комбинируется два и более атрибута.
- Каждый атрибут должен описывать свою сущность и никакую другую, связанную с ней.

Нормальные формы

Нормализация обычно определяется в терминах нормальных форм. Каждая из нормальных форм описывает возможные ошибки в модели сущностей и атрибутов и формулирует правило, способное устранить эти ошибки. Таким образом, нормальные формы аналогичны тому, как правила грамматики способны выявить ошибки в структуре предложений.

При создании логической модели *нормализация* представляет собой математический метод оценки качества отношений в модели данных. В противоположность реляционной модели базы данных "плоский" файл или ненормализованные модели данных подвержены проблемам при обновлении данных, обычно выраженных дублированием информации. Каждая следующая форма нормализации устраняет одну из проблем, связанных с "плоскими" файлами.

Нормализованная модель базы данных имеет следующие преимущества перед ненормализованными:

- повышенная целостность данных делает невозможным двойное хранение одних и тех же данных;
- сокращение споров за блокировку ресурсов и улучшенная многопользовательская конкуренция;
- меньшие размеры файлов.

Модель данных не начинает с ненормализованного состояния и не обязательно затем проходит через все нормальные формы. Обычно проектировщик изначально разрабатывает логическую схему не менее чем в третьей нормальной форме и может принять решение привести часть схемы к более высокой форме нормализации.

Простота и нормализация

Простота не обязательно предполагает нарушение целостности данных. Формы нормализации так же связаны с проектированием базы данных, как правила грамматики с письмом. Хороший текст не должен нарушать правила грамматики. Продолжая мысль, следует сказать, что главный принцип, сформулированный в книге Странка и Уайта *Элементы стиля* (Будь точен!), не менее важен для проектирования баз данных, чем для письма.

Первая нормальная форма (1НФ)

Первая нормальная форма подразумевает, что данные объединены в сущности и удовлетворяют следующим условиям.

- **Каждая единица данных представлена в скалярном атрибуте**. Согласно Мерриаму–Вебстеру, *скалярным* называется значение, которое можно представить точкой на шкале.

  Каждый атрибут должен содержать одну единицу данных, и каждая единица данных должна заполнять один атрибут. Модели, в которых в одном атрибуте объединено несколько элементов информации, нарушают принципы первой нормальной формы. Аналогично, если несколько атрибутов должны быть скомбинированы, чтобы получить один элемент данных, значит, модель атрибута некорректна.

- **Все данные должны быть представлены уникальными атрибутами**. Каждый атрибут должен иметь уникальное имя и уникальное назначение. Сущность не должна содержать повторяющиеся атрибуты. Если некоторый атрибут повторяется или сущность охватывает чересчур широкую предметную область, значит, объект неправильно спроектирован.

  Модель, в которой атрибуты повторяются (например, сущность заказа, содержащая атрибуты `товар1`, `товар2` и `товар3` для хранения нескольких позиций заказа), нарушает первую нормальную форму.

- **Все данные должны содержаться в уникальных элементах**. Если модель сущности требует или допускает дублирование элементов, значит, она нарушает первую нормальную форму.

В качестве примера первой нормальной формы рассмотрим список базовых лагерей и туров базы данных туристической фирмы Cape Hatteras Adventures. В табл. 2.3 представлены данные о базовых лагерях, которые нарушают первую нормальную форму. Повторяющийся атрибут тура не является уникальным.

Таблица 2.3. Нарушение первой нормальной формы

BaseCamp	Tour1	Tour2	Tour3
Ashville	Appalachian Trail	Blue Ridge Parkway Hike	
Cape Hatteras	Outer Banks Lighthouses		
Freeport	Bahamas Dive		
Ft. Lauderdale	Amazon Trek		
West Virginia	Gauley River Rafting		

Чтобы исправить эту модель и привести ее к первой нормальной форме, сведем повторяющиеся группы атрибутов тура в единый уникальный атрибут (табл. 2.4). После этого мы переместим все повторяющиеся значения в уникальные элементы. Сущность `BaseCamp` содержит

уникальный элемент для каждого базового лагеря, а атрибут BaseCampID (идентификатор базового лагеря) сущности Tour ссылается на первичный ключ сущности BaseCamp.

Таблица 2.4. База данных, удовлетворяющая первой нормальной форме

Сущность Tour		Сущность BaseCamp	
BaseCampID (внешний ключ)	**Tour**	**BaseCampID (первичный ключ)**	**Name**
1	Apalachian Trail	1	Ashville
1	Blue Ridge Parkway Hike	2	Cape Hatteras
2	Outer Banks Lighthouses	3	Freeport
3	Bahamas Dive	4	Ft. Lauderdale
4	Amazon Trek	5	West Virginia
5	Gauley River Rafting		

Еще одним примером структуры данных, которая требует приведения к первой нормальной форме, является корпоративный каталог продукции, содержащий подразделение, модель, цвет, размер и прочие характеристики в единой сущности. Мне даже доводилось видеть каталоги, которые были настолько сложные, что содержали числа, указывающие на синтаксис последующих чисел.

С теоретической точки зрения этот тип модели некорректен, поскольку атрибут не является скалярным значением. С практической точки зрения в нем есть следующие проблемы.

- Использование одного или двух чисел для каждого элемента данных указывает на то, что база данных скоро преодолеет диапазон допустимых значений.
- Базы данных не индексируются на основе внутренних значений строки, таким образом, любой поиск потребует сканирования всей таблицы, т.е. перебора всех ее строк.
- Бизнес-правила тяжело программировать и поддерживать.

Сущности с нескалярными атрибутами нужно полностью перепроектировать таким образом, чтобы каждый отдельный атрибут данных имел свой собственный атрибут.

Вторая нормальная форма (2НФ)

Вторая нормальная форма гарантирует, что каждый атрибут на самом деле является атрибутом этой сущности. Это вопрос зависимости. Каждый атрибут должен требовать свой первичный ключ, в противном случае он не будет принадлежать базе данных.

Если первичный ключ сущности представляет собой одно значение, все не так сложно. Составные первичные ключи могут иногда привести к противоречию со второй нормальной формой, если атрибуты не зависят от каждого из атрибутов первичного ключа. Если атрибут зависит только от одного атрибута первичного ключа и не зависит от другого, то существует только частичная зависимость, что нарушает вторую нормальную форму.

Примером базы данных, нарушающей вторую нормальную форму, является база данных телефонных номеров базовых лагерей туристической компании Cape Hatteras Adventures, добавленная в сущность BaseCampTour (табл. 2.5). Предположим, что первичный ключ представляет собой объединение атрибутов BaseCamp и Tour и что телефонные номера закреплены именно за лагерями, а не за турами.

Таблица 2.5. Нарушение второй нормальной формы

BaseCamp (ПК)	Tour (ПК)	BaseCampPhoneNumber
Ashville	Appalachian Trail	828-555-1212
Ashville	Blue Ridge Parkway Hike	828-555-1212
Cape Hatteras	Outer Banks Lighthouses	828-555-1213
Freeport	Bahamas Dive	828-555-1214
Ft. Lauderdale	Amazon Trek	828-555-1215
West Virginia	Gauley River Rafting	828-555-1216

Проблема данной модели состоит в том, что телефонный номер является атрибутом только базового лагеря, но не тура, таким образом, он только частично зависит от первичного ключа сущности. (Более существенной проблемой является то, что составной первичный ключ неоднозначно определяет базовый лагерь.)

Очевидной практической проблемой этой модели является то, что обновление телефонных номеров требует обновления множества кортежей, в противном случае существует риск, что одному лагерю будут соответствовать два телефона.

Решение этой проблемы состоит в том, чтобы удалить из сущности частично зависимый атрибут и создать отдельную сущность с уникальным первичным ключом для базового лагеря (табл. 2.6). Эта новая сущность будет подобающим местонахождением зависимого атрибута.

Таблица 2.6. Приведение ко второй нормальной форме

Сущность Tour		Сущность BaseCamp	
BaseCamp (ПК)	Tour (ПК)	BaseCamp (ПК)	PhoneNumber
Ashville	Appalachian Trail	Ashville	828-555-1212
Ashville	Blue Ridge Parkway Hike	Cape Hatteras	828-555-1213
Cape Hatteras	Outer Banks Lighthouses	Freeport	828-555-1214
Freeport	Bahamas Dive	Ft.Lauderdale	828-555-1215
Ft.Lauderdale	Amazon Trek	West Virginia	828-555-1216
West Virginia	Gauley River Rafting		

Атрибут `PhoneNumber` теперь полностью зависит от первичного ключа сущности. Каждый телефонный номер хранится только в одном месте, при этом отсутствует какая-либо частичная зависимость.

Третья нормальная форма (3НФ)

Третья нормальная форма проверяет наличие транзитивных зависимостей. *Транзитивная зависимость* аналогична частичной в том, что они обе относятся к атрибутам, которые не в полной мере зависят от первичного ключа. Зависимость называется транзитивной, когда один атрибут зависит от другого, который, в свою очередь, зависит от первичного ключа.

Как и вторая нормальная форма, третья разрешается перемещением независимого атрибута в новую сущность.

Продолжая работать с примером базы данных турфирмы Cape Hatteras Adventures, назначим каждому базовому лагерю ответственного экскурсовода. Атрибут `BaseCampGuide` при-

надлежит сущности BaseCamp, однако если имя экскурсовода сопровождает какая-либо дополнительная информация, третья нормальная форма нарушается (табл. 2.7).

Таблица 2.7. Нарушение третьей нормальной формы

Сущность BaseCamp			
BaseCamp (ПК)	**BaseCampPhoneNumber**	**LeadGuide**	**DateOfHire**
Ashville	828-555-1212	Jeff Davis	1/5/1999
Cape Hateras	828-555-1213	Ken Frank	15/4/1997
Freeport	828-555-1214	Dab Smith	7/7/2001
Ft.Lauderdale	828-555-1215	Sam Wilson	1/1/2002
West Virginia	828-555-1216	Lauren Jones	1/6/2000

Атрибут DateOfHire (дата приема на работу) описывает экскурсовода, а не базовый лагерь, следовательно, он не напрямую зависит от первичного ключа сущности. Эта зависимость является транзитивной через атрибут LeadGuide.

Создание сущности Guide и перемещение атрибутов экскурсовода в нее устраняет нарушение третьей нормальной формы, а также проясняет логическую модель (табл. 2.8).

Таблица 2.8. Приведение к третьей нормальной форме

Сущность BaseCamp		Сущность LeadGuide	
BaseCamp (ПК)	**LeadGuide**	**LeadGuide (ПК)**	**DateOfHire**
Ashville	Jeff Davis	Jeff Davis	1/5/1999
Cape Hatteras	Ken Frank	Ken Frank	15/4/1997
Freeport	Dab Smith	Dab Smith	7/7/2001
Ft.Lauderdale	Sam Wilson	Sam Wilson	1/1/2002
West Virginia	Lauren Jones	Lauren Jones	1/6/2000

Если сущность имеет хороший первичный ключ, все ее атрибуты скалярны и полностью зависят от первичного ключа, то логическая модель находится в третьей нормальной форме. Большинство баз данных останавливается именно на третьей нормальной форме.

Дополнительные нормальные формы решают проблемы за счет более сложной логической модели. Если вам интересно работать с головоломными проблемами моделирования и создавать креативные решения, то вам поможет знание дополнительных нормальных форм.

Нормальная форма Бойса–Кодда (BNCF)

Нормальная форма Бойса–Кодда занимает положение между третьей и четвертой нормальными формами и решает проблемы сущностей, которые могут иметь два набора первичных ключей. Эта форма предполагает, что в данном случае сущность должна быть разбита на две — по одной для каждого первичного ключа.

Четвертая нормальная форма (4НФ)

Четвертая нормальная форма решает проблемы, создаваемые сложными составными первичными ключами. Если два независимых атрибута сведены для формирования первичного ключа с третьим и при этом оба атрибута однозначно не определяют элемент без третьего, то модель нарушает четвертую нормальную форму.

В качестве примера представим следующую ситуацию.

1. В качестве составного первичного ключа используются атрибуты `BaseCamp` (базовый лагерь) и `LeadGuide` (ответственный экскурсовод).
2. Атрибуты `Event` и `Guide` сведены вместе в качестве первичного ключа.
3. Так как оба ключа используют экскурсовода, все три сведены в единую сущность.

Приведенный пример нарушает четвертую нормальную форму.

Четвертая нормальная форма используется, чтобы помочь идентифицировать сущности, которые могут быть разбиты на несколько отдельных. Обычно это касается только сложных составных ключей, в которых сведено несколько независимых атрибутов.

Пятая нормальная форма (5НФ)

Пятая нормальная форма реализует метод проектирования сложных отношений, включающих в себя множество (три и более) сущностей. Если *тройственное* отношение правильно построено, оно не нарушает пятую нормальную форму. Строгость таких отношений может быть как однозначной, так и множественной. Тройственное отношение в таком случае составляют множество связанных сущностей.

В качестве примера тройственного отношения можно рассмотреть производственный процесс, включающий оператора, машину и список материалов. С определенной точки зрения это должна быть операционная сущность, содержащая три внешних ключа. В качестве альтернативы можно рассмотреть вариант тройственного отношения с дополнительными атрибутами.

Подобно отношению “многие ко многим” между двумя сущностями, тройственное отношение требует разделения сущностей в модели физической схемы, что приведет к преобразованию отношения “многие ко многим” в несколько искусственных отношений “один ко многим”. Однако в данном случае связывающая сущность будет иметь три или более внешних ключа.

В таком сложном отношении пятая нормальная форма требует, чтобы каждая сущность, участвующая в разделенном тройственном отношении, оставалась целостной, без какой-либо потери данных.

Основной задачей реляционных баз данных является моделирование реальности. Здесь поддержание целостности данных важно как с теоретической, так и с практической стороны процесса разработки баз данных.

Одним из ключевых вопросов поддержания целостности базы данных является обучение владельцев данных тому, что они сами должны оценивать целостность и не только “владеть” задачей и проектом, но и поддерживать целостность данных. Вопрос целостности данных не возникает случайно, сам по себе. Им нужно заниматься с первого дня работы над проектом.

Одним из самых сложных факторов целостности данных являются старые данные, переносимые из устаревших систем. Если эти старые данные не удовлетворяют условиям целостности реляционных данных, то могут возникнуть серьезные проблемы.

Ответственный за данные старой системы должен чувствовать персональную ответственность за успех нового проекта. Нужно дать разработчикам старой системы почувствовать, что они отвечают за целостность данных и вовлечены в разработку нового проекта. Это лучше, чем представить им новый проект как готовый продукт и дать почувствовать себя оттесненными в сторону. Нужно позволить им самим ставить себе задачи и помогать их решать самым наилучшим образом.

Реляционная алгебра

Реляционные базы данных по своей природе сегментируют данные на несколько узких, но длинных таблиц. Редко когда в одной таблице содержатся данные, имеющие самостоятельное значение. Таким образом, важной задачей для разработчиков является слияние данных из множества таблиц. Теория слияния наборов данных называется *реляционной алгеброй*; она была разработана в 1970 году Эдгаром Коддом.

Реляционная алгебра содержит восемь *реляционных операторов*.

- **Ограничение**. Возвращает строки, которые удовлетворяют заданному условию.
- **Проекция**. Возвращает из набора данных заданные столбцы.
- **Произведение**. Реляционное умножение возвращает все возможные комбинации данных из двух наборов.
- **Слияние**. Реляционное сложение и вычитание, которое объединяет по вертикали две таблицы, помещая одну над другой и выравнивая столбцы.
- **Пересечение**. Возвращает строки, общие для двух наборов данных.
- **Разность**. Возвращает строки, уникальные для одного набора данных.
- **Объединение**. Возвращает объединение двух таблиц по горизонтали, сопоставляя их строки по общим данным.
- **Деление**. Возвращает строки, точно совпадающие в двух наборах данных.

В дополнение, в качестве метода, поддерживающего реляционную алгебру, SQL предлагает следующее.

- **Подзапросы**. Это сходный с объединением, но более гибкий механизм. Результаты подзапроса могут использоваться вместо выражения, списка или набора данных во внешнем запросе.

На формальном языке реляционной алгебры:

- таблица или набор данных называется *отношением* или *сущностью*;
- строка называется *кортежем*;
- столбец называется *атрибутом*.

Реляционную алгебру мы применим на практике в главах 9 и 10.

Резюме

Реляционная теория существует уже более трех десятилетий и была более четко определена, когда производители систем управления базами данных поработали над ее расширениями, а теоретики более четко определили задачу представления реальности с помощью структур данных. В то же время оригинальная работа доктора Кодда остается фундаментом проектирования и реализации реляционных баз данных.

Умение проектировать реляционные базы данных является фундаментальным умением любого разработчика. Чтобы состояться как разработчик, нужно четко представлять себе весь набор доступных шаблонов. В следующей главе мы перейдем к вопросу двухмерных реляционных и трехмерных объектно-ориентированных баз данных.