1. Базы данных и управление ими, архитектура системы баз данных

Любая организация нуждается в своевременном доступе к информации. Ценность информации в современном мире очень высока. Роль распорядителей информации в современном мире чаще всего выполняют базы данных. Базы данных обеспечивают надежное хранение информации, структурированном виде и своевременный доступ к ней. Практически любая современная организация нуждается в базе данных, удовлетворяющей те или иные потребности по хранению, управлению и администрированию данных.

База данных может быть определена как совокупность предназначенных для машинной обработки и хранения данных, которые могут использоваться одним или несколькими пользователями. Её можно также рассматривать как подобие электронной картотеки, т.е. контейнер для некоторого набора файлов данных (или таблиц), занесенных в компьютер.

Пользователям этой системы предоставляется возможность выполнять (или передавать системе запросы на выполнение) множество различных операций над такими файлами, например:

- добавлять новые пустые файлы в базу данных;
- вставлять новые данные в существующие файлы;
- получать данные из существующих файлов;
- удалять данные из существующих файлов;
- изменять данные в существующих файлах;
- удалять существующие файлы из базы данных.

Базы данных позволяют хранить, структурировать информацию и извлекать оптимальным для пользователя образом. Использование клиент-серверных технологий позволяют сберечь значительные средства, а главное и время для получения необходимой информации, а также упрощают доступ и ведение, поскольку они основываются на комплексной обработке данных и централизации их хранения. Кроме того, ЭВМ позволяет хранить любые форматы данных, текст, чертежи, данные в рукописной форме, фотографии, записи голоса и т.д.

Информация в БД должна быть:

- непротиворечивой
- неизбыточной
- целостной

Для использования столь огромных объемов хранимой информации, помимо развития системных устройств, средств передачи данных, памяти, необходимы средства обеспечения диалога человек – ЭВМ, которые позволяют пользователю вводить запросы, читать файлы, модифицировать хранимые данные, добавлять новые данные или принимать решения на

основании хранимых данных. Для обеспечения этих функций созданы специализированные средства – системы управления базами данных (СУБД). Современные СУБД – многопользовательские системы управления базой данных, которые специализируется на управлении массивом информации одним или множеством одновременно работающих пользователей.

Современные СУБД обеспечивают:

– набор средств для поддержки таблиц и отношений между связанными таблицами;

– развитый пользовательский интерфейс, который позволяет вводить и модифицировать информацию, выполнять поиск и представлять информацию в графическом или текстовом режиме;

– средства программирования высокого уровня, с помощью которых можно создавать собственные приложения.

Все запросы пользователей на получение доступа к базе данных обрабатываются СУБД. Все имеющиеся средства добавления файлов (или таблиц), выборки и обновления данных в этих файлах или таблицах также предоставляет СУБД. Основная задача СУБД – дать пользователю базы данных возможность работать с ней, *не вникая во все подробности работы на уровне аппаратного обеспечения.* (Пользователь СУБД более отстранен от этих подробностей, чем прикладной программист, применяющий языковую среду программирования.) Иными словами, СУБД позволяет конечному пользователю рассматривать базу данных как объект более высокого уровня по сравнению с аппаратным обеспечением, а также предоставляет в его распоряжение набор операций, выражаемых в терминах языка высокого уровня.

В состав языковых средств современных СУБД входят:

- язык описания данных, предназначенный для описания логической структуры данных;
- язык манипулирования данными, обеспечивающий выполнение основных операций над данными – ввод, модификацию и выборку;
- язык структурированных запросов (SQL – Structured Query Language), обеспечивающий управление структурой БД и манипулирование данными, а также являющийся стандартным средством доступа к удалённым БД;
- язык запросов по образцу (QBE – Query By Example), обеспечивающий визуальное конструирование запросов к БД

Прикладные программы, или приложения, служат для обработки данных, содержащихся в БД. Пользователь осуществляет управление БД и работу с её данными именно с помощью приложений, которые называются приложениями БД. Иногда термин «база данных» трактуют в более широком

смысле и обозначают им не только саму БД, но и приложения, обрабатывающие её данные.

В зависимости от взаимного расположения приложения и БД можно выделить:

- локальные БД;
- удалённые БД

Для выполнения операций с локальными БД разрабатываются и используются так называемые *локальные приложения*, а для операций с удалёнными БД – *клиент-серверные приложения*.

При использовании *локальной* БД в сети можно организовать *многопользовательский доступ*. В этом случае файлы БД и предназначенное для работы с ней приложение располагаются на сервере сети. Каждый пользователь запускает со своего компьютера это расположенное на сервере приложение, при этом у него запускается копия приложения. Такой сетевой вариант использования локальной БД соответствует архитектуре *файл-сервер*. Приложение при использовании архитектуры файл-сервер также может быть записано на каждый компьютер сети, в этом случае приложению отдельного компьютера должно быть известно местонахождение общей БД.

При работе с данными на каждом пользовательском компьютере сети используется локальная копия БД. Эта копия периодически обновляется данными, содержащимися в БД на сервере.

Архитектура файл-сервер обычно применяется в сетях с небольшим количеством пользователей, для её реализации подходят локальные СУБД, например, MS Access, Paradox или dBase. Достоинством этой архитектуры является простота реализации, а также то, что приложение фактически разрабатывается в расчёте на одного пользователя.

Недостатками такого варианта архитектуры можно считать следующие:

В связи с тем, что на каждом компьютере имеется своя копия БД, изменения, сделанные в ней одним пользователем, в течение некоторого времени не известны другим пользователям. Поэтому требуется постоянное обновление БД. Кроме того, возникает необходимость синхронизации работы отдельных пользователей, связанных с блокировкой в таблицах записей, которые в данный момент редактирует другой пользователь.

Удалённая БД размещается на компьютере-сервере сети, а приложение, работающее с этой БД, находится на компьютере пользователя. В этом случае мы имеем дело с архитектурой *клиент-сервер*, когда информационная система делится на неоднородные части – сервер и клиент БД. В связи с тем, что компьютер-сервер отделён от клиента, его также называют *удалённым сервером*.

Клиент – это приложение пользователя. Для получения данных клиент формирует и отсылает запрос удалённому серверу, на котором размещена БД.

Запрос формулируется на языке SQL, который является стандартным средством доступа к серверу при использовании реляционных моделей данных. После получения запроса удалённый сервер направляет его программе SQL Server (серверу БД) – специальной программе, управляющей удалённой БД и обеспечивающей выполнение запроса и выдачу результата клиенту. Вся обработка запроса выполняется на удалённом сервере.

Описанная архитектура является *двухуровневой* (уровень приложения-клиента и уровень сервера БД). Клиентское приложение называют *сильным*, или «толстым» клиентом. Дальнейшее развитие данной архитектуры привело к появлению *трехуровневого* варианта архитектуры клиент-сервер: приложение-клиент, сервер приложений и сервер БД.

В *трехуровневой* архитектуре часть средств и кода, предназначенных для организации доступа к данным и их обработке, из приложения-клиента выделяется в сервер приложений. Само клиентское приложение при этом называют *слабым*, или «тонким» клиентом. В сервере приложений удобно располагать средства и код, общие для всех клиентских приложений, например, средства доступа к БД.

2. Реляционная модель данных

Любая модель данных должна содержать три компонента:

1) ***структуру данных*** – описывает точку зрения пользователя на представление данных;

2) ***набор допустимых операций***, выполняемых на структуре данных. Модель данных предполагает, как минимум, наличие языка определения данных (DDL – Data Definition Language), описывающего структуру их хранения, и языка манипулирования данными (DML – Data Manipulation Language), включающего операции извлечения и модификации данных;

3) ***ограничения целостности*** – механизм поддержания соответствия данных предметной области на основе формально описанных правил.

В процессе исторического развития в СУБД использовались следующие модели данных: *иерархическая, сетевая, реляционная.* В последнее время все большее значение приобретает объектно ориентированный подход к представлению данных.

Здесь мы рассмотрим *реляционную модель*.

В основе реляционной модели данных лежат три основные концепции: *реляционные отношения, ограничения целостности данных и алгебра реляционных операторов*

Переход к реляционным базам данных обычно датируется публикацией статьи сотрудника IBM Эдгара Кодда «Реляционная модель данных для больших совместно используемых банков данных», опубликованной в 1970 году. В этой и последующих статьях Э. Кодда и других авторов (среди которых следует отметить автора популярнейшего учебника по базам данных

К. Дейта) были развиты основные положения реляционной модели данных. Интересно отметить, что само понятие «модель данных», означающее переход на более абстрактный уровень оперирования данными, было сформулировано именно Коддом и в базах данных первого поколения не использовалось. Кодд предложил использовать для обработки данных аппарат теории множеств.

Особенностью реляционной модели является совершенно новый взгляд на построение базы данных и действия с данными. Основной структурной единицей реляционной модели является *отношение* (relation), которое с теоретической точки зрения является легко описываемым и хорошо изученным математическим объектом, а с практической – может, с некоторыми оговорками, трактоваться как таблица простейшей структуры, не имеющая повторяющихся строк. Это резко отличает ее от иерархической и сетевой моделей данных, в которых узлы графа, описывающего структуру базы данных, могли содержать сколь угодно сложные информационные объекты. Упрощение структуры данных, тем не менее, не привело к потере функциональных возможностей обработки данных. Кодду удалось показать, что отношение (таблица) является в достаточной степени насыщенным информационным объектом, чтобы обеспечить произвольную обработку данных, если они изначально записаны в табличной форме. Простота структурных единиц реляционной модели позволила привлечь формальные математические методы для описания обработки данных. С этой целью Э. Коддом были разработаны языки *реляционной алгебры* и *реляционного исчисления*, обладающие необходимой полнотой, требуемой для такой обработки. Названные языки высокоуровневые, операндами и результатами в которых являются отношения, а детализация алгоритмов их реализации возложена на систему управления базой данных.

Положив теорию отношений в основу реляционной модели, Э. Кодд обосновал реляционную замкнутость отношений и ряда некоторых специальных операций, которые применяются сразу ко всему множеству строк отношения, а не к отдельной строке. Указанная реляционная замкнутость означает, что результатом выполнения операций над отношениями является также отношение, над которым в свою очередь можно осуществить некоторую операцию. Из этого следует, что в данной модели можно оперировать реляционными выражениями, а не только отдельными операндами в виде простых имен таблиц.

Одним из основных преимуществ реляционной модели является ее однородность. Все данные рассматриваются как хранимые в таблицах и только в таблицах. Каждая строка такой таблицы имеет один и тот же формат.

К числу достоинств реляционного подхода можно отнести:

– наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;

– наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации БД;

– возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

2.1. Реляционное отношение

Рассмотрим следующий пример. Имеется список организаций-поставщиков, поставлявших некоторые товары в определенном количестве по определенной цене за 1 ед. Каждый поставщик описывается следующими характеристиками (свойствами, атрибутами): *код_п* (код поставщика), *имя_п* (имя поставщика), *гор* (город), в котором находится офис данного поставщика. Для описания товара используются такие характеристики, как *код_т* (код товара), *назв_т* (название товара).

Всю информацию о поставках и свойствах объектов данной предметной области можно представить в виде таблицы, например, такого вида:

Таблица «Поставщик-товар»

код_п	*имя_п*	*гор*	*код_т*	*назв_т*	*цена_1*	*кол_во*
п01	Скан	Ярославль	т14	монитор	7 500	10
п01	Скан	Ярославль	т09	принтер	3 400	4
п02	Альфа	Москва	т14	монитор	7 200	6
п03	Тензор	Ярославль	т03	HD диск	4 300	12
п11	ИП Иванов С. Н.	Тутаев	т23	стол	2 000	3

Имена столбцов таблицы – сокращенные названия характеристик (*атрибутов*) объектов рассматриваемой предметной области (поставщиков и товаров), а также свойств поставок (цена, количество поставляемых изделий). Каждая строка таблицы описывает одну конкретную поставку. Эту таблицу можно трактовать нестрого как задание *реляционного отношения*.

Реляционная база данных – это конечный (ограниченный) набор *отношений*. Отношения используются для представления сущностей, а также для представления связей между сущностями. *Отношение* – это двумерная *таблица*, имеющая уникальное имя и состоящая из *строк* и *столбцов*, где *строки соответствуют записям, а столбцы – атрибутам*. Каждая строка в таблице представляет некоторый объект реального мира или соотношения между объектами.

Атрибут – это поименованный столбец отношения. Свойства сущности, его характеристики определяются значениями атрибутов. Порядок следования атрибутов не влияет на само отношение.

Для строгого определения потребуются следующие понятия.

Типы данных, используемые в реляционной модели

Реляционная модель требует, чтобы типы используемых данных были *простыми*. *Простые, или атомарные, типы данных* не обладают внутренней

структурой. Требование, 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 возможна операция конкатенации и т. д.

Прежде чем говорить о целостности сущностей, опишем использование null-значений в реляционных базах данных.

Null-значения данных

Основное назначение баз данных состоит в том, чтобы хранить и предоставлять информацию о реальном мире. Для представления этой информации в базе данных используются привычные для программистов типы данных – строковые, численные, логические и т. п. Однако в реальном мире часто встречается ситуация, когда данные неизвестны или неполны. Например, место жительства или дата рождения человека могут быть неизвестны (база данных разыскиваемых преступников). Если вместо неизвестного адреса уместно было бы вводить пустую строку, то что вводить вместо неизвестной даты?

Для того чтобы обойти проблему неполных или неизвестных данных, в базах данных могут использоваться типы данных, пополненные так называемым ***null-значением***. null-значение – это, собственно, не значение, а некий маркер, показывающий, что значение неизвестно.

Таким образом, в ситуации, когда возможно появление неизвестных или неполных данных, разработчик имеет на выбор два варианта.

Первый вариант состоит в том, чтобы ограничиться использованием обычных типов данных и не использовать null-значения, а вместо неизвестных данных вводить либо нулевые значения, либо значения специального вида – например, договориться, что строка «АДРЕС НЕИЗВЕСТЕН» и есть те данные, которые нужно вводить вместо неизвестного адреса. В любом случае на пользователя (или на разработчика) ложится ответственность за правильную трактовку таких данных. В частности, может потребоваться написание специального программного кода, который в нужных случаях «вылавливал» бы такие данные. Проблемы, возникающие при этом, очевидны – не все данные становятся равноправны, требуется дополнительный программный код, отслеживающий эту неравноправность, в результате чего усложняется разработка и сопровождение приложений.

Второй вариант состоит в использовании null-значений вместо неизвестных данных. При таком подходе есть менее очевидные и более глубокие проблемы. Наиболее бросающейся в глаза проблемой является необходимость использования трехзначной логики при оперировании с данными, которые могут содержать null-значения. В этом случае при неаккуратном формулировании запросов даже самые естественные запросы могут давать неправильные ответы.

Практически все реализации современных реляционных СУБД позволяют использовать null-значения, несмотря на их недостаточную теоретическую обоснованность.

Домены

В реляционной модели данных с понятием тип данных тесно связано понятие домена, которое можно считать уточнением типа данных.

Определение 2.1. Домен – это подмножество значений некоторого типа данных имеющих определенный смысл.

Домен характеризуется следующими свойствами:

- домен имеет *уникальное имя* (в пределах базы данных);
- домен определен на некотором *простом* типе данных или на другом домене;
- домен может иметь некоторое *логическое условие*, позволяющее описать подмножество данных, допустимых для данного домена;
- домен несет определенную *смысловую нагрузку*.

Например, домен *D*, имеющий смысл «возраст сотрудника» можно описать как следующее подмножество множества *N* натуральных чисел:

$$D = \{n \in N : n \geq 18 \;\; and \;\; n \leq 60\}.$$

Если тип данных можно считать множеством всех возможных значений данного типа, то домен напоминает подмножество в этом множестве.

Отличие домена от понятия подмножества состоит именно в том, что *домен отражает семантику*, определенную предметной областью. Может быть несколько доменов, совпадающих как подмножества, но несущих различный смысл. Например, домены «Вес детали» и «Имеющееся количество» можно одинаково описать как множество неотрицательных целых чисел, но смысл этих доменов будет различным, и это будут *различные* домены.

Основное значение доменов состоит в том, что *домены ограничивают сравнения*. Некорректно с логической точки зрения сравнивать значения из различных доменов, даже если они имеют одинаковый тип. В этом проявляется смысловое ограничение доменов. Синтаксически правильный запрос «выдать список всех деталей, у которых вес детали больше имеющегося количества», не соответствует смыслу понятий «количество» и «вес».

Замечание. Не все домены обладают логическим условием, ограничивающим возможные значения домена. В таком случае множество возможных значений домена совпадает с множеством возможных значений типа данных.

Замечание. Не всегда очевидно, как задать логическое условие, ограничивающее возможные значения домена. Например, по-видимому, невозможно задать условие на строковый тип данных, задающий домен «Фамилия сотрудника». Ясно, что строки, являющиеся фамилиями, не

должны начинаться с цифр, служебных символов, мягкого знака и т. д. Но вот является ли допустимой фамилия, состоящая из бессмысленного набора букв типа «Нпрнекккыыыов»? Очевидно, нет! Трудности такого рода возникают потому, что смысл реальных явлений далеко не всегда можно формально описать. Просто человек интуитивно понимает, что такое фамилия, но никто не может дать такое формальное определение, которое отличало бы фамилии от строк, фамилиями не являющимися. Выход из этой ситуации простой – положиться на разум сотрудника, вводящего фамилии в компьютер.

Определение 2.2. Атрибут отношения есть пара вида (*имя_атрибута*: *имя_домена*).

Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.

Реляционное отношение (далее просто отношение), определенное на множестве доменов с именами D_1, D_2,…, D_n – это именованный объект, содержащий две части: ***заголовок*** и ***тело.***

Заголовок $\{A_1{:}D_1, \ldots A_n{:}D_n\}$ – это множество имен атрибутов или, точнее, пар вида (*имя_атрибута* : *имя_домена*). Тело – это множество ***кортежей*** $\{t_1, t_2, \ldots, t_m\}$, где *i*-й кортеж имеет вид $t_i = \{A_1{:}v_{i1}, \ldots A_n{:}v_{in}\}$, где v_{ij} – значение *j*-го атрибута A_j в *i*-м кортеже, $v_{ij} \in D_j$.

$$\left\{\begin{array}{l}\{A_1 : v_{11},\ A_2 : v_{12}, \ldots, A_n : v_{1n}\}\\ \{A_1 : v_{21},\ A_2 : v_{22}, \ldots, A_n : v_{2n}\}\\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\ \{A_1 : v_{m1},\ A_2 : v_{m2}, \ldots, A_n : v_{mm}\}\end{array}\right\}$$

В дальнейшем значение атрибута A_i в кортеже t будем обозначать с помощью точечной нотации $t.A_i$.

Множество кортежей называется *телом отношения.* Тело отношения отражает состояние сущности, поэтому во времени оно постоянно меняется. Тело отношения характеризуется *кардинальным числом,* которое равно количеству содержащихся в нем кортежей.

Одной из главных характеристик отношения является его степень.

Степень отношения определяется количеством атрибутов, которое в нем присутствует. Эта характеристика отношения имеет еще названия: ранг и арность. Отношение с одним атрибутом называется унарным, с двумя атрибутами – бинарным, с тремя – тернарным, с n атрибутами n-арным.

Определение степени отношения осуществляется по заголовку отношения.

Свойства отношений:

- в теле отношения нет одинаковых кортежей;

- кортежи не упорядочены сверху вниз;
- атрибуты не упорядочены слева направо.

Подчеркнем, что все значения атрибутов – атомарные, т. е. относятся к простым типам данных в указанном выше смысле. В этом случае говорят, что отношение находится в *первой нормальной форме* (1НФ) или *нормализовано*.

Отношение удобно представлять в виде таблицы, в которой имена столбцов – это атрибуты, а строки представляют кортежи, как показано ниже.

Представление реляционного отношения

A_1	A_2	.....	A_n
v_{11}	v_{12}	.....	v_{1n}
v_{21}	v_{22}	.....	v_{2n}
.....	.....	.....	.....
v_{m1}	v_{m2}	.....	v_{mn}

В дальнейшем термины «отношение» и «таблица» будут использоваться как синонимы, хотя, строго говоря, отношение и таблица – это не совсем одно и то же. В отличие от отношения, в таблице столбцы (атрибуты) и строки (кортежи) уже идут в определенном порядке.

Определение 2.3. Реляционная база данных (РБД) – это БД, воспринимаемая пользователем как набор нормализованных отношений разной арности.

Выражение «воспринимаемая пользователем» является решающим: идея реляционной модели применяется к концептуальному (и внешнему) уровню системы, а не к внутреннему уровню.

В дальнейшем в некоторых примерах, чтобы не конкретизировать тело отношения (заполнение таблицы конкретными строками), будем использовать т. н. *схемы отношений. Схема отношения* R – это запись вида $R\{A_1, A_2, ..., A_n\}$., где R – имя отношения, $A_1, A_2, ..., A_n$ – имена атрибутов. Примером является следующая схема:

ПОСТАВЩИК-ТОВАР {код_пост, имя_пост, город, код_тов, назв_т, цена_1, кол_во}.

В схеме $R\{A_1, A_2, ..., A_n\}$ символ R играет роль *переменной отношения*. Аналогично переменным определенного типа в языках программирования переменная R может принимать некоторые значения, т. е. определенные заполнения тела отношения. Использовать понятие «переменная отношения» удобно при обсуждении разного рода теоретических вопросов, когда конкретное тело отношения не играет роли.

Почему сегодня так широко используется реляционная модель данных? Дело в том, что отношения можно рассматривать как математические объекты, а это дает возможность работы с ними (и, следовательно, с РБД) строго формализованными математическими процедурами.

2.2. Целостность базы данных: потенциальные и внешние ключи

В любой момент РБД содержит некоторую определенную конфигурацию данных, и эта конфигурация должна отражать реальную действительность (целостность данных). Следовательно, определение РБД нуждается в расширении, включающем *правила целостности данных,* назначение которых в том, чтобы информировать СУБД о разного рода ограничениях реального мира.

Есть правила целостности, которые относятся к конкретной БД, например:

– количество поставляемых деталей > 0;

– код поставщика имеет формат Пdd , где *d* – цифра;

– атрибут *город* принимает значения из определенного списка.

Это специфические для конкретных отношений правила целостности, их выполнение – задача разработчика БД. Однако есть два общих особых правила целостности, которые должны выполняться для *любой* РБД. Эти правила связаны с понятием *потенциальных* и *внешних ключей*.

2.2.1. Потенциальные ключи

Пусть $R\{A_1, ..., A_n\}$ – схема некоторого отношения. Пусть $K \subseteq \{A_1, ..., A_n\}$ – некоторое подмножество атрибутов.

Определение 2.4. Подмножество *K* называется потенциальным ключом, если оно удовлетворяет следующим свойствам:

1. Уникальность ключа – в любой момент в R нет двух различных кортежей с одинаковым значением K;

2. Неизбыточность ключа – $\forall K' \subset K$ не выполняется свойство уникальности.

Пример. В отношении *ПОСТАВЩИК_ТОВАР{код_пост, имя_пост, город, код_тов, название, цена_1, кол_во}* потенциальный ключ *K* = {*код_пост*, *код_товара*}.

Замечание. Возможны отношения, в которых единственным потенциальным ключом будет комбинация *всех* атрибутов, например, это справедливо для отношения со схемой *ПОСТАВЩИК-ТОВАР_1(код_пост, код_тов).*

Для чего нужны потенциальные ключи? Они обеспечивают основной механизм адресации на уровне кортежей. Иными словами, единственный гарантируемый системой способ точно указать на какой-либо кортеж отношения – это указать значение некоторого потенциального ключа.

Если в реляционном отношении есть несколько потенциальных ключей, то один из них указывается в качестве *первичного* ключа, остальные считаются *альтернативными* ключами.

Первое правило целостности данных (целостность сущностей) – *в любом отношении реляционной базы данных должен быть указан первичный ключ.*

В принципе при создании таблицы СУБД должна запрашивать у разработчика первичный ключ и при работе контролировать состояние таблицы после каждой ее модификации, не допуская наличия разных кортежей с одинаковым значением первичного ключа.

Далее в схеме отношения атрибуты, входящие в первичный ключ, будут выделяться подчеркиванием, например

СТУДЕНТ-ОЦЕНКА{*номер_студ_билета, фамилия, группа, дисциплина, оценка*}.

Второе правило, которое называется *правилом целостности по ссылкам*, является более сложным. Для этого потребуется ввести понятие внешних ключей.

2.2.2. Внешние ключи

Понятие внешних ключей поясним на примере реляционной базы данных со следующей схемой:

ТОВАР(код_товара, название, фирма_производитель) – справочник деталей;

ПОСТАВЩИК(код_поставщика, имя_поставщика, город) – справочник поставщиков;

ПОСТАВЩИК-ТОВАР(код_пост, код_тов, кол_во) – данные о поставках.

Рассмотрим отношение *ПОСТАВЩИК-ТОВАР*. Конкретное значение атрибута *код_пост* допустимо для отношения *ПОСТАВЩИК-ТОВАР* лишь в том случае, если такое же значение существует в качестве первичного ключа *K={код_поставщика}* в отношении *ПОСТАВЩИК*. Другими словами, не имеет смысла включать в *ПОСТАВЩИК-ТОВАР* поставку для поставщика *П07*, если в справочнике поставщиков *ПОСТАВЩИК отсутствует* поставщик с кодом *П07*. Аналогичная ситуация для деталей.

Определение 2.5. Пусть *R*1 – отношение. Тогда внешний ключ *L*1 в *R*1 – это подмножество атрибутов отношения *R*1, такое что

- существует отношение *R*2 с первичным ключом *L*2;
- каждое значение *L*1 в текущем значении *R*1 совпадает со значением *L*2 некоторого кортежа в текущем значении *R*2 (обратное не требуется!).

Иначе говоря, внешний ключ – это атрибут (или подмножество атрибутов), чьи значения совпадают с имеющимися значениями первичного ключа другого отношения. В приведенном выше примере атрибут *код_пост* в отношении *ПОСТАВЩИК-ТОВАР* является внешним ключом, связывающим это отношение с отношением *ПОСТАВЩИК,* т. к. в *ПОСТАВЩИК* атрибут *код_поставщика,* имеющий тот же смысл, что и *код_пост,* является первичным ключом.

Подобное взаимоотношение между таблицами называется *связью (relationship)*. Связь между двумя таблицами устанавливается путем присвоения значений внешнего ключа одной таблицы значениям первичного ключа другой.

Как правило, внешний ключ отношения *R* является частью потенциального ключа этого же отношения. Однако это не обязательно, например, в БД со схемой

ОТДЕЛ(номер, название, фонд_зарплаты),
СЛУЖАЩИЙ(таб_номер, ФИО, номер_отдела, зарплата, должность)

в отношении *СЛУЖАЩИЙ* атрибут *номер_отдела* является внешним ключом (относительно *ОТДЕЛ*), но он не входит ни в один потенциальный ключ этого отношения.

Второе правило целостности данных (*целостность по ссылкам*) – *БД не содержит несогласованных значений внешних ключей.*

Для отображения связей между отношениями (ссылок по внешним ключам) удобно использовать диаграммы следующего вида:

Рис. 2.1 Связь между таблицами по внешним ключам

При наличии связи $R1 \xrightarrow{L} R2$ по внешнему ключу *L* отношение *R*1 называется *главным* (master-table), а отношение *R*2 – *подчиненным* (detail-table) в данной связи. Так, на рисунке выше отношения *ПОСТАВЩИК* и *ТОВАР* – главные по отношению к подчиненному отношению *ПОСТАВЩИК-ТОВАР*.

2.2.3. Как обеспечивается ссылочная целостность

Второе правило целостности выражено исключительно в терминах *состояний* БД. Любое состояние БД, не удовлетворяющее этому правилу, некорректно. Одна из возможностей избежать некорректности – это запретить любые операции, приводящие к этому состоянию. Однако в некоторых случаях предпочтительнее допустить такую операцию, но при необходимости выполнить некоторые *компенсирующие* операции. Например, если пользователь удаляет из отношения *ПОСТАВЩИК* поставщика с кодом ’П01’ система сама может удалить все поставки этого поставщика из отношения *ПОСТАВЩИК-ТОВАР* (т. н. *каскадное удаление*). Следовательно, у разработчика БД должна быть возможность определить, какие операции должны быть запрещены, а какие разрешены, нужны ли для разрешенных операций компенсирующие, и если да, то какие именно (язык SQL позволяет это делать).

После назначения внешнего ключа СУБД имеет возможность автоматически отслеживать вопросы «ненарушения» связей между отношениями, а именно:

- если пользователь попытается вставить в подчиненную таблицу запись, для внешнего ключа которой не существует соответствия в главной таблице (например, там нет еще записи с таким первичным ключом), СУБД сгенерирует ошибку;
- если пользователь попытается удалить из главной таблицы запись, на первичный ключ которой имеется хотя бы одна ссылка из подчиненной таблицы, СУБД также сгенерирует ошибку;
- если пользователь попытается изменить первичный ключ записи главной таблицы, на которую имеется хотя бы одна ссылка из подчиненной таблицы, СУБД также сгенерирует ошибку.

Замечание. Существуют два подхода к удалению и изменению записей в главной таблице:

- запретить удаление всех записей, а также изменение первичных ключей главной таблицы, на которые имеются ссылки подчиненной таблицы;
- распространить всякие изменения в первичном ключе главной таблицы на подчиненную таблицу, а именно:
  - если в главной таблице удалена запись, то в подчиненной таблице должны быть удалены все записи, ссылающиеся на удаляемую;
  - если в главной таблице изменен первичный ключ записи, то в подчиненной таблице должны быть изменены (автоматически) все внешние ключи записей, ссылающихся на изменяемую.

2.3. Средства манипулирования реляционными данными

Третий компонент реляционной модели (помимо понятия отношений и целостности данных) включает в себя набор операторов, которые дают возможность разработчику реляционной БД генерировать новые отношения из старых (базовых) для решения прикладных задач. Существуют два подхода к построению таких операторов – *реляционная алгебра* и *реляционное исчисление.*

В реализациях конкретных реляционных СУБД сейчас не используется в чистом виде ни реляционная алгебра, ни реляционное исчисление. Фактическим стандартом доступа к реляционным данным стал язык SQL (Structured Query Language). Язык SQL представляет собой смесь операторов реляционной алгебры и выражений реляционного исчисления, использующий синтаксис, близкий к фразам английского языка и расширенный дополнительными возможностями, отсутствующими в реляционной алгебре и реляционном исчислении. Вообще, язык доступа к данным называется ***реляционно полным***, если он по выразительной силе не уступает реляционной алгебре (или, что то же самое, реляционному

исчислению), т. е. любой оператор реляционной алгебры может быть выражен средствами этого языка. Именно таким и является язык SQL.

Первая редакция этого языка была опубликована в 1986 году и уточнена в 1989 году. Она обладала заметной неполнотой и в 1992 году была заменена новой редакцией SQL-92 (или SQL2). Вторая редакция появилась в период бурного роста числа автоматизированных систем, использующих базы данных, и явилась важным ориентиром для разработчиков СУБД. Совместимость с этим стандартом и сейчас выступает важной характеристикой той или иной системы управления базами данных.

Безусловно, расширение возможностей компьютерной техники и рост потребностей конечных пользователей порождают новые средства реализации этих потребностей со стороны систем управления базами данных. Это приводит к тому, что используемые в реальных системах языки работы с данными являются расширениями подмножеств языка SQL (кстати, то же самое происходит и с языками программирования). Критический анализ этих сокращений и расширений приводит к созданию новых редакций стандарта языка SQL. Так, за SQL-92 последовали SQL:1999 (SQL3), SQL:2003 и SQL:2008. Однако стоит заметить, что хотя реальные СУБД в ходе своего развития и приближаются к текущему стандарту, но не соответствуют ему, и для эффективного их использования все равно следует изучать документацию по конкретной СУБД. Тем не менее, знание стандарта позволяет оценить перспективы развития СУБД и избежать решений, которые не будут впоследствии соответствовать стандарту и которые придется серьезно дорабатывать.

2.3.1. Реляционная алгебра Кодда

Реляционная алгебра представляет собой набор операторов, использующих отношения в качестве аргументов и возвращающих отношения в качестве результата. Таким образом, реляционный оператор выглядит как функция с отношениями в качестве аргументов:

$$R = f(R_1, R_2, \ldots, R_n).$$

Реляционная алгебра является замкнутой, т. к. в качестве аргументов в реляционные операторы можно подставлять другие реляционные операторы, подходящие по типу:

$$R = f\left(f_1\left(R_{11}, \ldots, R_{1k_1}\right), f_2\left(R_{21}, \ldots, R_{2k_2}\right), \ldots\right).$$

Таким образом, в реляционных выражениях можно использовать вложенные выражения сколь угодно сложной структуры.

Каждое отношение обязано иметь уникальное имя в пределах базы данных. Имя отношения, полученного в результате выполнения реляционной операции, определяется в левой части равенства. Однако можно не требовать наличия имен от отношений, полученных в результате реляционных выражений, если эти отношения подставляются в качестве аргументов в другие реляционные выражения. Такие отношения будем называть

неименованными отношениями. Неименованные отношения реально не существуют в базе данных, а только вычисляются в момент вычисления значения реляционного оператора.

Существует много подходов к определению наборов операторов и способов их интерпретации, но в принципе все они являются более или менее равносильными. Здесь будет рассмотрен классический подход, предложенный Кристофером Коддом. В этом варианте множество операторов состоит из восьми операций, составляющих две группы по четыре оператора в каждой:

- Традиционные операторы над множествами: *объединение, пересечение, вычитание* и *декартово произведение* (все они модифицированы с учетом того, что их операндами являются отношения, а не произвольные множества).
- Специальные реляционные операторы: *выборка, проекция, соединение и деление*.

Кроме того, в состав алгебры включается операция *присваивания*, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и операция *переименования атрибутов*, дающая возможность корректно сформировать схему результирующего отношения.

Операция присваивания имеет вид R = <выражение реляционной алгебры>, где R – переменная отношения. Операция переименования атрибута – это выражение вида R RENAME A_1, A_2 ,… AS B_1,B_2 ,…, где R – переменная отношения, A_1, A_2 ,… – имена атрибутов отношения R, B_1,B_2,… – новые имена. Это выражение приводит к получению отношения с тем же заголовком и телом, что и отношение, которое является текущим значением переменной R, за исключением того, что в нем атрибуты A_i называются теперь B_i.

Два отношения называются *совместимыми по типу*, если они имеют одинаковые заголовки.

Теоретико-множественные операторы

Объединение. Объединением двух совместимых по типу реляционных отношений A и B (A UNION B) называется отношение C с тем же заголовком, что у A и B, и с телом, состоящим из множества всех кортежей, принадлежащим A или B или обоим вместе:

$$t \in A \ \ UNION \ \ B \Leftrightarrow t \in A \ \ OR \ \ t \in B.$$

Оператор объединения поясняется на рис. 2.2 (а).

Пересечение. Пересечением двух совместимых по типу отношений R и S (синтаксис R INTERSECT S) называется отношение с тем же заголовком, что у R и S, и с телом, состоящим из множества всех кортежей, принадлежащим R и S:

$$t \in A \ \ INTERSECT \ \ B \Leftrightarrow t \in A \ \ AND \ \ t \in B.$$

Оператор пересечения поясняется на рис. 2.2 (б).

Вычитание. Вычитанием двух совместимых по типу отношений A и B (A MINUS B) называется отношение с тем же заголовком, что у A и B, и с телом, состоящим из множества всех кортежей, принадлежащим A и не принадлежащим B:

$$t \in A \;\; MINUS \;\; B \Leftrightarrow t \in A \;\; AND \;\; t \notin B.$$

Оператор вычитания поясняется на рис. 2.2 (в).

Декартово произведение. Пусть имеются два отношения R $\{A_1, A_2, \ldots, A_m\}$ и S $\{B_1, B_2, \ldots, B_n\}$. Тогда результатом операции произведения R TIMES S является отношение T $\{A_1, \ldots, A_m, B_1, \ldots, B_n\}$.

Оператор декартового произведения поясняется на рис. 2.2 (г).

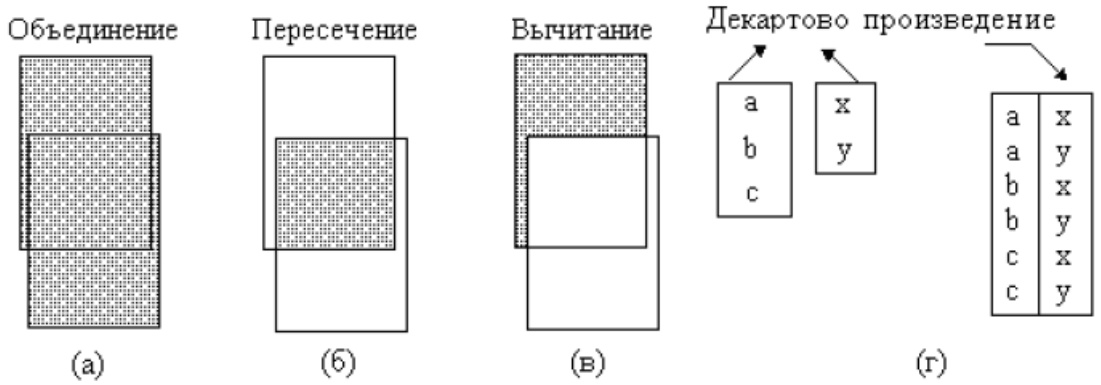


Рис. 2.2. Иллюстрация теоретико-множественных операций

Здесь может возникнуть проблема – как получить корректно сформированный заголовок отношения-результата? Поскольку заголовок результирующего отношения является сцеплением заголовков отношений-операндов, то очевидной проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами.

Эти соображения приводят к введению понятия совместимости по взятию декартова произведения. Два отношения совместимы по взятию декартова произведения в том и только том случае, если пересечение множеств имен атрибутов, взятых из их схем отношений, пусто. Любые два отношения всегда могут стать совместимыми по взятию декартова произведения, если применить операцию переименования одноименных атрибутов.

Следует заметить, что операция взятия декартова произведения не является слишком осмысленной на практике. Во-первых, мощность тела ее результата очень велика даже при допустимых мощностях операндов, а, во-вторых, результат операции не более информативен, чем взятые в совокупности операнды. Как будет показано далее, основной смысл включения операции расширенного декартова произведения в состав реляционной алгебры Кодда состоит в том, что на ее основе определяется действительно полезная операция соединения.

Специальные операторы

Выборка (θ-выборка). Пусть $\theta \in \{=, >, <, \geq, \leq, \neq\}$ – символ операции сравнения, R – отношение. Тогда θ-выборка (R WHERE $X\,\theta\,Y$) – это отношение с тем же заголовком, что и R, содержащее кортежи t из R, для которых условие $X\,\theta\,Y$ истинно. Здесь X – атрибут, а Y – атрибут, определенный на том же домене, что и X, или литерал (т. е. символьная строка или число).

Пример. Рассмотрим отношение *ПОСТАВЩИК-ТОВАР* (табл. 2.1). Тогда выражение

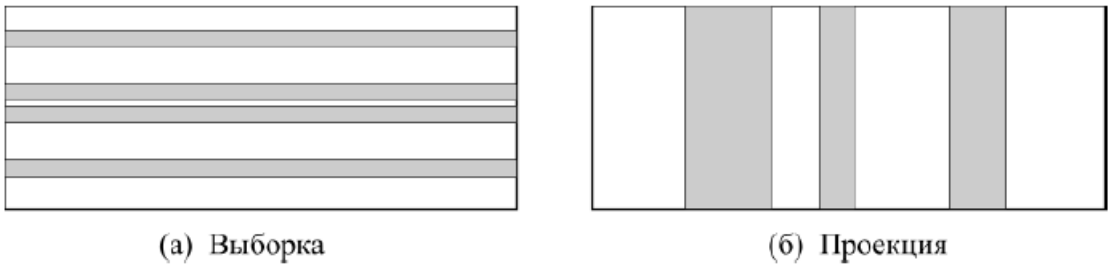
ПОСТАВЩИК-ТОВАР WHERE *назв_т* = ‘Монитор’ определяет отношение с телом вида

код_п	*имя_п*	*гор*	*код_т*	*назв_т*	*цена_1*	*кол_во*
П01	Скан	Ярославль	Т14	монитор	7 500	10
П11	Альфа	Москва	Т14	монитор	7 200	3

Обычно определение выборки расширяется до формы, в которой условие в выражении WHERE будет содержать произвольное число логических сочетаний простых условий, например,

ПОСТАВЩИК-ТОВАР WHERE (*назв_т* = ‘монитор’)
AND (*гор* = ‘Москва’ OR *гор* = ‘Воронеж’).

На интуитивном уровне оператор выборки лучше всего представлять как взятие некоторой «горизонтальной» вырезки из отношения-операнда (выборки некоторых строк из таблицы), как показано на рис. 2.3 (а).



(а) Выборка (б) Проекция

Рис. 2.3. Операции выборки и проекции

Проекция. Проекция отношения R на атрибуты $A,B,\ldots,C$ (синтаксис $R[A,B,\ldots,C]$) – это отношение с заголовком $A,B,\ldots,C$ и телом, состоящим из кортежей $\{A{:}a,\ B{:}b,\ldots,C{:}c\}$, таких, для которых в отношении R значение атрибута A равно a, атрибута B равно b,..., атрибута C равно c. Таким образом, проекция – это подмножество кортежей, получаемое исключением тех атрибутов, которые не указаны в списке атрибутов, и последующим исключением дублирующих подкортежей. Тем самым при выполнении операции проекции выделяется «вертикальная» вырезка отношения-операнда (рис. 2.3 (б)).

Пример. Рассмотрим снова отношение *ПОСТАВЩИК-ТОВАР* (см. табл. 2.1). Выражение

ПОСТАВЩИК-ТОВАР[назв_ т, кол_во]

формирует отношение со следующими кортежами:

Назв_т	*кол_во*
Монитор	10
Принтер	16
Монитор	3
Процессор	3

Соединение. Оператор соединения (называемый также *соединением по условию, или θ-соединением*) требует наличия двух операндов – соединяемых отношений и третьего операнда – простого условия. Пусть соединяются отношения *R* и *S*. Тогда, по определению, результатом операции соединения (*R* JOIN *S*) WHERE usl совместимых по взятию декартова произведения отношений *R* и *S* является отношение, получаемое путем выполнения операции выборки по условию usl декартова произведения отношений *R* и *S*:

(*R* JOIN *S*) WHERE usl ≡ (*R* TIMES *S*) WHERE usl.

Хотя операция соединения в приведенной интерпретации не является примитивной (поскольку определяется с использованием операций декартова произведения и проекции), в силу особой практической важности она включается в базовый набор операций реляционной алгебры Кодда. Заметим также, что в практических реализациях соединение обычно не выполняется именно как выборка декартова произведения. Имеются более эффективные алгоритмы, гарантирующие получение такого же результата.

Важным частным случаем операции соединения по условию является *естественное соединение.* Операция естественного соединения применяется к паре отношений *R*{*A*,*B*} и *S*{*B*,*C*}, обладающих (возможно, составным) общим атрибутом *B* (т. е. атрибутом с одним и тем же именем и определенным на одном и том же домене). Пусть *ABC* обозначает объединение заголовков отношений A и B. Тогда естественное соединение отношений *R* и *S* (*R NATURAL JOIN S*) – это спроецированный на *ABC* результат соединения *R* и *S* по условию *R.B* = *S.B* [1] . Хотя операция естественного соединения выражается через операции переименования, соединения по условию и проекции, для нее обычно используется сокращенная форма, называемая NATURAL JOIN.

[1] Здесь *R.B* и *S.B* представляют собой так называемые **квалифицированные (уточненные)** имена атрибутов (часто такой способ именования называют точечной нотацией). Мы будем использовать подобную нотацию в тех случаях, когда требуется явно показать, схеме какого отношения принадлежит данный атрибут.

Пример. Пусть отношения *ПОСТАВЩИК* и *ПОСТАВЩИК-ТОВАР* имеют следующее заполнение:

ПОСТАВЩИК

код_п	*имя_п*	*гор*
П01	Скан	Ярославль
П02	Альфа	Москва
П03	Тензор	Ярославль

ПОСТАВЩИК-ТОВАР

код_п	*код_т*	*цена_1*	*кол_во*
П01	Т06	23	10
П01	Т04	234	5
П03	Т06	120	6

Выражение *ПОСТАВЩИК* NATURAL JOIN *ПОСТАВЩИК-ТОВАР* создает новое отношение со следующими кортежами:

код_п	*имя_п*	*гор*	*код_т*	*цена1*	*кол_во*
П01	Скан	Ярославль	Т06	123	10
П01	Скан	Ярославль	Т04	234	5
П03	Тензор	Ярославль	Т06	120	6

Замечание. В синтаксисе естественного соединения не указываются, по каким атрибутам производится соединение. Естественное соединение производится *по всем* одинаковым атрибутам.

Замечание. Естественное соединение эквивалентно следующей последовательности реляционных операций:

– переименовать одинаковые атрибуты в отношениях,
– выполнить декартово произведение отношений,
– выполнить выборку по совпадающим значениям атрибутов, имевших одинаковые имена,
– выполнить проекцию, удалив повторяющиеся атрибуты,
– переименовать атрибуты, вернув им первоначальные имена.

Операция деления отношений. Эта операция наименее очевидна из всех операций реляционной алгебры Кодда и поэтому нуждается в более подробном объяснении. Пусть заданы два отношения – $R\{A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m\}$ (делимое) и $S\{B_1, B_2, \ldots, B_m\}$ (делитель). Будем считать, что атрибут B_i отношения R и атрибут B_i отношения S ($i = 1, \ldots, m$) не только обладают одним и тем же именем, но и определены на одном и том же домене.

По определению, результатом деления R на S (R DIVIDE BY S) является отношение $T\{A_1, A_2, \ldots, A_n\}$ (частное), которое состоит из кортежей $t = \{a_1, a_2, \ldots, a_n\}$ таких, что для *всех* кортежей $\{b_1, b_2, \ldots, b_m\} \in S$ в отношении R найдется кортеж $\{a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m\}$. Другими словами, тело декартова произведения T TIMES S содержится в теле отношения R . Поясним это определение на простейшем примере.

Пример. Рассмотрим отношения следующего вида:

R

A	*B*
a1	b1
a1	b2
a2	b1

S

B
b1
b2

Тогда команда *T* = *R* DIVIDE BY *S* сформирует отношение *T* , тело которого содержит один кортеж

A
a1

Ситуацию, в которой удобно использовать операцию реляционного деления, проиллюстрируем на более содержательном примере. Типичные запросы, реализуемые с помощью операции деления, обычно в своей формулировке имеют слово «все» – «Какие поставщики поставляют *все* детали?», «Какие сотрудники работают во *всех* проектах?» и т. п. Пусть в БД некоторой организации поддерживаются два отношения: *ПРОЕКТЫ*{*проект, объем_финанс*}, представляющее справочник всех проектов, выполняемых в организации, и *ПРОЕКТЫ-СОТРУДНИКИ{проект, сотр_имя, отд_номер* }, где для каждого проекта указаны все задействованные в этом проекте сотрудники и отделы, в которых они работают. Требуется составить список всех тех сотрудников, которые работают ***во всех*** проектах. Требуемый список может быть получен в виде отношения *СОТРУД-ВСЕ-ПРОЕКТЫ*{*имя_сотруд, отдел*} следующим образом:

СОТРУД-ВСЕ-ПРОЕКТЫ = ПРОЕКТ-СОТРУДНИК
DIVIDE BY *ПРОЕКТЫ[проект].*

Этот результат поясняется ниже на конкретных значениях переменных отношений *ПРОЕКТЫ* и *ПРОЕКТ-СОТРУДНИК*.

Отметим, что операция реляционного деления не является примитивной – она выражается через операции декартова произведения, взятия разности и проекции.

Основная цель создания реляционной алгебры – возможность формирования запросов к базе данных на формальном языке, операторами которого являются введенные операторы алгебры Кодда.

ПРОЕКТ-СОТРУДНИК-ОТДЕЛ

проект	*сотр_имя*	*отд_номер*
Пр1	Меньшиков О.Е.	1
Пр1	Домогаров А.Ю.	2
Пр1	Безруков С.В.	2
Пр2	Меньшиков О.Е.	1
Пр2	Домогаров А.Ю.	2
Пр2	Безруков С.В.	2
Пр3	Домогаров А.Ю.	2
Пр3	Боярская Е.М.	2
Пр3	Безруков С.В.	2

ПРОЕКТЫ

проект	*объем_финансирования*
Пр1	***100000***
Пр2	***234000***
Пр3	150000

СОТРУД-ВСЕ-ПРОЕКТЫ

сотр_имя	*отд_номер*
Домогаров А.Ю.	2
Безруков С.В.	2

3. Нормализация базы данных

Каждый программист обычно по-своему проектирует базу данных для программы, над которой работает. У одних это получается лучше, у других – хуже. Качество спроектированной БД в немалой степени зависит от опыта и интуиции программиста, однако существуют некоторые правила, помогающие улучшить проектируемую БД. Такие правила носят *рекомендательный* характер, и называются **нормализацией** базы данных.

Процесс нормализации данных заключается в устранении избыточности данных в таблицах.

Существует несколько *нормальных форм*, но для практических целей интерес представляют только первые три *нормальные формы*.

Первая нормальная форма (1НФ) требует, чтобы каждое поле таблицы БД было неделимым (атомарным) и не содержало повторяющихся групп.

Неделимость означает, что в таблице не должно быть полей, которые можно разбить на более мелкие поля. Например, если в одном поле мы объединим фамилию студента и группу, в которой он учится, требование неделимости соблюдаться не будет. Первая нормальная форма требует, чтобы мы разбили эти данные по двум полям.

Под понятием *повторяющиеся группы* подразумевают поля, содержащие одинаковые по смыслу значения. Взгляните на рисунок:

№	Студент 1	Студент 2	Студент 3
1	Иванов И.И.	Петров П.П.	Сидоров С.С.

Повторяющиеся группы

Верно, такую таблицу можно сделать, однако она нарушает правило *первой нормальной формы*. Поля «Студент 1», «Студент 2» и «Студент 3» содержат одинаковые по смыслу объекты, их требуется поместить в одно поле «Студент». Ведь в группе не бывает по три студента, правда? Их гораздо больше. И представляете, как будет выглядеть таблица, содержащая данные на тридцать студентов? Это тридцать одинаковых полей!

В этом случае таблицу следует преобразовать к виду, представленному на рисунке ниже.

№	Студент

Преобразованная структура таблицы

В приведенной выше таблице поля описывают студентов в формате «Фамилия И.О.». Однако если оператор будет вводить эти описания в формате «Фамилия Имя Отчество», то нарушается также правило неделимости. В этом случае каждое такое поле следует разбить на три отдельных поля, так как поиск может вестись не только по фамилии, но и по имени или по отчеству.

№	Фамилия	Имя	Отчество

Вторая нормальная форма (2НФ) требует, чтобы таблица удовлетворяла всем требованиям первой нормальной формы, и чтобы любое неключевое поле однозначно идентифицировалось полным набором ключевых полей. Рассмотрим пример: некоторые студенты посещают спортивные платные секции, и вуз взял на себя оплату этих секций. Взгляните на рисунок:

№ студента	Секция	Плата
100	Плавание	100
110	Скейтборд	150
112	Теннис	175
254	Плавание	100
260	Теннис	175

Нарушение второй нормальной формы

В чем здесь нарушение? Ключом этой таблицы служат поля «№ студента» – «Секция». Однако данная таблица также содержит отношение «Секция» – «Плата». Если мы удалим запись студента № 110, то потеряем данные о стоимости секции по скейтборду. А после этого мы не сможем ввести информацию об этой секции, пока в нее не запишется хотя бы один студент. Говорят, что такое отношение подвержено как аномалии удаления, так и аномалии вставки. В соответствие с требованиями *второй нормальной формы*, каждое неключевое поле должно однозначно зависеть от ключа. Поле «Плата» в приведенном примере содержит сведения о стоимости данной секции, и ни коим образом не зависит от ключа – номера студента. Таким образом, чтобы удовлетворить требованию *второй нормальной формы*,

данную таблицу следует разбить на две таблицы, каждая из которых зависит от своего ключа:

№ студента	Секция
100	Плавание
110	Скейтборд
112	Теннис
254	Плавание
260	Теннис

Ключ: № студента

Секция	Плата
Плавание	100
Скейтборд	150
Теннис	175

Ключ: Секция

Правильная вторая нормальная форма

Мы получили две таблицы, в каждой из которых не ключевые данные однозначно зависят от своего ключа.

Третья нормальная форма (3НФ) требует, чтобы в таблице не имелось транзитивных зависимостей между неключевыми полями, то есть, чтобы значение любого поля, не входящего в первичный ключ, не зависело от другого поля, также не входящего в первичный ключ. Допустим, в нашей студенческой базе данных есть таблица с расходами на спортивные секции:

Секция	Плата	Кол-во студентов	Общая стоимость
Плавание	100	2	200
Скейтборд	150	1	150
Теннис	175	2	350

Нарушение третьей нормальной формы

Как нетрудно заметить, ключевым полем здесь является поле «Секция». Поля «Плата» и «Кол-во студентов» зависят от ключевого поля и не зависят друг от друга. Однако поле «Общая стоимость» зависит от полей «Плата» и «Кол-во студентов», которые не являются ключевыми, следовательно, нарушается правило *третьей нормальной формы*.

Поле «Общая стоимость» в данном примере можно спокойно убрать из таблицы, ведь если потребуется вывести такие данные, нетрудно будет перемножить значения полей «Плата» и «Кол-во студентов», и создать для вывода вычисляемое поле.

Таким образом, нормализация данных подразумевает, что вы вначале проектируете свою базу данных: планируете, какие таблицы у вас будут, какие в них будут поля, какого типа и размера. Затем вы приводите каждую таблицу к первой нормальной форме. После этого приводите полученные таблицы ко второй, затем к третьей нормальной форме, после чего можете утверждать, что ваша база данных нормализована.

Однако такой подход имеет и недостатки: если вам требуется разработать программный комплекс для крупного предприятия, база данных будет довольно большой. При нормализации данных, вы можете получить сотни взаимосвязанных между собой таблиц. С увеличением числа нормализованных таблиц уменьшается восприятие программистом базы

данных в целом, то есть вы можете потерять общее представление вашей базы данных, запутаетесь в связях. Кроме того, поиск в чересчур нормализованных данных может быть замедлен. Отсюда вывод: при работе с данными большого объема ищите компромисс между требованиями нормализации и собственным общим восприятием базы данных.

3. Проектирование баз данных

Реляционные отношения между таблицами

В частном случае БД может состоять из одной таблицы. Но чаще всего реляционная БД состоит из нескольких взаимосвязанных таблиц. Организация связи между таблицами называется *связыванием.* Связи между таблицами можно устанавливать как на этапе разработки БД, так и при разработке приложения. Для связывания таблиц используются соответствующие *поля связи.* Поле связи – особое поле таблицы, которое однозначно идентифицирует запись. В подчиненной таблице для связи с главной также используется особый набор полей, называемый *внешним ключом.* Поле связи и первичный ключ должны быть индексированы, так как это ускоряет доступ к связанным записям. В общем случае поле связи и внешний ключ представляют собой индексы.

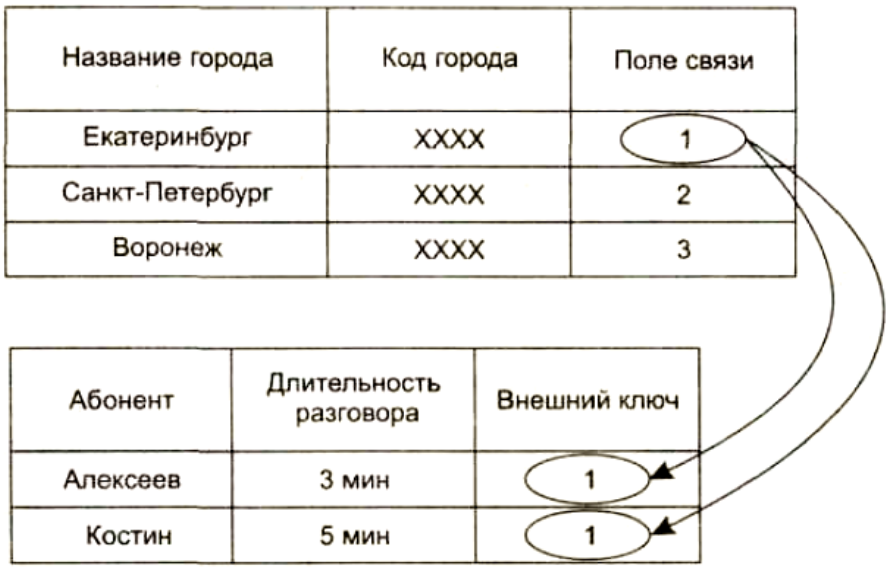


Рис. 3.1. Пример связи между таблицами

В качестве примера на рис. 3.1 приведены данные о длительности разговоров абонентов телефонной сети. «Поле связи» главной таблицы содержит некое значение. С этим значением связано поле подчиненной таблицы «внешний ключ», которое указывает на двух абонентов с фамилиями Алексеев и Костин. Использование поля связи позволяет получить данные о разговорах тех абонентов, которые звонили в Екатеринбург. В качестве поля связи можно было использовать поле «Код города», поскольку оно является уникальным.

Связь между таблицами определяет отношение подчиненности, при котором одна таблица является главной, другая подчиненной. Главная таблица обычно называется Master, подчиненная – Detail.

Различают следующие разновидности связи:

- отношение «один-к-одному»;
- отношение «один-ко-многим»;
- отношение «многие-к-одному»;
- отношение «многие-ко-многим».

Отношение *«один-к-одному»* имеет место, когда одной записи родительской таблицы соответствует одна запись в подчиненной. При этом в подчиненной таблице может содержаться, а может и не содержаться запись, соответствующая записи в главной таблице. Данное отношение обычно используется при разбиении таблицы с большим числом полей на несколько таблиц, чтобы таблица не «распухала» от второстепенной информации. В этом случае в первой таблице остаются поля с наиболее важной и часто используемой информацией, а остальные поля переносятся в другие таблицы. Пример данного отношения изображен на рис. 3.2.

Таблица «Студенты»

№	ФИО	Группа
1	Иванов И. А.	ОПИ-99–2
2	Кузнецов А. Г.	АГП-00–1
3	Перминов И. В.	ИТР-01–2

Таблица «Сдача зачетов»

№	Дата	Оценка
1	24.12.2014	5
2	18.05.2014	5
3	15.06.2013	4

Рис. 3.2. Связь «один-к-одному»

Отношение *«один-ко-многим»* подразумевает, что одной записи главной таблицы может соответствовать несколько записей в одной или нескольких подчиненных таблицах. Этот вид отношения встречается наиболее часто.

Таблица «Студенты»

№	ФИО	Группа
1	Иванов И. А.	ОПИ-99–2
2	Кузнецов А. Г.	АГП-00–1
3	Перминов И. В.	ИТР-01–2

Таблица «Сдача зачетов»

№	Дата	Предмет	Оценка
1	24.12.2004	Химия	5
2	18.05.2004	Физика	5
3	15.06.2003	Математика	4
2	18.05.2003	Математика	4
3	15.06.2003	Математика	4

Рис. 3.3. Связь «один-ко-многим»

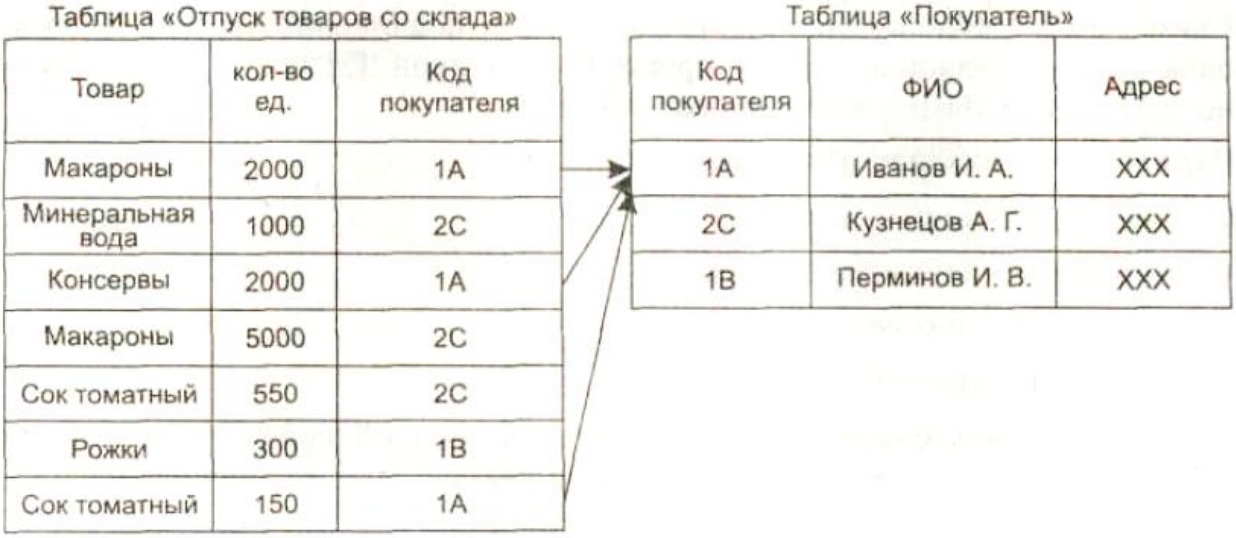


Рис. 3.4. Связь «многие-ко-многим»

Различают две разновидности связи «один-ко-многим». В первом случае для каждой записи главной таблицы должны существовать записи в подчиненной. Во втором – наличие записей в подчиненной таблице необязательно. В примере, приведенном на рис. 3.3, одному студенту в главной таблице соответствует несколько различных предметов. Связь осуществляется по полю «Номер».

Отношение *«многие-ко-многим»* имеет место, когда одной записи главной таблицы соответствует несколько записей подчиненной, а одной записи из подчиненной таблицы может соответствовать несколько записей из главной.

Как видно из рис. 3.4, один вид товара могут купить несколько покупателей, в то же время один покупатель может купить несколько товаров.

Несмотря на то, что многие СУБД не поддерживают данный вид отношения, его можно реализовать неявным способом, если возникнет такая необходимость.

Ссылочная целостность

На рис. 3.5 представлена таблица, находящаяся в отношении «один-ко-многим». Связь производиться по полю «Номер».

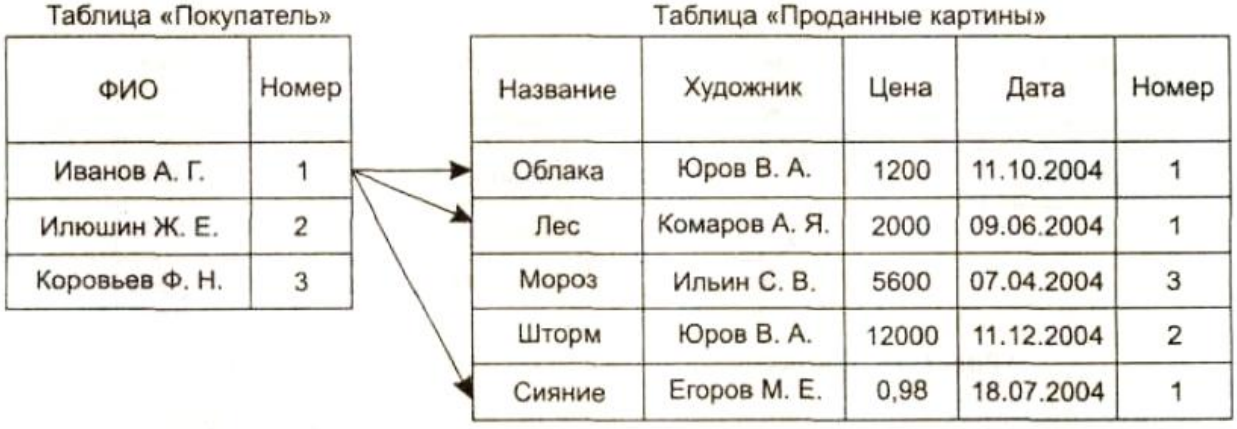


Рис. 3.5. Связанные таблицы

Потеря связей между записями может произойти в нескольких случаях:

- Если будет изменено значение поля связи в родительской таблице без изменения значений в полях дочерней таблицы.
- Если будет изменено значение поля (полей) связи в дочерней таблице без изменения соответствующего значения в родительской таблице.

Рассмотрим простой случай. Неожиданно у одного из клиентов картинной галереи сменился личный номер. Таким образом, в таблице «Покупатель» у Иванова оказался номер 11.

Так как у Иванова А. Г. номер 11, а заказанные им товары находятся под номером 1, то налицо нарушение целостности и достоверности данных. В новом, измененном варианте, Иванов ничего не заказал, а в таблице «Проданные картины» появились «бесхозные данные».

Рассмотрим второй вариант, изображенный на рис. 3.7. Данные в поле «Номер» были изменены на другие. Таким образом, в полях, имеющих прежнее значение «1», данные стали иметь значение «6». В этой ситуации тоже теряются нужные данные.

Рис. 3.6. Нарушение целостности БД

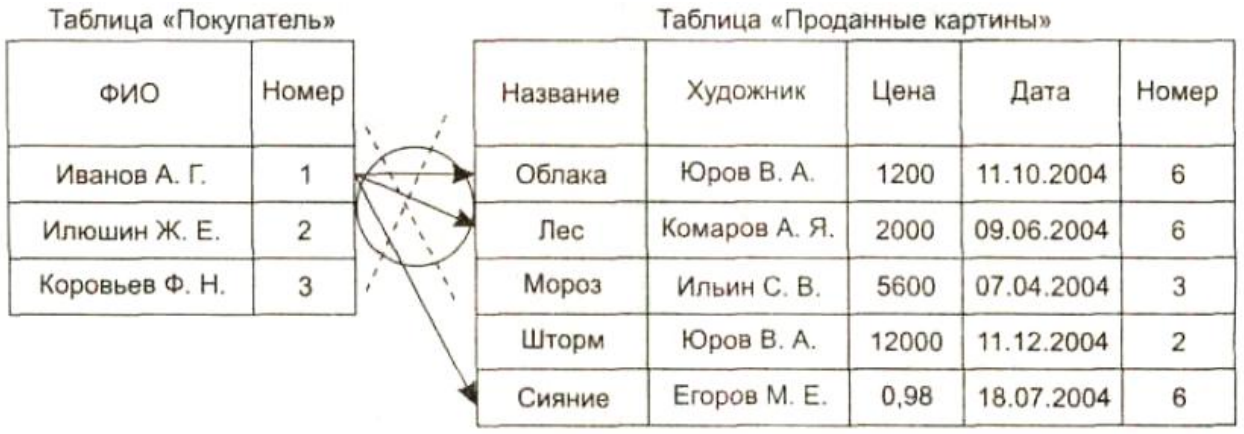


Рис. 3.7. Нарушение целостности БД из дочерней таблицы

В обоих примерах имело место нарушение целостности БД, то есть информация, хранящаяся в БД, стала недостоверной из-за искажения связей.

Нарушение ссылочной целостности может возникнуть в нескольких случаях:

- удаление записи из главной таблицы, без удаления связанных записей в дочерней таблице;
- изменение значения поля связи главной таблицы, без изменения ключа дочерней;
- изменение ключа дочерней таблицы без изменения поля связи главной.

Для предотвращения потери ссылочной целостности, используется *механизм каскадных изменений.* Суть его довольно проста:

- При изменении значения поля связи в главной таблице должен быть синхронно изменен ключ, то есть его значение.
- При удалении записи в родительской таблице обязательно следует удалить все связанные записи.

Ограничения на изменение полей связи и их каскадное удаление могут быть наложены на таблицы при их создании. Эти ограничения обычно хранятся в системных таблицах наряду с индексами, триггерами и хранимыми процедурами. В некоторых случаях забота о сохранении ссылочной целостности ложиться на плечи разработчика.

Понятие транзакции

Транзакция – это последовательность действий с базой данных, в которой либо все действия выполняются успешно, либо не выполняется ни одно из них. Для того чтобы наглядно продемонстрировать суть транзакции, стоит рассмотреть простой пример. На склад пришла новая партия какого-либо товара. Необходимо принять его и занести информацию о нем в базу данных. Возникает некая цепочка действий:

- Увеличить количество единиц товара на складе.
- Ввести дату поступления новой партии.
- Ввести номер площадки, где будет храниться новая партия товара.

Предположим, что на последнем шаге произошла какая-то ошибка. Товар был зарегистрирован, его количество было увеличено, но место его расположения на складе потеряно. Такая ситуация недопустима. Транзакция должна выполняться полностью. Только тогда изменения сохранятся в базе данных. В противном случае, если один из операторов транзакции по какой-либо причине не был выполнен, измененные данные в базе данных не сохраняются, а транзакция отменяется.

Физически транзакция представляет собой последовательность команд, производящих с базой данных те или иные действия. Главная особенность транзакций заключается в том, что все действия должны выполниться, иначе будет отменена вся транзакция.

Транзакция может быть неявной и явной. *Неявная* транзакция стартует автоматически, а по завершении также автоматически подтверждается или

отменяется. Явной транзакцией управляет программист, используя для этого средства языка SQL.