Реляционные базы данных.

Основные понятия БД

Реляционные БД это средство для рационального и эффективного хранения информации. БД обеспечивают надежную защиту данных от случайной потери или порчи, экономно используют ресурсы (как людские, так и технические) и снабжена механизмом поиска информации удовлетворяющим разумным требованиям к производительности.

При разработки БД используют средства систем управления базами данных СУБД.

Реляционная БД — это реализация реляционной модели данных на физическом уровне.

Необходимо четко различать модель данных и базу данных. На стадии проектирования невозможно полностью изолироваться от ограничений, налагаемых средой разработки, в то время как в основу проекта рекомендуется закладывать максимально «чистую» модель.

Что такое БД

Понятие БД может обозначать как отдельный набор данных (например список телефонов), так и гораздо более сложную систему (например SQL Server).

Для реляционных БД нет прямых аналогий в реальном мире, большинство их предназначено для моделирования некоторых аспектов реальности. Именно этот кусочек «реального» мира, другими словами, аспект реальности, мы будем называть предметной областью. Предметная область имеет сложную структуру и не упорядочена. Для успешной реализации проекта необходимо ограничить проектируемую систему. Выбрать объекты и связи между ними. Только после этого возможно оценить масштабы проектируемой системы.

Модель данных — это концептуальное описание предметной области. Она включает определение сущностей и их атрибутов: например сущность покупатель имеет атрибут имя и адрес. Для сущности определяем ограничения. Модель данных не содержит ссылок и указаний на физическую модель самой системы.

Предметная область это определенная часть реального мира.

Модель данных это концептуальное описание предметной области.

Схема БД содержит описание модели данных, используемое БД.

БД представляет собой реализацию схемы БД и модели данных на физическом уровне.

Механизм СУБД не входит в БД.

Приложение состоит из форм и отчетов, с которыми работает пользователь, данные для отчетов берутся из БД и данные из форм поступают в БД через СУБД.

Определение физической модели — создаваемых таблиц и представлений, называется схемой базы данных. Схема — это перевод концептуальной модели в физическое представление, осуществляемое средствами СУБД. Схема — это понятие, относящееся к концептуальному, а не к физическому уровню. Это все та же модель данных, описываемая в терминах, используемых механизмом СУБД (database engine) — таблицы, триггеры и т. п. Механизм СУБД хорош и тем, что при его использовании не приходится иметь дело с физической реализацией модели; до известного предела вы можете игнорировать такие сущности, как би-деревья и листовые узлы.

Структура и данные вместе составляют то, что я обычно называю БД. БД содержит физические таблицы, представления, запросы, хранимые процедуры, а также правила, используемые механизмом СУБД для зашиты данных.

В понятие ≪БД≫ не входят приложение, состоящее из форм и отчетов, с которыми работают пользователи, а также средства, обеспечивающие связь между серверной и клиентской частями клиент-серверных приложений.

Инструменты для работы с БД

Механизм СУБД

Механизм базы данных — это специальные средства, предназначенные для физического манипулирования данными: хранением их на диске и извлечением по запросу. SQL Server использует клиент-серверную архитектуру и предназначен для создания систем, от средних до больших. Он прекрасно масштабируется и может поддерживать несколько тысяч пользователей, работающих с важными приложениями.

Объектная модель доступа к данным

Объектная модель доступа к данным представляет собой своего рода ≪промежуточный слой≫ между средой программирования и механизмом СУБД. Она содержит набор объектов, обладающих свойствами и методами, которыми можно манипулировать в коде.

Средства для разработки клиентской части приложений

SQL Server берет на себя манипулирование данными на физическом уровне, однако необходимо ему дать четкие указания, каким образом эти данные должны быть структурированы. Следует использовать интерактивные средства — они гораздо удобнее, и к тому же экономят время.

После того как проектирование базы данных на физическом уровне будет завершено, вам потребуются инструменты для создания форм и отчетов, с которыми будут работать пользователи.

Реляционная модель

Реляционная модель основывается на математических принципах, вытекающих непосредственно из теории множеств и логики предикатов. Эти принципы впервые были применены в области моделирования данных в конце 60-х гг. доктором Е.Ф. Коддом, в то время работавшим в IBM, а впервые опубликованы — в 1970 г. Реляционная модель определяет способ представления данных (структуру данных), методы зашиты данных (целостность данных), а также операции, выполняемые с данными (манипулирование данными).

Реляционная модель не единственный метод хранения и манипулирования данными. Существуют альтернативные варианты: иерархическая, сетевая, а также звездообразная модели данных. У каждой из них свои преимущества при решении задач определенного типа.

В общих чертах основные принципы реляционных систем баз данных можно сформулировать так:

- Все данные на концептуальном уровне представляются в виде упорядоченной организации, определенной в виде строк и столбцов и называемой отношением.
- Все значения являются скалярами. Это означает, что для любой строки и столбца любого отношения существует одно и только одно значение.
- Все операции выполняются над целым отношением, и результатом выполнения этих операций также является целое отношение. Этот принцип называется замыканием.

В рамках реляционной модели данные представлены в виде отношения на концептуальном уровне; однако при этом совсем не дается никаких указаний, каким образом данные будут реализованы на физическом уровне.

В рамках реляционной модели данные представлены в виде отношения на концептуальном уровне; однако при этом совсем не дается никаких указаний, каким образом

данные будут реализованы на физическом уровне.

Принцип замыкания заключается в том, что и базовые таблицы, и результаты операций над ними на концептуальном уровне представляются как отношения. Он позволяет непосредственно использовать результаты одной операции в качестве исходных данных для выполнения другой. Таким образом, SQL Server дает возможность использовать результаты одного запроса для составления нового. Этот принцип реализует в области разработки баз данных функциональность, аналогичную подпрограммам в процедурном программировании — возможность инкапсуляции сложных или часто повторяющихся операций для повторного использования.

Термины, используемые в реляционной теории

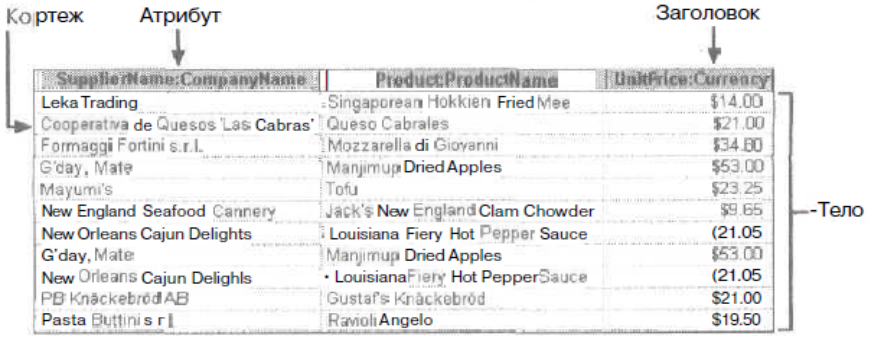


Рис. Компоненты отношения

Отношением называется вся структура в целом. Каждая строка, содержащая данные, является кортежем. Строго говоря, каждая строка является n-кортежем. Число кортежей в отношении определяет мощность отношения. В примере мощность отношения равна 11. Каждый столбец отношения называется атрибутом. Число атрибутов в отношении определяет размерность этого отношения, в примере она равняется трем.

Каждое отношение можно разделить на две части — заголовок и тело. Тело отношения состоит из кортежей, в то время как заголовок не имеет более мелких компонентов структуры. Название каждого из атрибутов состоит из двух терминов, разделенных двоеточием (например, UnitPrice:Currency). Первая часть названия — непосредственно имя атрибута, вторая — имя домена. Домен атрибута — это ≪вид≫ данных, которые представляет данный атрибут (в приведенном примере — валюта). Понятие ≪домен≫ не эквивалентно понятию ≪тип данных≫. Различие между этими двумя понятиями будет подробно обсуждаться далее в этой главе. На практике домен в заголовках часто не указывается.

Тело отношения состоит из неупорядоченного набора кортежей (число кортежей может быть любым, от 0 и более). Во-первых, отношение не упорядочено. Понятие ≪номер строки≫ не применимо к отношению. Для отношений не существует никакого внутреннего порядка. Во-вторых, отношение может иметь нулевое число кортежей (это так называемое пустое отношение, которое, тем не менее, является отношением). В третьих, отношение представляет собой набор. Элементы в этом наборе по определению уникально идентифицируемые. Поэтому чтобы таблица являлась отношением, каждая ее строка должна быть уникально идентифицируемой, записи в ней не должны повторяться.

Для SQL Server атрибут ≪превращается≫ в поле, а кортеж, соответственно — в запись. Эти соотношения практически взаимно однозначны; однако нужно помнить, что отношения существуют на концептуальном уровне, в то время как наборы записей и наборы результатов — на физическом.

Модель данных

Модель данных, то есть концептуальное описание предметной области — самый абстрактный уровень проектирования баз данных. Элементами описания модели данных

являются сущности, атрибуты, домены и отношения.

Сущности

Сущность — это нечто такое, о чем нужно хранить информацию в разрабатываемой системе.

Составить первоначальный список сущностей не составляет труда. Когда вы ведете деловую беседу с заказчиком, большинство существительных и часть глаголов, используемых в разговоре, и есть кандидаты на эту роль.

Как правило, чтобы составить список сущностей, одной беседы с заказчиком недостаточно. Нужно просмотреть как можно больше документов, имеющих отношение к предметной области. Заполняемые бланки, отчеты, инструкции для персонала — это настоящая сокровищница, из которой и следует извлекать ≪кандидаты≫ в сущности. Причем анализировать документы следует предельно внимательно.

После того как составлен первоначальный вариант списка сущностей, следует обязательно проверить его на полноту и связность. Кроме того, нужно выявить ≪дубли≫, то есть повторяющиеся сущности, и сущности, на самом деле разные, но ошибочно представленные в списке как одна.

Может оказаться, что некоторые подтипы сущностей обладают одинаковым набором атрибутов. Тогда лучше определить атрибуты над-типа, а не моделировать подтипы как отдельные сущности.

Большинство сущностей моделируют объекты или события реального мира, примерами могут служить клиенты, товары, или звонки в службу продаж. Это конкретные сущности.

Сущности также могут моделировать и абстрактные понятия. (торговый агент отвечает за определенного клиента, или некий студент записан на определенный курс лекций).

Иногда необходимо моделировать только сам факт наличия отношения, иногда — хранить дополнительную информацию об этом отношении (дата возникновения данного отношения, а также его дополнительные характеристики).

Атрибуты

Атрибуты сущности это записи об определенных параметрах каждой из сущностей. (сущность Покупатель, атрибуты: фамилия, имя, род деятельности или сущность «Звонок в службу тех поддержки», атрибутами являются: кто звонил, когда, вид проблемы, удалось ли решить проблему).

Определение атрибутов, которые нужно включить в разрабатываемую модель — это семантический процесс. Нужно основываться на том, что реально означают хранимые данные и как они будут использоваться. (Пример адрес: строка или разделенные по полям отдельно).

Почти всегда невозможно сделать общие предположения о том, как следует моделировать конкретный вид данных. Сложная структура одних и тех же данных оправдана для одних задач и не пригодна для других.

Проектируя БД, разработчики пользуются общими стратегическими подходами.

Первое правило: начните с результата и старайтесь по возможности упрощать модель, а не усложнять ее.

При проектировании баз данных нужна определенная гибкость реализуемой системы, заложенная в ней возможность не просто давать ответы на вопросы, которые пользователь задает сегодня, но и предвидеть, какую информацию он захочет получить завтра.

Следует уделить особое внимание вопросам, которые пользователь мог задать, если бы знал, что это принципиально возможно. (Важно при проектировании БД

автоматизирующих то, что раньше делалось вручную.)

Один из признаков профессионализма разработчика — скрупулезный анализ потенциальных вопросов, ответ на которые могут захотеть получить пользователи разрабатываемой системы. От неопытных аналитиков часто слышишь, что пользователи сами не знают, чего хотят. И это вполне естественно! В том и заключается работа аналитика — помочь пользователю понять, чего же он хочет.

Но стремление сделать систему как можно более гибкой неизбежно приводит к ее усложнению.

Может наступить момент, когда все преимущества автоматизации сойдут на нет просто потому, что разрабатываемая система станет чересчур сложной.

Второе правило:

Следует выявлять исключения, с которыми предстоит иметь дело: во-первых, при разработке системы важно определить все исключения, и во-вторых, заложить в нее обработку максимально возможного числа исключений (то есть тех, которые вы в состоянии обработать, не запутав при этом пользователя).

Большая гибкость достигается, как правило, за счет увеличения сложности. Конечно, нужно стремиться выявить и обработать как можно большее количество исключений, Но иной раз стоит остановиться и подумать: а стоит ли вообще заниматься этим исключением? Если его обработка чересчур усложнит систему, или маловероятно, что пользователи когда-либо с этим исключением встретятся — то нет.

Домены

Домен определяет ≪вид≫ данных, которые представляет данный атрибут. Если дать более четкое определение, то домен — это набор всех допустимых значений, которые может содержать данный атрибут.

Понятие ≪домен≫ часто путают с понятием ≪тип данных≫. Необходимо четко различать эти два понятия. Тип данных — это физическая концепция, а домен — логическая. Например, ≪целое число≫ — это тип данных, а ≪возраст≫ — это домен.

Понятие домена намного шире понятия ≪тип данных≫, поскольку определение домена включает в себя более детальное описание допустимых значений данных.

Домен — это тип данных и логические правила, определенные для данной сущности, но логические правила — это один из механизмов реализации целостности данных, а отнюдь не элемент их описания.

Связи

Кроме атрибутов каждой сущности модель данных должна определять связи между сущностями. На концептуальном уровне связи представляют собой простые ассоциации между сущностями.

Например, утверждение «Покупатели покупают продукты» указывает, что между сущностями «Покупатели» и «Продукты» существует связь, и такие сущности называются участниками этой связи. Число участников определяет размерность связи.

Большинство связей — двойные, в них два участника.

Существуют и другие виды связей, например, на существование неявной тройной связи указывает утверждение ≪Сотрудники фирмы продают товары покупателям≫.

Существуют связи, в которых сущность является участником связи с самой собой. Такую связь часто называют связью типа ≪спецификация товаров≫ и используют для представления иерархических структур.

Существует несколько типов связей между двумя сущностями: это связи ≪один к одному≫, ≪один ко многим≫ и ≪многие ко многим≫.

Связи ≪один к одному≫ встречаются достаточно редко, в основном, между сущностями надтипов и подтипов.

Связи ≪один ко многим≫ более часты. ≪В счете перечислено множество товаров≫, ≪Торговый агент выписывает много счетов≫

Связи ≪многие ко многим≫, хотя и не столь широко распространены, как ≪один ко многим≫, также не редкость. (Пример: покупатель покупает товары, студент изучает предметы). Связи ≪многие ко многим≫ невозможно непосредственно реализовать в реляционной модели, однако их косвенная реализация вполне проста и однозначна.

Участие каждой сущности в определенной связи может быть частичным или полным. Если существование данной сущности полностью определяется ее участием в связи, то такое участие будет полным, в противном случае — частичным.

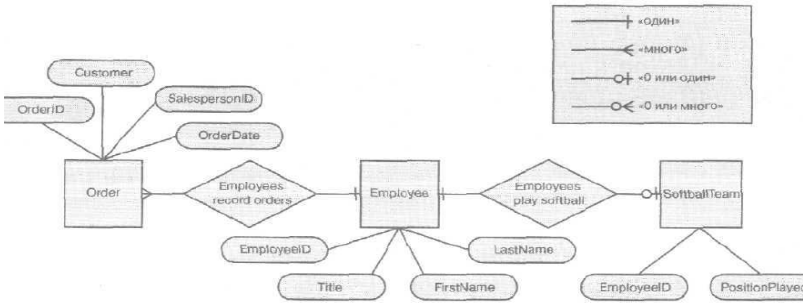
Схема должна содержать верные определения связей для каждой сущности на протяжении всего срока эксплуатации системы.

Диаграмма ≪сущности — связи≫

В 1976 г. Питер Пин Шань Чен (Peter Pin Shan Chen) внес существенный вклад в теорию моделирования данных, разработав модель ≪сущности — связи≫, в которой реализовано описание данных в терминах сущностей, атрибутов и связей.

Он предложил новый метод построения диаграмм — диаграммы ≪сущности — связи≫

На диаграммах ≪сущности — связи≫ сущности изображаются в виде прямоугольников, атрибуты — эллипсов, а отношения — ромбов.



Обычно разработчики баз данных при проектировании системы концентрируют внимание либо на сущностях данной модели и связях между ними, либо на атрибутах конкретной сущности, но не на том и том одновременно.

Структура базы данных

Начальная стадия проектирования моделей реляционных баз данных — создание структуры отношений. Модель максимально корректно и полно описывает предметную область и минимизировать избыточность данных.

Избыточность данных есть зло не только потому, что приводит к бесполезному расходованию ресурсов, она еще и осложняет жизнь тем, кто работает с базой.

Дублированные данные могут присутствовать в нескольких отношениях.

Способность модели данных отвечать на поставленные вопросы определяется прежде всего полнотой хранящихся в ней данных (никакая база данных не может обеспечить пользователей той информацией, которой не содержит), и только во вторую очередь — ее структурой. Но легкость, с которой можно получить необходимые данные, во всех случаях определяется исключительно структурой базы. Принципиально важно, что собрать в единое целое информацию из отдельных атрибутов и отношений достаточно легко, а вот провести дальнейшую детализацию уже занесенной в базу информации очень сложно.

Избегать дублирования одной и той же информации во множестве полей.

Устранить избыточность и упростить структуру данных, чтобы облегчить их получение — вот главные ориентиры при построении модели.

Основные принципы

Принципы нормализации, являются инструментом контроля структуры данных.

Нормальные формы определяют возрастающую строгость правил, которым подчиняются структуры отношений. Мы рассмотрим шесть таких форм. Каждая последующая форма расширяет предыдущую, устраняя при этом возможность возникновения аномалий обновления определенного типа.

Нормальные формы не являются руководством для создания ≪правильной≫ модели данных. Модель данных может быть совершенной с точки зрения теории нормализации, но созданная на ее основе реальная база данных будет выдавать требуемые результаты столь медленно и неуклюже, что ее просто не удастся использовать. Если модель данных нормализована (то есть удовлетворяет принципам, принятым для реляционных структур) шансы получить в результате эффективную модель данных увеличиваются.

Декомпозиция без потерь

Реляционная модель позволяет различным образом соединять отношения, связывая их через атрибуты. Процесс получения полностью нормализованной модели данных включает в себя устранение избыточности. Для этого отношение, содержащее избыточные данные, разбивают на несколько других отношений. Нужно сделать это так, чтобы получившиеся в результате отношения можно было бы вновь соединить и получить точную копию структуры и данных исходного отношения.

Ключи-кандидаты и первичные ключи

Мы знаем, что содержимое отношения это неупорядоченное множество, состоящее из 0 или более кортежей, и каждый элемент множества кортежей должен быть уникален. В таком случае для любого отношения должна существовать комбинация атрибутов, однозначно определяющая каждый кортеж. Такой набор из одного или более атрибутов называют ключом-кандидатом.

Может существовать более одного ключа-кандидата, но каждый ключ-кандидат должен однозначно определять каждый кортеж, не только для любого специфического множества кортежей, а для всех возможных кортежей в любой момент времени.

Обратный принцип тоже верен: если взять два кортежа с одинаковыми значениями ключа-кандидата, то оба этих кортежа должны представлять одну и ту же сущность.

Однозначная идентификация любого кортежа является непременным требованием к ключу-кандидату. Необходимо досконально понимать специфику предметной области, чтобы правильно определить ключевой набор атрибутов.

Если буквально следовать определению, то каждое отношение должно иметь, по меньшей мере, один ключ-кандидат — набор всех атрибутов, составляющих кортеж. Ключ-кандидат может состоять из единственного атрибута (простой ключ) или множества атрибутов (составной ключ). Дополнительное требование к ключукандидату такое: он должен состоять из минимального набора атрибутов, однозначно идентифицирующих кортеж, поэтому полный набор атрибутов отношения не обязательно является ключом кандидатом.

Иногда, хотя и не часто, отношение имеет несколько возможных ключей-кандидатов. Тогда проектировщик должен по своему усмотрению выбрать один ключ-кандидат в качестве первичного ключа, а оставшиеся ключи-кандидаты будут являться альтернативными ключами.

Это особенность логической модели довольно удобна. (Логическая модель данных — абстрактное понятие). Чтобы различать модель и ее физическую реализацию, я предпочитаю использовать термин ≪ключ-кандидат≫ для логической модели данных и приберечь ≪первичный ключ≫ для использования в физической реализации модели.

Когда единственно возможный ключ-кандидат чересчур громоздок (например,

состоит из слишком многих атрибутов или слишком велик), можно использовать специальный тип данных, который механизм баз данных SQL Server поддерживает для создания искусственных ключей. В таких искусственных ключах хранятся значения, генерируемые самой системой. Называемые Auto-Increment в SQL Server, поля этого типа очень удобны для создания идентификаторов строк. При этом подразумевается, что вы не будете пытаться связать поле этого типа с какой-нибудь конкретной сущностью предметной области. Такие поля не более чем ярлыки. Ничто не гарантирует, что значения содержащихся в них величин будут строго последовательными, вы практически не можете контролировать процесс их генерации системой. Так что не пытайтесь использовать их для чего-либо еще, кроме нумерации.

Функциональная зависимость

Концепция функциональной зависимости чрезвычайно удобна для анализа структуры данных. Рассматривая любой кортеж T и два набора атрибутов этого кортежа {X1,...,Xn} и {Yl,...,Yn} (множества X и Y не обязательно являются взаимоисключающими) говорят, что Y функционально зависит от X, если для каждого возможного значения X существует единственно возможное соответствующее значение Y.

Указать на функциональную зависимость между атрибутами можно так, как это сделано на рис. В тексте следует записать функциональную зависимость в виде выражения X -> Y, что читается как ≪X функционально зависит от Y≫.

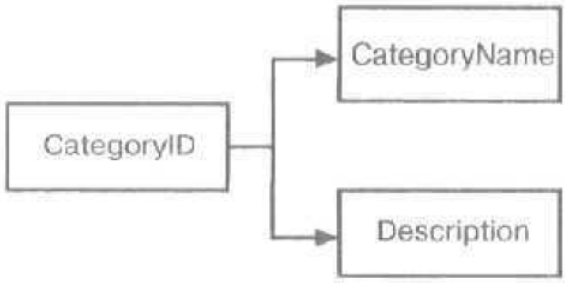


Рис. Диаграмма функциональной зависимости информативна и интуитивно понятна.

На практике функциональная зависимость — удобный способ отразить тот очевидный факт, что для любого отношения существует некий набор атрибутов, уникальный для каждого кортежа данного отношения, и зная этот набор, можно определить значения других, не уникальных атрибутов.

Нормализация —- это процесс, конечным результатом которого является диаграмма функциональной зависимости, подобная изображенной на рис., на которой все стрелки выходят из ключей-кандидатов.

Первая нормальная форма

Отношение находится в первой нормальной форме, если домены, в которых определены его атрибуты, являются скалярными величинами. Понятие скалярной величины — одновременно самое простое и самое сложное в моделировании данных. Принцип таков: каждый атрибут кортежа должен содержать отдельную величину.

Не так просто определить, является ли атрибут скалярным — мы сталкивались с этим, когда рассматривали моделирование имен и адресов.

Дата состоит из трех различных компонентов,: дня, месяца и года. Как лучше хранить дату: как три различных атрибута или как единое целое? Как всегда, ответ зависит от особенностей предметной области, которую вы моделируете.

Часто проблемы, связанные с составными данными, возникают при моделировании кодов и флагов.

Другой тип составных данных, также вызывающий сложности при моделировании — битовый флаг. В принятой повсеместно практике программирования множество булевых

величин сохраняют в виде значений индивидуальных битов в слове, а затем используют побитовые операции, чтобы проверять и извлекать эти значения.

К сожалению, подобные ограничения часто налагаются предысторией разработки: использовавшимися в прошлом средами разработки, корпоративной политикой и т. д.. Но если у вас есть какой-то выбор, сохраняйте в каждом отдельном атрибуте только одну логически неделимую часть информации. Если вы используете унаследованную информацию, то всегда можете разделить данные на логически независимые составляющие и хранить обе версии данных — старую и модифицированную.

Существуют другие виды составных данных это повторяющиеся группы данных.

Вторая нормальная форма

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме, и кроме того, все его атрибуты зависят от полного набора атрибутов ключа-кандидата.

Решение: декомпозиция исходного отношения на два независимых отношения.

Третья нормальная форма

Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме, и кроме того, все не ключевые атрибуты совершенно независимы.

Решение, когда и как вводить в модель третью нормальную форму, может приниматься только с учетом семантики строящейся модели (впрочем, как и любое другое решение в ходе построения модели данных). Как правило, отдельные отношения проектируются только для моделирования наиболее важных сущностей предметной области, или когда предполагается частое изменение каких-то данных, или когда вы получаете в результате создания конкретного отношения серьезные технологические выгоды. Почтовые индексы изменяются, но не часто; кроме того, они не являются существенно важными элементами большинства систем.

Дальнейшая нормализация

Первые три нормальные формы были введены доктором Коддом в его оригинальной работе по реляционной теории, и в подавляющем большинстве случаев это все, о чем вам следует беспокоиться. Вспоминаю пословицу, которую я слышал в университете: ≪Ключ, целый ключ и ничего кроме ключа, и да поможет мне Кодд!≫ Другие нормальные формы — Бойса-Кодда, четвертая и пятая, были разработаны для специальных случаев, которые редко встречаются на практике.

Нормальная форма Бойса-Кодда

Нормальная форма Бойса-Кодда рассматривается как вариант третьей нормальной формы. Она имеет дело со специальной разновидностью отношений, для которых существует несколько ключей-кандидатов. Фактически, чтобы применять нормализацию по Бойсу-Кодду, нужно сочетание нескольких условий:

- отношение должно иметь не менее двух ключей-кандидатов;
- по крайней мере два ключа-кандидата должны быть составными;
- ключи-кандидаты должны иметь перекрывающиеся атрибуты.

Простейший способ понять, что такое нормальная форма Бойса-Кодда — использовать функциональные зависимости. Нормальная форма Бойса-Кодда, в сущности, равносильна утверждению, что между ключами-кандидатам и отношения нет функциональных зависимостей

Нарушения нормальной формы Бойса-Кодда легко избежать, если уделить внимание

логическому смыслу проектируемого отношения. Не следует собирать в одно отношение информацию о разных сущностях. Если отношение содержит информацию о производимой продукции, то туда не следует включать информацию о поставщике (разумеется, верно и обратное утверждение).

Четвертая нормальная форма

Четвертая нормальная форма подводит теоретическую базу под интуитивно очевидный принцип: независимые повторяющиеся группы данных не следует размещать в одном и том же отношении.

Упрощенно, нормализация до четвертой нормальной формы состоит в выделении многозначных зависимостей в разные отношения.

Формально отношение находится в четвертой нормальной форме, если оно находится в нормальной форме Бойса-Кодда, и кроме того, все многозначные зависимости являются также функциональными зависимостями от ключей-кандидатов.

Приведение отношения к четвертой нормальной форме актуально, только если между атрибутами существуют многозначные связи.

Пятая нормальная форма

Пятая нормальная форма имеет дело с чрезвычайно редко встречающимся случаем зависимостей соединения. Зависимости соединения подчиняются следующему принципу: ≪Если сущность 1 зависит от сущности 2, сущность 2 зависит от сущности 3, а сущность 3 в свою очередь зависит от сущности 1, то все три сущности обязательно должны входить в один и тот же кортеж≫.

≪Если поставщик поставляет товар, этот товар интересует заказчика, и заказчик работает с поставщиком, то заказчик непременно получает товар у поставщика≫.

Связи

Ранее подробно рассмотрен процесс нормализации модели данных. Суть этого процесса — в анализе сущностей предметной области для моделирования отношений, охватывающих все относящиеся к ним данные. Но отношения составляют лишь определенную часть модели данных. Ее другая неотъемлемая часть — связи между отношениями и ограничения, налагаемые на эти связи. Материал достаточно прост и понятен, если вы знакомы с семантикой модели данных. Есть и некоторые особые случаи, которые не укладываются в рамки модели связей.

Основные понятия и определения

Сущности, между которыми существуют связи, называются «участниками», а число участников связи — «размерностью» связи. Большинство связей между сущностями — это двойные связи, то есть такие, в которых участвуют две сущности. Встречаются также унарные связи (в которых сущность связана сама с собой) и тройные связи (в которых участвуют три сущности).

Участие каждой сущности в связи бывает «полным» или «частичным», в зависимости оттого, может ли эта сущность существовать, если данная связь не определена.

Тот же самый принцип часто используется при классификации сущностей как «слабых» (то есть таких, участие которых в связи является полным) или «обычных» (участие их в связи является частичным). «Слабые» сущности могут существовать только при наличии связей с другими сущностями, в то время как «обычные» сущности существуют независимо от наличия связей между ними и другими сущностями. Подобная классификация положена в основу метода составления диаграмм ≪сущности — связи≫, предложенного Ченом (Chen).

Связи можно классифицировать одним из трех возможных способов: как «полные» или «частичные», «необязательные» или «обязательные», а также в терминах «слабых» и «обычных» сущностей.

Подобная классификация всех связей не является строго обязательной при создании модели данных. При моделировании больших сложных систем указано влияние отношения на сущность, задача разработчика заметно облегчается. Чтобы указать на диаграмме ≪сущности — связи≫, является ли сущность слабой или обычной, вы можете использовать обозначения.

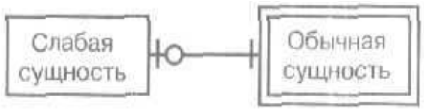


Рис. Различия между «слабыми» и «обычными» сущностями.

В некоторых случаях полезно разделить все связи на два типа — «является А» или «входит в А» и «обладает А». Принцип такого разделения весьма прост, мы проиллюстрируем его на примере сущностей В и С: В либо входит в С, либо обладает С. Например, для сущностей «Сотрудник» и «Баскетбольная команда» может быть определена связь «Сотрудник входит в баскетбольную команду»; для сущностей «Сотрудник» и «Адрес» может быть определена связь «Сотрудник обладает Адресом».

Классификация участия в связи подразумевает указание обязательности данной связи: является ли для сущности участие в данной связи обязательным.

Максимальное число экземпляров одной сущности, которое может быть связано с экземпляром другой сущности, мы будем называть мощностью данной связи. Существуют три основных разновидности мощности связей: ≪один к одному≫, ≪один ко многим≫ и ≪многие ко многим≫.

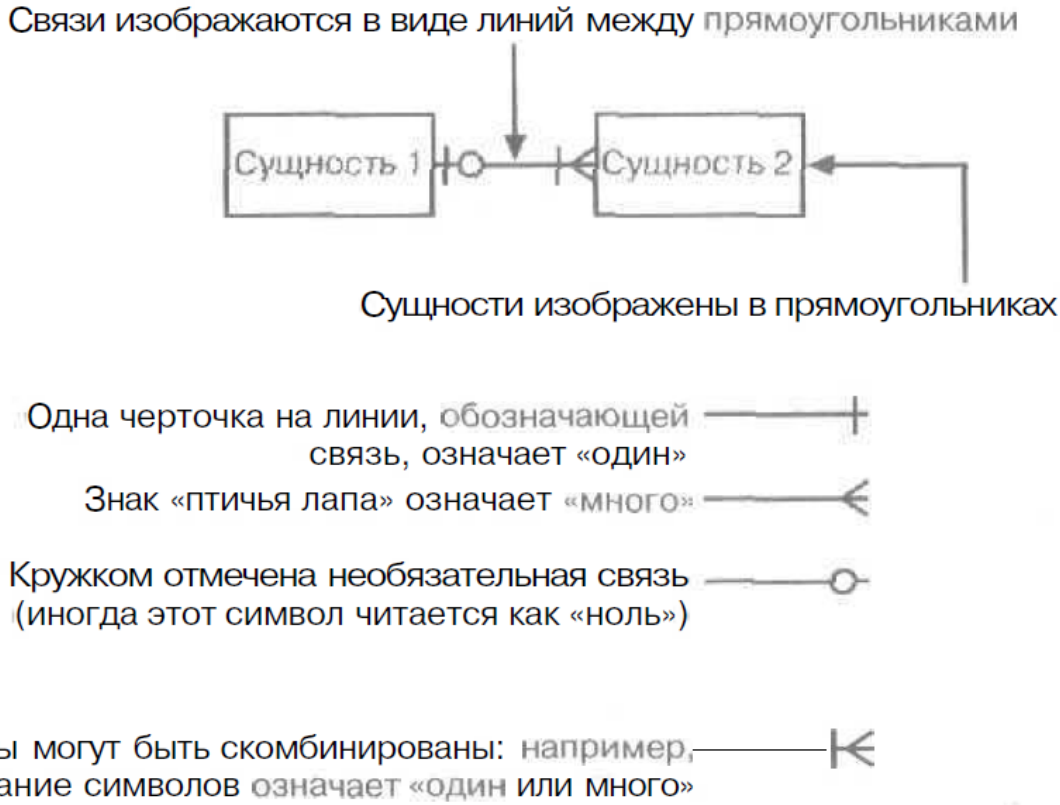


Рис. Обозначение обязательности и мощности связей на диаграммах.