Федеральное агентство по образованию

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра электронных приборов (ЭП)

Е.С. Шандаров

СИСТЕМЫ УПРАВЛЕНИЯ БАЗАМИ ДАННЫХ

Конспект лекций по курсу «Системы управления базами данных»
для студентов специальности 200300
«Электронные приборы и устройства»

Томск 2007

Оглавление

1. Базы данных и файловые системы....................4
1.1. Файловые системы....................6
1.2. Области применения файлов....................13
1.3. Потребности информационных систем....................14
2. Функции СУБД. Типовая организация СУБД. Примеры....................19
2.1. Основные функции СУБД....................19
2.2. Типовая организация современной СУБД....................27
3. Общие понятия реляционного подхода к организации БД. Основные концепции и термины....................29
3.1. Базовые понятия реляционных баз данных....................29
3.2. Фундаментальные свойства отношений....................33
3.3. Реляционная модель данных....................36
4. Базисные средства манипулирования реляционными данными....................40
4.1. Реляционная алгебра....................42
4.2. Реляционное исчисление....................52
5. Проектирование реляционных БД....................58
5.1. Проектирование реляционных баз данных с использованием нормализации…60
5.2. Семантическое моделирование данных, ER-диаграммы....................66
6. Структурированный язык запросов SQL....................77
6.1. История языка баз данных SQL....................78
6.2. Стандартизация SQL....................80
6.3. Современное состояние SQL....................81
Список литературы....................108

1. Базы данных и файловые системы

Рассмотрим общий смысл понятий БД и СУБД. Начнем с того, что с самого начала развития вычислительной техники образовались два основных направления ее использования. Первое направление - применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Становление этого направления способствовало интенсификации методов численного решения сложных математических задач, развитию класса языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ.

Второе направление, которое непосредственно касается темы нашего курса, это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера, выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т.д.

На самом деле, второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями в части памяти. Понятно, что можно говорить о надежном и долговременном хранении информации только при наличии запоминающих устройств, сохраняющих информацию после выключения электри-

ческого питания. Оперативная память этим свойством обычно не обладает. В начале использовались два вида устройств внешней памяти: магнитные ленты и барабаны. При этом емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные же барабаны (они больше всего похожи на современные магнитные диски с фиксированными головками) давали возможность произвольного доступа к данными, но были ограниченного размера.

Легко видеть, что указанные ограничения не очень существенны для чисто численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти, чтобы программа работала как можно быстрее.

С другой стороны, для информационных систем, в которых потребность в текущих данных определяется пользователем, наличие только магнитных лент и барабанов неудовлетворительно. Представьте себе покупателя билета, который стоя у кассы должен дождаться полной перемотки магнитной ленты. Одним из естественных требований к таким системам является средняя быстрота выполнения операций.

Как кажется, именно требования к вычислительной технике со стороны нечисленных приложений вызвали появление съемных магнитных дисков с подвижными головками, что явилось революцией в истории вычислительной техники. Эти устройства внешней памяти обладали существенно большей емкостью, чем магнитные барабаны, обеспечивали удовлетворительную скорость доступа к данным в режиме произвольной выборки, а возможность смены дискового пакета на устройстве позволяла иметь практически неограниченный архив данных.

С появлением магнитных дисков началась история систем управления данными во внешней памяти. До этого каждая прикладная программа, которой

требовалось хранить данные во внешней памяти, сама определяла расположение каждой порции данных на магнитной ленте или барабане и выполняла обмены между оперативной и внешней памятью с помощью программно-аппаратных средств низкого уровня (машинных команд или вызовов соответствующих программ операционной системы). Такой режим работы не позволяет или очень затрудняет поддержание на одном внешнем носителе нескольких архивов долговременно хранимой информации. Кроме того, каждой прикладной программе приходилось решать проблемы именования частей данных и структуризации данных во внешней памяти.

1.1. Файловые системы

Историческим шагом явился переход к использованию централизованных систем управления файлами. С точки зрения прикладной программы файл - это именованная область внешней памяти, в которую можно записывать и из которой можно считывать данные. Правила именования файлов, способ доступа к данным, хранящимся в файле, и структура этих данных зависят от конкретной системы управления файлами и, возможно, от типа файла. Система управления файлами берет на себя распределение внешней памяти, отображение имен файлов в соответствующие адреса во внешней памяти и обеспечение доступа к данным.

Первая развитая файловая система была разработана фирмой IBM для ее серии 360. К настоящему времени она очень устарела, и мы не будем рассматривать ее подробно. Заметим лишь, что в этой системе поддерживались как чисто последовательные, так и индексно-последовательные файлы, а реализация во многом опиралась на возможности только появившихся к этому времени контроллеров управления дисковыми устройствами. Если учесть к тому же, что понятие файла в OS/360 было выбрано как основное абстрактное понятие, которому соответствовал любой внешний объект, включая внешние устройства, то

работать с файлами на уровне пользователя было очень неудобно. Требовался целый ряд громоздких и перегруженных деталями конструкций.

1.1.1. Структуры файлов

Дальше мы будем говорить о более современных организациях файловых систем. Начнем со структур файлов. Прежде всего, практически во всех современных компьютерах основными устройствами внешней памяти являются магнитные диски с подвижными головками, и именно они служат для хранения файлов. Такие магнитные диски представляют собой пакеты магнитных пластин (поверхностей), между которыми на одном рычаге двигается пакет магнитных головок. Шаг движения пакета головок является дискретным, и каждому положению пакета головок логически соответствует цилиндр магнитного диска. На каждой поверхности цилиндр "высекает" дорожку, так что каждая поверхность содержит число дорожек, равное числу цилиндров. При разметке магнитного диска (специальном действии, предшествующем использованию диска) каждая дорожка размечается на одно и то же количество блоков таким образом, что в каждый блок можно записать по максимуму одно и то же число байтов. Таким образом, для произведения обмена с магнитным диском на уровне аппаратуры нужно указать номер цилиндра, номер поверхности, номер блока на соответствующей дорожке и число байтов, которое нужно записать или прочитать от начала этого блока.

Однако эта возможность обмениваться с магнитными дисками порциями меньше объема блока в настоящее время не используется в файловых системах. Это связано с двумя обстоятельствами. Во-первых, при выполнении обмена с диском аппаратура выполняет три основных действия: подвод головок к нужному цилиндру, поиск на дорожке нужного блока и собственно обмен с этим блоком. Из всех этих действий в среднем наибольшее время занимает первое. Поэтому существенный выигрыш в суммарном времени обмена за счет считы-

вания или записывания только части блока получить практически невозможно. Во-вторых, для того, чтобы работать с частями блоков, файловая система должна обеспечить соответствующего размера буфера оперативной памяти, что существенно усложняет распределение оперативной памяти.

Поэтому во всех файловых системах явно или неявно выделяется некоторый базовый уровень, обеспечивающий работу с файлами, представляющими набор прямо адресуемых в адресном пространстве файла блоков. Размер этих логических блоков файла совпадает или кратен размеру физического блока диска и обычно выбирается равным размеру страницы виртуальной памяти, поддерживаемой аппаратурой компьютера совместно с операционной системой.

В некоторых файловых системах базовый уровень доступен пользователю, но более часто прикрывается некоторым более высоким уровнем, стандартным для пользователей. Распространены два основных подхода. При первом подходе, свойственном, например, файловым системам операционных систем фирмы DEC RSX и VMS, пользователи представляют файл как последовательность записей. Каждая запись - это последовательность байтов постоянного или переменного размера. Записи можно читать или записывать последовательно или позиционировать файл на запись с указанным номером. Некоторые файловые системы позволяют структурировать записи на поля и объявлять некоторые поля ключами записи. В таких файловых системах можно потребовать выборку записи из файла по ее заданному ключу. Естественно, что в этом случае файловая система поддерживает в том же (или другом, служебном) базовом файле дополнительные, невидимые пользователю, служебные структуры данных. Распространенные способы организации ключевых файлов основываются на технике хэширования и B-деревьев (мы будем говорить об этих приемах более подробно в следующих лекциях). Существуют и многоключевые способы организации файлов.

Второй подход, ставший распространенным вместе с операционной си-

стемой UNIX, состоит в том, что любой файл представляется как последовательность байтов. Из файла можно прочитать указанное число байтов либо начиная с его начала, либо предварительно произведя его позиционирование на байт с указанным номером. Аналогично, можно записать указанное число байтов в конец файла, либо предварительно произведя позиционирование файла. Заметим, что тем не менее скрытым от пользователя, но существующим во всех разновидностях файловых систем ОС UNIX, является базовое блочное представление файла.

Конечно, для обоих подходов можно обеспечить набор преобразующих функций, приводящих представление файла к некоторому другому виду. Примером тому служит поддержание стандартной файловой среды системы программирования на языке Си в среде операционных систем фирмы DEC.

1.1.2. Именование файлов

Остановимся коротко на способах именования файлов. Все современные файловые системы поддерживают многоуровневое именование файлов за счет поддержания во внешней памяти дополнительных файлов со специальной структурой - каталогов. Каждый каталог содержит имена каталогов и/или файлов, содержащихся в данном каталоге. Таким образом, полное имя файла состоит из списка имен каталогов плюс имя файла в каталоге, непосредственно содержащем данный файл. Разница между способами именования файлов в разных файловых системах состоит в том, с чего начинается эта цепочка имен.

В этом отношении имеются два крайних варианта. Во многих системах управления файлами требуется, чтобы каждый архив файлов (полное дерево справочников) целиком располагался на одном дисковом пакете (или логическом диске, разделе физического дискового пакета, представляемом с помощью средств операционной системы как отдельный диск). В этом случае полное имя файла начинается с имени дискового устройства, на котором установлен соот-

ветствующий диск. Такой способ именования используется в файловых системах фирмы DEC, очень близко к этому находятся и файловые системы персональных компьютеров. Можно назвать эту организацию поддержанием изолированных файловых систем.

Другой крайний вариант был реализован в файловых системах операционной системы Multics. Эта система заслуживает отдельного большого разговора, в ней был реализован целый ряд оригинальных идей, но мы остановимся только на особенностях организации архива файлов. В файловой системе Miltics пользователи представляли всю совокупность каталогов и файлов как единое дерево. Полное имя файла начиналось с имени корневого каталога, и пользователь не обязан был заботиться об установке на дисковое устройство каких-либо конкретных дисков. Сама система, выполняя поиск файла по его имени, запрашивала установку необходимых дисков. Такую файловую систему можно назвать полностью централизованной.

Конечно, во многом централизованные файловые системы удобнее изолированных: система управления файлами принимает на себя больше рутинной работы. Но в таких системах возникают существенные проблемы, если кому-то требуется перенести поддерево файловой системы на другую вычислительную установку. Компромиссное решение применено в файловых системах ОС UNIX. На базовом уровне в этих файловых системах поддерживаются изолированные архивы файлов. Один из этих архивов объявляется корневой файловой системой. После запуска системы можно "смонтировать" корневую файловую систему и ряд изолированных файловых систем в одну общую файловую систему. Технически это производится с помощью заведения в корневой файловой системе специальных пустых каталогов. Специальный системный вызов курьер ОС UNIX позволяет подключить к одному из этих пустых каталогов корневой каталог указанного архива файлов. После монтирования общей файловой системы именование файлов производится так же, как если бы она с самого нача-

ла была централизованной. Если учесть, что обычно монтирование файловой системы производится при раскрутке системы, то пользователи ОС UNIX обычно и не задумываются об исходном происхождении общей файловой системы.

1.1.3. Защита файлов

Поскольку файловые системы являются общим хранилищем файлов, принадлежащих, вообще говоря, разным пользователям, системы управления файлами должны обеспечивать авторизацию доступа к файлам. В общем виде подход состоит в том, что по отношению к каждому зарегистрированному пользователю данной вычислительной системы для каждого существующего файла указываются действия, которые разрешены или запрещены данному пользователю. Существовали попытки реализовать этот подход в полном объеме. Но это вызывало слишком большие накладные расходы как по хранению избыточной информации, так и по использованию этой информации для контроля правомочности доступа.

Поэтому в большинстве современных систем управления файлами применяется подход к защите файлов, впервые реализованный в ОС UNIX. В этой системе каждому зарегистрированному пользователю соответствует пара целочисленных идентификаторов: идентификатор группы, к которой относится этот пользователь, и его собственный идентификатор в группе. Соответственно, при каждом файле хранится полный идентификатор пользователя, который создал этот файл, и отмечается, какие действия с файлом может производить он сам, какие действия с файлом доступны для других пользователей той же группы, и что могут делать с файлом пользователи других групп. Эта информация очень компактна, при проверке требуется небольшое количество действий, и этот способ контроля доступа удовлетворителен в большинстве случаев.

1.1.4. Режим многопользовательского доступа

Последнее, на чем мы остановимся в связи с файлами, - это способы их использования в многопользовательской среде. Если операционная система поддерживает многопользовательский режим, вполне реальна ситуация, когда два или более пользователей одновременно пытаются работать с одним и тем же файлом. Если все эти пользователи собираются только читать файл, ничего страшного не произойдет. Но если хотя бы один из них будет изменять файл, для корректной работы этой группы требуется взаимная синхронизация.

Исторически в файловых системах применялся следующий подход. В операции открытия файла (первой и обязательной операции, с которой должен начинаться сеанс работы с файлом) помимо прочих параметров указывался режим работы (чтение или изменение). Если к моменту выполнения этой операции от имени некоторой программы А файл уже находился в открытом состоянии от имени некоторой другой программы В (правильнее говорить "процесса", но мы не будем вдаваться в терминологические тонкости), причем существующий режим открытия был несовместимым с желаемым режимом (совместимы только режимы чтения), то в зависимости от особенностей системы программе А либо сообщалось о невозможности открытия файла в желаемом режиме, либо она блокировалась до тех пор, пока программа В не выполнит операцию закрытия файла.

Заметим, что в ранних версиях файловой системы ОС UNIX вообще не были реализованы какие бы то ни было средства синхронизации параллельного доступа к файлам. Операция открытия файла выполнялась всегда для любого существующего файла, если данный пользователь имел соответствующие права доступа. При совместной работе синхронизацию следовало производить вне файловой системы (и особых средств для этого ОС UNIX не предоставляла). В современных реализациях файловых систем ОС UNIX по желанию пользователя поддерживается синхронизация при открытии файлов. Кроме того, суще-

ствует возможность синхронизации нескольких процессов, параллельно модифицирующих один и тот же файл. Для этого введен специальный механизм синхронизационных захватов диапазонов адресов открытого файла.

1.2. Области применения файлов

После этого краткого экскурса в историю файловых систем рассмотрим возможные области их применения. Прежде всего, конечно, файлы применяются для хранения текстовых данных: документов, текстов программ и т.д. Такие файлы обычно образуются и модифицируются с помощью различных текстовых редакторов. Структура текстовых файлов обычно очень проста: это либо последовательность записей, содержащих строки текста, либо последовательность байтов, среди которых встречаются специальные символы (например, символы конца строки).

Файлы с текстами программ используются как входные тексты компиляторов, которые в свою очередь формируют файлы, содержащие объектные модули. С точки зрения файловой системы, объектные файлы также обладают очень простой структурой - последовательность записей или байтов. Система программирования накладывает на эту структуру более сложную и специфичную для этой системы структуру объектного модуля. Подчеркнем, что логическая структура объектного модуля неизвестна файловой системе, эта структура поддерживается программами системы программирования.

Аналогично обстоит дело с файлами, формируемыми редакторами связей и содержащими образы выполняемых программ. Логическая структура таких файлов остается известной только редактору связей и загрузчику - программе операционной системы. Примерно такая же ситуация с файлами, содержащими графическую и звуковую информацию.

Одним словом, файловые системы обычно обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию при-

кладным программам. В перечисленных выше случаях использования файлов это даже хорошо, потому что при разработке любой новой прикладной системы опираясь на простые, стандартные и сравнительно дешевые средства файловой системы можно реализовать те структуры хранения, которые наиболее естественно соответствуют специфике данной прикладной области.

1.3. Потребности информационных систем

Однако ситуация коренным образом отличается для упоминавшихся в начале лекции информационных систем. Эти системы главным образом ориентированы на хранение, выбор и модификацию постоянно существующей информации. Структура информации зачастую очень сложна, и хотя структуры данных различны в разных информационных системах, между ними часто бывает много общего. На начальном этапе использования вычислительной техники для управления информацией проблемы структуризации данных решались индивидуально в каждой информационной системе. Производились необходимые надстройки над файловыми системами (библиотеки программ), подобно тому, как это делается в компиляторах, редакторах и т.д.

Но поскольку информационные системы требуют сложных структур данных, эти дополнительные индивидуальные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Стремление выделить и обобщить общую часть информационных систем, ответственную за управление сложно структурированными данными, явилось, на наш взгляд, первой побудительной причиной создания СУБД. Очень скоро стало понятно, что невозможно обойтись общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данных.

Покажем это на примере. Предположим, что мы хотим реализовать простую информационную систему, поддерживающую учет сотрудников некото-

рой организации. Система должна выполнять следующие действия: выдавать списки сотрудников по отделам, поддерживать возможность перевода сотрудника из одного отдела в другой, приема на работу новых сотрудников и увольнения работающих. Для каждого отдела должна поддерживаться возможность получения имени руководителя этого отдела, общей численности отдела, общей суммы выплаченной в последний раз зарплаты и т.д. Для каждого сотрудника должна поддерживаться возможность выдачи номера удостоверения по полному имени сотрудника, выдачи полного имени по номеру удостоверения, получения информации о текущем соответствии занимаемой должности сотрудника и о размере его зарплаты.

Предположим, что мы решили основывать эту информационную систему на файловой системе и пользоваться при этом одним файлом, расширив базовые возможности файловой системы за счет специальной библиотеки функций. Поскольку минимальной информационной единицей в нашем случае является сотрудник, естественно потребовать, чтобы в этом файле содержалась одна запись для каждого сотрудника. Какие поля должна содержать такая запись? Полное имя сотрудника (СОТР_ИМЯ), номер его удостоверения (СОТР_НОМЕР), информацию о его соответствии занимаемой должности (для простоты, "да" или "нет") (СОТР_СТАТ), размер зарплаты (СОТР_ЗАРП), номер отдела (СОТР_ОТД_НОМЕР). Поскольку мы хотим ограничиться одним файлом, та же запись должна содержать имя руководителя отдела (СОТР_ОТД_РУК).

Функции нашей информационной системы требуют, чтобы обеспечивалась возможность многоключевого доступа к этому файлу по уникальным ключам (недублируемым в разных записях) СОТР_ИМЯ и СОТР_НОМЕР. Кроме того, должна обеспечиваться возможность выбора всех записей с общем значением СОТР_ОТД_НОМЕР, то есть доступ по неуникальному ключу. Для того, чтобы получить численность отдела или общий размер зарплаты, каждый раз при выполнении такой функции информационная система должна будет вы-

брать все записи о сотрудниках отдела и посчитать соответствующие общие значения.

Таким образом мы видим, что даже для такой простой системы ее реализация на базе файловой системы, во-первых, требует создания достаточно сложной надстройки для многоключевого доступа к файлам, и, во-вторых, вызывает требование существенной избыточности хранения (для каждого сотрудника одного отдела повторяется имя руководителя) и выполнение массовой выборки и вычислений для получения суммарной информации об отделах. Кроме того, если в ходе эксплуатации системы нам захочется, например, выдавать списки сотрудников, получающих заданную зарплату, то придется либо полностью просматривать файл, либо реструктуризовывать его, объявляя ключевым поле СОТР_ЗАРП.

Первое, что приходит на ум, - это поддерживать два многоключевых файла: СОТРУДНИКИ и ОТДЕЛЫ. Первый файл должен содержать поля СОТР_ИМЯ, СОТР_НОМЕР, СОТР_СТАТ, СОТР_ЗАРП и СОТР_ОТД_НОМЕР, а второй - ОТД_НОМЕР, ОТД_РУК, ОТД_СОТР_ЗАРП (общий размер зарплаты) и ОТД_РАЗМЕР (общее число сотрудников в отделе). Большинство неудобств, перечисленных в предыдущем абзаце, будут преодолены. Каждый из файлов будет содержать только недублируемую информацию, необходимости в динамических вычислениях суммарной информации не возникает. Но заметим, что при таком переходе наша информационная система должна обладать некоторыми новыми особенностями, сближающими ее с СУБД.

Прежде всего, система должна теперь знать, что она работает с двумя информационно связанными файлами (это шаг в сторону схемы базы данных), должна знать структуру и смысл каждого поля (например, что СОТР_ОТД_НОМЕР в файле СОТРУДНИКИ и ОТД_НОМЕР в файле ОТДЕЛЫ означают одно и то же), а также понимать, что в ряде случаев изменение информации в одном файле должно автоматически вызывать модификацию во втором файле, чтобы

их общее содержимое было согласованным. Например, если на работу принимается новый сотрудник, то необходимо добавить запись в файл СОТРУДНИКИ, а также соответствующим образом изменить поля ОТД_ЗАРП и ОТД_РАЗМЕР в записи файла ОТДЕЛЫ, описывающей отдел этого сотрудника.

Понятие согласованности данных является ключевым понятием баз данных. Фактически, если информационная система (даже такая простая, как в нашем примере) поддерживает согласованное хранение информации в нескольких файлах, можно говорить о том, что она поддерживает базу данных. Если же некоторая вспомогательная система управления данными позволяет работать с несколькими файлами, обеспечивая их согласованность, можно назвать ее системой управления базами данных. Уже только требование поддержания согласованности данных в нескольких файлах не позволяет обойтись библиотекой функций: такая система должна иметь некоторые собственные данные (метаданные) и даже знания, определяющие целостность данных.

Но это еще не все, что обычно требуют от СУБД. Во-первых, даже в нашем примере неудобно реализовывать такие запросы как "выдать общую численность отдела, в котором работает Петр Иванович Сидоров". Было бы гораздо проще, если бы СУБД позволяла сформулировать такой запрос на близком пользователям языке. Такие языки называются языками запросов к базам данных. Например, на языке SQL наш запрос можно было бы выразить в форме:

```
SELECT ОТД_РАЗМЕР
FROM СОТРУДНИКИ, ОТДЕЛЫ
WHERE СОТР_ИМЯ = "ПЕТР ИВАНОВИЧ СИДОРОВ"
AND СОТР_ОТД_НОМЕР = ОТД_НОМЕР
```

Таким образом, при формулировании запроса СУБД позволит не задумываться о том, как будет выполняться этот запрос. Среди ее метаданных будет содержаться информация о том, что поле СОТР_ИМЯ является ключевым для файла СОТРУДНИКИ, а ОТД_НОМЕР - для файла ОТДЕЛЫ, и система сама восполь-

зуется этим. Если же возникнет потребность в получении списка сотрудников, не соответствующих занимаемой должности, то достаточно предъявить системе запрос

```
SELECT СОТР_ИМЯ, СОТР_НОМЕР
FROM СОТРУДНИКИ
WHERE СОТР_СТАТ = "НЕТ",
```

и система сама выполнит необходимый полный просмотр файла СОТРУДНИКИ, поскольку поле СОТР_СТАТ не является ключевым.

Далее, представьте себе, что в нашей первоначальной реализации информационной системы, основанной на использовании библиотек расширенных методов доступа к файлам, обрабатывается операция регистрации нового сотрудника. Следуя требованиям согласованного изменения файлов, информационная система вставила новую запись в файл СОТРУДНИКИ и собиралась модифицировать запись файла ОТДЕЛЫ, но именно в этот момент произошло аварийное выключение питания. Очевидно, что после перезапуска системы ее база данных будет находиться в рассогласованном состоянии. Потребуется выяснить это (а для этого нужно явно проверить соответствие информации с файлах СОТРУДНИКИ и ОТДЕЛЫ) и привести информацию в согласованное состояние. Настоящие СУБД берут такую работу на себя. Прикладная система не обязана заботиться о корректности состояния базы данных.

Наконец, представим себе, что мы хотим обеспечить параллельную (например, многотерминальную) работу с базой данных сотрудников. Если опираться только на использование файлов, то для обеспечения корректности на все время модификации любого из двух файлов доступ других пользователей к этому файлу будет блокирован (вспомните возможности файловых систем для синхронизации параллельного доступа). Таким образом, зачисление на работу Петра Ивановича Сидорова существенно затормозит получение информации о сотруднике Иване Сидоровиче Петрове, даже если они будут работать в разных отделах.

Настоящие СУБД обеспечивают гораздо более тонкую синхронизацию параллельного доступа к данным.

Таким образом, СУБД решают множество проблем, которые затруднительно или вообще невозможно решить при использовании файловых систем. При этом существуют приложения, для которых вполне достаточно файлов; приложения, для которых необходимо решать, какой уровень работы с данными во внешней памяти для них требуется, и приложения, для которых безусловно нужны базы данных.

2. Функции СУБД. Типовая организация СУБД. Примеры

Как было показано ранее, традиционных возможностей файловых систем оказывается недостаточно для построения даже простых информационных систем. Мы выявили несколько потребностей, которые не покрываются возможностями систем управления файлами: поддержание логически согласованного набора файлов; обеспечение языка манипулирования данными; восстановление информации после разного рода сбоев; реально параллельная работа нескольких пользователей. Можно считать, что если прикладная информационная система опирается на некоторую систему управления данными, обладающую этими свойствами, то эта система управления данными является системой управления базами данных (СУБД).

2.1. Основные функции СУБД

Более точно, к числу функций СУБД принято относить следующие:

2.1.1. Непосредственное управление данными во внешней памяти

Эта функция включает обеспечение необходимых структур внешней памяти как для хранения данных, непосредственно входящих в БД, так и для слу-

жебных целей, например, для убыстрения доступа к данным в некоторых случаях (обычно для этого используются индексы). В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других работа производится вплоть до уровня устройств внешней памяти. Но подчеркнем, что в развитых СУБД пользователи в любом случае не обязаны знать, использует ли СУБД файловую систему, и если использует, то как организованы файлы. В частности, СУБД поддерживает собственную систему именования объектов БД.

2.1.2. Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

Заметим, что существует отдельное направление СУБД, которое ориентировано на постоянное присутствие в оперативной памяти всей БД. Это направление основывается на предположении, что в будущем объем оперативной памяти компьютеров будет настолько велик, что позволит не беспокоиться о буферизации. Пока эти работы находятся в стадии исследований.

2.1.3. Управление транзакциями

Транзакция - это последовательность операций над БД, рассматриваемых СУБД как единое целое. Либо транзакция успешно выполняется, и СУБД фиксирует (COMMIT) изменения БД, произведенные этой транзакцией, во внешней памяти, либо ни одно из этих изменений никак не отражается на состоянии БД. Понятие транзакции необходимо для поддержания логической целостности БД. Если вспомнить наш пример информационной системы с файлами СОТРУДНИКИ и ОТДЕЛЫ, то единственным способом не нарушить целостность БД при выполнении операции приема на работу нового сотрудника является объединение элементарных операций над файлами СОТРУДНИКИ и ОТДЕЛЫ в одну транзакцию. Таким образом, поддержание механизма транзакций является обязательным условием даже однопользовательских СУБД (если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

То свойство, что каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения, делает очень удобным использование понятия транзакции как единицы активности пользователя по отношению к БД. При соответствующем управлении параллельно выполняющимися транзакциями со стороны СУБД каждый из пользователей может в принципе ощущать себя единственным пользователем СУБД (на самом деле, это несколько идеализированное представление, поскольку в некоторых случаях пользователи многопользовательских СУБД могут ощутить присутствие своих коллег).

С управлением транзакциями в многопользовательской СУБД связаны важные понятия сериализации транзакций и сериального плана выполнения смеси транзакций. Под сериализаций параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последователь-

ного выполнения. Сериальный план выполнения смеси транзакций - это такой план, который приводит к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Существует несколько базовых алгоритмов сериализации транзакций. В централизованных СУБД наиболее распространены алгоритмы, основанные на синхронизационных захватах объектов БД. При использовании любого алгоритма сериализации возможны ситуации конфликтов между двумя или более транзакциями по доступу к объектам БД. В этом случае для поддержания сериализации необходимо выполнить откат (ликвидировать все изменения, произведенные в БД) одной или более транзакций. Это один из случаев, когда пользователь многопользовательской СУБД может реально (и достаточно неприятно) ощутить присутствие в системе транзакций других пользователей.

2.1.4. Журнализация

Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя. Обычно рассматриваются два возможных вида аппаратных сбоев: так называемые мягкие сбои, которые можно трактовать как внезапную остановку работы компьютера (например, аварийное выключение питания), и жесткие сбои, характеризуемые потерей информации на носителях внешней памяти. Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция

остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал - это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД. В разных СУБД изменения БД журнализуются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

Во всех случаях придерживаются стратегии "упреждающей" записи в журнал (так называемого протокола Write Ahead Log - WAL). Грубо говоря, эта стратегия заключается в том, что запись об изменении любого объекта БД должна попасть во внешнюю память журнала раньше, чем измененный объект попадет во внешнюю память основной части БД. Известно, что если в СУБД корректно соблюдается протокол WAL, то с помощью журнала можно решить все проблемы восстановления БД после любого сбоя.

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал опе-

раций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).

При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту сбоя, и могут отсутствовать объекты, модифицированные транзакциями, которые к моменту сбоя успешно завершились (по причине использования буферов оперативной памяти, содержимое которых при мягком сбое пропадает). При соблюдении протокола WAL во внешней памяти журнала должны гарантированно находиться записи, относящиеся к операциям модификации обоих видов объектов. Целью процесса восстановления после мягкого сбоя является состояние внешней памяти основной части БД, которое возникло бы при фиксации во внешней памяти изменений всех завершившихся транзакций и которое не содержало бы никаких следов незаконченных транзакций. Для того, чтобы этого добиться, сначала производят откат незавершенных транзакций (undo), а потом повторно воспроизводят (redo) те операции завершенных транзакций, результаты которых не отображены во внешней памяти. Этот процесс содержит много тонкостей, связанных с общей организацией управления буферами и журналом. Более подробно мы рассмотрим это в соответствующей лекции.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повы-

шенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

2.1.5. Поддержка языков БД

Для работы с базами данных используются специальные языки, в целом называемые языками баз данных. В ранних СУБД поддерживалось несколько специализированных по своим функциям языков. Чаще всего выделялись два языка - язык определения схемы БД (SDL - Schema Definition Language) и язык манипулирования данными (DML - Data Manipulation Language). SDL служил главным образом для определения логической структуры БД, т.е. той структуры БД, какой она представляется пользователям. DML содержал набор операторов манипулирования данными, т.е. операторов, позволяющих заносить данные в БД, удалять, модифицировать или выбирать существующие данные. Мы рассмотрим более подробно языки ранних СУБД в следующих разделах.

В современных СУБД обычно поддерживается единый интегрированный язык, содержащий все необходимые средства для работы с БД, начиная от ее создания, и обеспечивающий базовый пользовательский интерфейс с базами данных. Стандартным языком наиболее распространенных в настоящее время реляционных СУБД является язык SQL (Structured Query Language). В нескольких лекциях этого курса язык SQL будет рассматриваться достаточно подробно, а пока мы перечислим основные функции реляционной СУБД, поддерживаемые на "языковом" уровне (т.е. функции, поддерживаемые при реализации интерфейса SQL).

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет

определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

Язык SQL содержит специальные средства определения ограничений целостности БД. Опять же, ограничения целостности хранятся в специальных таблицах-каталогах, и обеспечение контроля целостности БД производится на языковом уровне, т.е. при компиляции операторов модификации БД компилятор SQL на основании имеющихся в БД ограничений целостности генерирует соответствующий программный код.

Специальные операторы языка SQL позволяют определять так называемые представления БД, фактически являющиеся хранимыми в БД запросами (результатом любого запроса к реляционной БД является таблица) с именованными столбцами. Для пользователя представление является такой же таблицей, как любая базовая таблица, хранимая в БД, но с помощью представлений можно ограничить или наоборот расширить видимость БД для конкретного пользователя. Поддержание представлений производится также на языковом уровне.

Наконец, авторизация доступа к объектам БД производится также на основе специального набора операторов SQL. Идея состоит в том, что для выполнения операторов SQL разного вида пользователь должен обладать различными полномочиями. Пользователь, создавший таблицу БД, обладает полным набором полномочий для работы с этой таблицей. В число этих полномочий входит полномочие на передачу всех или части полномочий другим пользователям, включая полномочие на передачу полномочий. Полномочия пользователей описываются в специальных таблицах-каталогах, контроль полномочий под-

держивается на языковом уровне.

Более точное описание возможных реализаций этих функций на основе языка SQL будет приведено в разделах, посвященных языку SQL и его реализации.

2.2. Типовая организация современной СУБД

Естественно, организация типичной СУБД и состав ее компонентов соответствует рассмотренному нами набору функций. Напомним, что мы выделили следующие основные функции СУБД:

- управление данными во внешней памяти;
- управление буферами оперативной памяти;
- управление транзакциями;
- журнализация и восстановление БД после сбоев;
- поддержание языков БД.

Логически в современной реляционной СУБД можно выделить наиболее внутреннюю часть - ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения, набор утилит. В некоторых системах эти части выделяются явно, в других - нет, но логически такое разделение можно провести во всех СУБД.

Ядро СУБД отвечает за управление данными во внешней памяти, управление буферами оперативной памяти, управление транзакциями и журнализацию. Соответственно, можно выделить такие компоненты ядра (по крайней мере, логически, хотя в некоторых системах эти компоненты выделяются явно), как менеджер данных, менеджер буферов, менеджер транзакций и менеджер журнала. Как можно было понять из первой части этой лекции, функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все эти компоненты должны взаимодействовать по тщательно продуманным и про-

веренным протоколам. Ядро СУБД обладает собственным интерфейсом, не доступным пользователям напрямую и используемым в программах, производимых компилятором SQL (или в подсистеме поддержки выполнения таких программ) и утилитах БД. Ядро СУБД является основной резидентной частью СУБД. При использовании архитектуры "клиент-сервер" ядро является основной составляющей серверной части системы.

Основной функцией компилятора языка БД является компиляция операторов языка БД в некоторую выполняемую программу. Основной проблемой реляционных СУБД является то, что языки этих систем (а это, как правило, SQL) являются непроцедурными, т.е. в операторе такого языка специфицируется некоторое действие над БД, но эта спецификация не является процедурой, а лишь описывает в некоторой форме условия совершения желаемого действия (вспомните примеры из первой лекции). Поэтому компилятор должен решить, каким образом выполнять оператор языка прежде, чем произвести программу. Применяются достаточно сложные методы оптимизации операторов, которые мы подробно рассмотрим в следующих лекциях. Результатом компиляции является выполняемая программа, представляемая в некоторых системах в машинных кодах, но более часто в выполняемом внутреннем машинно-независимом коде. В последнем случае реальное выполнение оператора производится с привлечением подсистемы поддержки времени выполнения, представляющей собой, по сути дела, интерпретатор этого внутреннего языка.

Наконец, в отдельные утилиты БД обычно выделяют такие процедуры, которые слишком накладно выполнять с использованием языка БД, например, загрузка и выгрузка БД, сбор статистики, глобальная проверка целостности БД и т.д. Утилиты программируются с использованием интерфейса ядра СУБД, а иногда даже с проникновением внутрь ядра.

3. Общие понятия реляционного подхода к организации БД. Основные концепции и термины

В данном разделе мы введем на сравнительно неформальном уровне основные понятия реляционных баз данных, а также определим существо реляционной модели данных. Основной целью лекции является демонстрация простоты и возможности интуитивной интерпретации этих понятий. В дальнейшем будут приводиться более формальные определения, на которых основывается математическая теория реляционных баз данных.

3.1. Базовые понятия реляционных баз данных

Основными понятиями реляционных баз данных являются тип данных, домен, атрибут, кортеж, первичный ключ и отношение.

Для начала покажем смысл этих понятий на примере отношения СОТРУДНИКИ, содержащего информацию о сотрудниках некоторой организации:

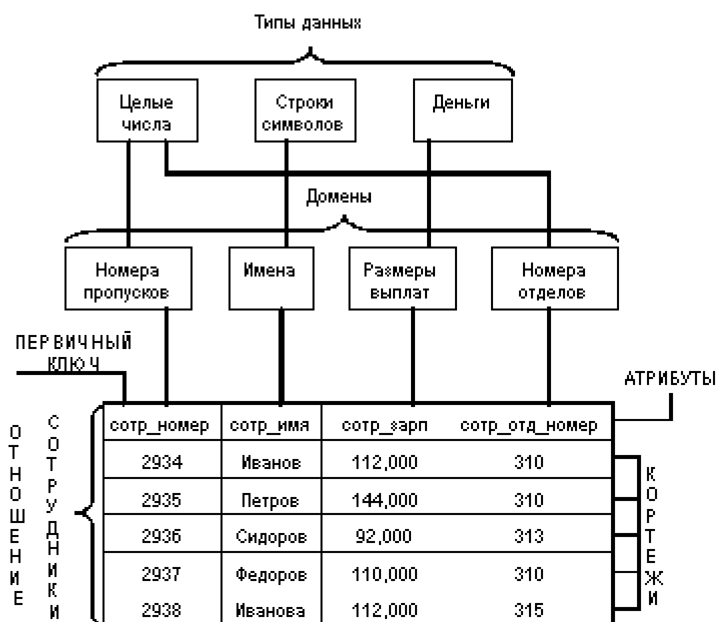


Рис. 3.1. Отношение СОТРУДНИКИ

3.1.1. Тип данных

Понятие тип данных в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как "деньги"), а также специальных "темпоральных" данных (дата, время, временной интервал). Достаточно активно развивается подход к расширению возможностей реляционных систем абстрактными типами данных (соответствующими возможностями обладают, например, системы семейства Ingres/Postgres). В нашем примере мы имеем дело с данными трех типов: строки символов, целые числа и "деньги".

3.1.2. Домен

Понятие домена более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных. Если вычисление этого логического выражения дает результат "истина", то элемент данных является элементом домена.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа. Например, домен "Имена" в нашем примере определен на базовом типе строк символов, но в число его значений могут входить только те строки, которые могут изображать имя (в частности, такие строки не могут начинаться с мягкого знака).

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному

домену. В нашем примере значения доменов "Номера пропусков" и "Номера групп" относятся к типу целых чисел, но не являются сравнимыми. Заметим, что в большинстве реляционных СУБД понятие домена не используется, хотя в Oracle V.7 оно уже поддерживается.

3.1.3. Схема отношения, схема базы данных

Схема отношения - это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}. Степень или "арность" схемы отношения - мощность этого множества. Степень отношения СОТРУДНИКИ равна четырем, то есть оно является 4-арным. Если все атрибуты одного отношения определены на разных доменах, осмысленно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что это является всего лишь удобным способом именования и не устраняет различия между понятиями домена и атрибута).

Схема БД (в структурном смысле) - это набор именованных схем отношений.

3.1.4. Кортеж, отношение

Кортеж, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отноше-

ние-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. На самом деле, понятие схемы отношения ближе всего к понятию структурного типа данных в языках программирования. Было бы вполне логично разрешать отдельно определять схему отношения, а затем одно или несколько отношений с данной схемой.

Однако в реляционных базах данных это не принято. Имя схемы отношения в таких базах данных всегда совпадает с именем соответствующего отношения-экземпляра. В классических реляционных базах данных после определения схемы базы данных изменяются только отношения-экземпляры. В них могут появляться новые и удаляться или модифицироваться существующие кортежи. Однако во многих реализациях допускается и изменение схемы базы данных: определение новых и изменение существующих схем отношения. Это принято называть эволюцией схемы базы данных.

Обычным житейским представлением отношения является таблица, заголовком которой является схема отношения, а строками - кортежи отношения-экземпляра; в этом случае имена атрибутов именуют столбцы этой таблицы. Поэтому иногда говорят "столбец таблицы", имея в виду "атрибут отношения". Когда мы перейдем к рассмотрению практических вопросов организации реляционных баз данных и средств управления, мы будем использовать эту житейскую терминологию. Этой терминологии придерживаются в большинстве коммерческих реляционных СУБД.

Реляционная база данных - это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Как видно, основные структурные понятия реляционной модели данных (если не считать понятия домена) имеют очень простую интуитивную интерпретацию, хотя в теории реляционных БД все они определяются абсолютно формально и точно.

3.2. Фундаментальные свойства отношений

Остановимся теперь на некоторых важных свойствах отношений, которые следуют из приведенных ранее определений:

3.2.1. Отсутствие кортежей-дубликатов

То свойство, что отношения не содержат кортежей-дубликатов, следует из определения отношения как множества кортежей. В классической теории множеств по определению каждое множество состоит из различных элементов.

Из этого свойства вытекает наличие у каждого отношения так называемого первичного ключа - набора атрибутов, значения которых однозначно определяют кортеж отношения. Для каждого отношения по крайней мере полный набор его атрибутов обладает этим свойством. Однако при формальном определении первичного ключа требуется обеспечение его "минимальности", т.е. в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства - однозначно определять кортеж. Понятие первичного ключа является исключительно важным в связи с понятием целостности баз данных.

Забегая вперед, заметим, что во многих практических реализациях РСУБД допускается нарушение свойства уникальности кортежей для промежуточных отношений, порождаемых неявно при выполнении запросов. Такие отношения являются не множествами, а мультимножествами, что в ряде случаев позволяет добиться определенных преимуществ, но иногда приводит к серьезным проблемам.

3.2.2. Отсутствие упорядоченности кортежей

Свойство отсутствия упорядоченности кортежей отношения также яв-

ляется следствием определения отношения-экземпляра как множества кортежей. Отсутствие требования к поддержанию порядка на множестве кортежей отношения дает дополнительную гибкость СУБД при хранении баз данных во внешней памяти и при выполнении запросов к базе данных. Это не противоречит тому, что при формулировании запроса к БД, например, на языке SQL можно потребовать сортировки результирующей таблицы в соответствии со значениями некоторых столбцов. Такой результат, вообще говоря, не отношение, а некоторый упорядоченный список кортежей.

3.2.3. Отсутствие упорядоченности атрибутов

Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар {имя атрибута, имя домена}. Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута. Это свойство теоретически позволяет, например, модифицировать схемы существующих отношений не только путем добавления новых атрибутов, но и путем удаления существующих атрибутов. Однако в большинстве существующих систем такая возможность не допускается, и хотя упорядоченность набора атрибутов отношения явно не требуется, часто в качестве неявного порядка атрибутов используется их порядок в линейной форме определения схемы отношения.

3.2.4. Атомарность значений атрибутов

Значения всех атрибутов являются атомарными. Это следует из определения домена как потенциального множества значений простого типа данных, т.е. среди значений домена не могут содержаться множества значений (отношения). Принято говорить, что в реляционных базах данных допускаются только нормализованные отношения или отношения, представленные в первой нормальной форме. Потенциальным примером ненормализованного отношения является следующее:

НОМЕР_ОТДЕЛА	ОТДЕЛ		
	СОТР_НОМЕР	СОТР_ИМЯ	СОТР_ЗАРП
310	2934	Иванов	112,000
	2935	Петров	112,500
313	2937	Федоров	110,000
315	2938	Иванова	112,000

Рис. 3.2. Пример ненормализованного отношения

Можно сказать, что здесь мы имеем бинарное отношение, значениями атрибута ОТДЕЛЫ которого являются отношения. Заметим, что исходное отношение СОТРУДНИКИ является нормализованным вариантом отношения ОТДЕЛЫ:

СОТР_НОМЕР	СОТР_ИМЯ	СОТР_ЗАРП	СОТР_ОТД_НОМЕР
2934	Иванов	112,000	310
2935	Петров	144,000	310
2936	Сидоров	92,000	313
2937	Федоров	110,000	310
2938	Иванова	112,000	315

Нормализованные отношения составляют основу классического реляци-

онного подхода к организации баз данных. Они обладают некоторыми ограничениями (не любую информацию удобно представлять в виде плоских таблиц), но существенно упрощают манипулирование данными. Рассмотрим, например, два идентичных оператора занесения кортежа:

Зачислить сотрудника Ксенофонтова (пропуск номер 3000, зарплата 11,000) в отдел номер 320 и

Зачислить сотрудника Ксенофонтова (пропуск номер 3000, зарплата 11,000) в отдел номер 310.

Если информация о сотрудниках представлена в виде отношения СОТРУДНИКИ, оба оператора будут выполняться одинаково (вставить кортеж в отношение СОТРУДНИКИ). Если же работать с ненормализованным отношением ОТДЕЛЫ, то первый оператор выразится в занесение кортежа, а второй - в добавление информации о Ксенофонтове в множественное значение атрибута ОТДЕЛ кортежа с первичным ключом 310.

3.3. Реляционная модель данных

Когда в предыдущих разделах мы говорили об основных понятиях реляционных баз данных, мы не опирались на какую-либо конкретную реализацию. Эти рассуждения в равной степени относились к любой системе, при построении которой использовался реляционный подход.

Другими словами, мы использовали понятия так называемой реляционной модели данных. Модель данных описывает некоторый набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, если они основываются на этой модели. Наличие модели данных позволяет сравнивать конкретные реализации, используя один общий язык.

Хотя понятие модели данных является общим, и можно говорить о иерар-

хической, сетевой, некоторой семантической и т.д. моделях данных, нужно отметить, что это понятие было введено в обиход применительно к реляционным системам и наиболее эффективно используется именно в этом контексте. Попытки прямолинейного применения аналогичных моделей к дореляционным организациям показывают, что реляционная модель слишком "велика" для них, а для постреляционных организаций она оказывается "мала".

3.3.1. Общая характеристика

Наиболее распространенная трактовка реляционной модели данных, по-видимому, принадлежит Дейту, который воспроизводит ее (с различными уточнениями) практически во всех своих книгах. Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной части, манипуляционной части и целостной части.

В структурной части модели фиксируется, что единственной структурой данных, используемой в реляционных БД, является нормализованное n-арное отношение. По сути дела, в предыдущих двух разделах этой лекции мы рассматривали именно понятия и свойства структурной составляющей реляционной модели.

В манипуляционной части модели утверждаются два фундаментальных механизма манипулирования реляционными БД - реляционная алгебра и реляционное исчисление. Первый механизм базируется в основном на классической теории множеств (с некоторыми уточнениями), а второй - на классическом логическом аппарате исчисления предикатов первого порядка. Мы рассмотрим эти механизмы более подробно на следующей лекции, а пока лишь заметим, что основной функцией манипуляционной части реляционной модели является обеспечение меры реляционности любого конкретного языка реляционных БД: язык называется реляционным, если он обладает не меньшей выразительностью и мощностью, чем реляционная алгебра или реляционное исчисление.

3.3.2. Целостность сущности и ссылок

Наконец, в целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД. Первое требование называется требованием целостности сущностей. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е. другими словами, любое отношение должно обладать первичным ключом. Как мы видели в предыдущем разделе, это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Второе требование называется требованием целостности по ссылкам и является несколько более сложным. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений. Например, представим, что нам требуется представить в реляционной базе данных сущность ОТДЕЛ с атрибутами ОТД_НОМЕР (номер отдела), ОТД_КОЛ (количество сотрудников) и ОТД_СОТР (набор сотрудников отдела). Для каждого сотрудника нужно хранить СОТР_НОМЕР (номер сотрудника), СОТР_ИМЯ (имя сотрудника) и СОТР_ЗАРП (заработная плата сотрудника). Как мы вскоре увидим, при правильном проектировании соответствующей БД в ней появятся два отношения: ОТДЕЛЫ (ОТД_НОМЕР, ОТД_КОЛ) (первичный ключ - ОТД_НОМЕР) и СОТРУДНИКИ (СОТР_НОМЕР, СОТР_ИМЯ, СОТР_ЗАРП, СОТР_ОТД_НОМ) (первичный ключ - СОТР_НОМЕР).

Как видно, атрибут СОТР_ОТД_НОМ появляется в отношении СОТРУДНИКИ не потому, что номер отдела является собственным свойством сотрудника, а лишь для того, чтобы иметь возможность восстановить при необходимости полную сущность ОТДЕЛ. Значение атрибута СОТР_ОТД_НОМ в любом кортеже отношения СОТРУДНИКИ должно соответствовать значению атрибу-

та ОТД_НОМ в некотором кортеже отношения ОТДЕЛЫ. Атрибут такого рода называется внешним ключом, поскольку его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения (т.е. задают значения их первичного ключа). Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором такой же атрибут является первичным ключом.

Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать). Для нашего примера это означает, что если для сотрудника указан номер отдела, то этот отдел должен существовать.

Ограничения целостности сущности и по ссылкам должны поддерживаться СУБД. Для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа. С целостностью по ссылкам дела обстоят несколько более сложно.

Понятно, что при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа. Но как быть при удалении кортежа из отношения, на которое ведет ссылка?

Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение

внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

В развитых реляционных СУБД обычно можно выбрать способ поддержания целостности по ссылкам для каждой отдельной ситуации определения внешнего ключа. Конечно, для принятия такого решения необходимо анализировать требования конкретной прикладной области.

4. Базисные средства манипулирования реляционными данными

Итак, существует три составляющих реляционной модели данных. Две из них - структурную и целостную составляющие - мы рассмотрели более или менее подробно, а манипуляционной части реляционной модели данных посвящается эта глава.

Как уже было отмечено, в манипуляционной составляющей определяются два базовых механизма манипулирования реляционными данными - основанная на теории множеств реляционная алгебра и базирующееся на математической логике (точнее, на исчислении предикатов первого порядка) реляционное исчисление. В свою очередь, обычно рассматриваются два вида реляционного исчисления - исчисление доменов и исчисление предикатов.

Все эти механизмы обладают одним важным свойством: они замкнуты относительно понятия отношения. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатом вычисления также являются отношения. В результате любое выражение или формула могут интерпретироваться как отношения, что позволяет использовать их в других выражениях или формулах.

Как будет показано в дальнейшем, алгебра и исчисление обладают

большой выразительной мощностью: очень сложные запросы к базе данных могут быть выражены с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления. Именно по этой причине именно эти механизмы включены в реляционную модель данных. Конкретный язык манипулирования реляционными БД называется реляционно полным, если любой запрос, выражаемый с помощью одного выражения реляционной алгебры или одной формулы реляционного исчисления, может быть выражен с помощью одного оператора этого языка.

Известно (и мы не будем это доказывать), что механизмы реляционной алгебры и реляционного исчисления эквивалентны, т.е. для любого допустимого выражения реляционной алгебры можно построить эквивалентную (т.е. производящую такой же результат) формулу реляционного исчисления и наоборот. Почему же в реляционной модели данных присутствуют оба эти механизма?

Дело в том, что они различаются уровнем процедурности. Выражения реляционной алгебры строятся на основе алгебраических операций (высокого уровня), и подобно тому, как интерпретируются арифметические и логические выражения, выражение реляционной алгебры также имеет процедурную интерпретацию. Другими словами, запрос, представленный на языке реляционной алгебры, может быть вычислен на основе вычисления элементарных алгебраических операций с учетом их старшинства и возможного наличия скобок. Для формулы реляционного исчисления однозначная интерпретация, вообще говоря, отсутствует. Формула только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения. Поэтому языки реляционного исчисления являются более непроцедурными или декларативными.

Поскольку механизмы реляционной алгебры и реляционного исчисления эквивалентны, то в конкретной ситуации для проверки степени реляционности некоторого языка БД можно пользоваться любым из этих механизмов.

Заметим, что крайне редко алгебра или исчисление принимаются в качестве полной основы какого-либо языка БД. Обычно (как, например, в случае языка SQL) язык основывается на некоторой смеси алгебраических и логических конструкций. Тем не менее, знание алгебраических и логических основ языков баз данных часто бывает полезно на практике.

В нашем изложении мы в основном следуем подходу Дейта, примененному (хотя и не изобретенному) им в последнем издании книги "Введение в системы баз данных". Для экономии времени и места мы не будем вводить каких-либо строгих синтаксических конструкций, а в основном ограничимся рассмотрением материала на содержательном уровне.

4.1. Реляционная алгебра

Основная идея реляционной алгебры состоит в том, что коль скоро отношения являются множествами, то средства манипулирования отношениями могут базироваться на традиционных теоретико-множественных операциях, дополненных некоторыми специальными операциями, специфичными для баз данных.

Существует много подходов к определению реляционной алгебры, которые различаются набором операций и способами их интерпретации, но в принципе, более или менее равносильны. Мы опишем немного расширенный начальный вариант алгебры, который был предложен Коддом. В этом варианте набор основных алгебраических операций состоит из восьми операций, которые делятся на два класса - теоретико-множественные операции и специальные реляционные операции. В состав теоретико-множественных операций входят операции:

- объединения отношений;
- пересечения отношений;
- взятия разности отношений;

- прямого произведения отношений.

Специальные реляционные операции включают:

- ограничение отношения;
- проекцию отношения;
- соединение отношений;
- деление отношений.

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 заголовок (схему) результирующего отношения.

4.1.1. Общая интерпретация реляционных операций

Если не вдаваться в некоторые тонкости, которые мы рассмотрим в следующих подразделах, то почти все операции предложенного выше набора обладают очевидной и простой интерпретацией.

- При выполнении операции объединения двух отношений производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов.
- Операция пересечения двух отношений производит отношение, включающее все кортежи, входящие в оба отношения-операнда.
- Отношение, являющееся разностью двух отношений включает все кортежи, входящие в отношение - первый операнд, такие, что ни один из них не входит в отношение, являющееся вторым операндом.
- При выполнении прямого произведения двух отношений производится отношение, кортежи которого являются конкатенацией (сцеплением) кортежей первого и второго операндов.
- Результатом ограничения отношения по некоторому условию является от-

ношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.

- При выполнении проекции отношения на заданный набор его атрибутов производится отношение, кортежи которого производятся путем взятия соответствующих значений из кортежей отношения-операнда.
- При соединении двух отношений по некоторому условию образуется результирующее отношение, кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию.
- У операции реляционного деления два операнда - бинарное и унарное отношения. Результирующее отношение состоит из одноатрибутных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) совпадает со множеством значений второго операнда.
- Операция переименования производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены.
- Операция присваивания позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Поскольку результатом любой реляционной операции (кроме операции присваивания) является некоторое отношение, можно образовывать реляционные выражения, в которых вместо отношения-операнда некоторой реляционной операции находится вложенное реляционное выражение.

4.1.2. Замкнутость реляционной алгебры и операция переименования

Каждое отношение характеризуется схемой (или заголовком) и набором кортежей (или телом). Поэтому, если действительно желать иметь алгебру, операции которой замкнуты относительно понятия отношения, то каждая операция должна производить отношение в полном смысле, т.е. оно должно обладать и

телом, и заголовком. Только в этом случае будет действительно возможно строить вложенные выражения.

Заголовок отношения представляет собой множество пар <имя-атрибута, имя-домена>. Если посмотреть на общий обзор реляционных операций, приведенный в предыдущем подразделе, то видно, что домены атрибутов результирующего отношения однозначно определяются доменами отношений-операндов. Однако с именами атрибутов результата не всегда все так просто.

Например, представим себе, что у отношений-операндов операции прямого произведения имеются одноименные атрибуты с одинаковыми доменами. Каким был бы заголовок результирующего отношения? Поскольку это множество, в нем не должны содержаться одинаковые элементы. Но и потерять атрибут в результате недопустимо. А это значит, что в этом случае вообще невозможно корректно выполнить операцию прямого произведения.

Аналогичные проблемы могут возникать и в случаях других двуместных операций. Для их разрешения в состав операций реляционной алгебры вводится операция переименования. Ее следует применять в любом случае, когда возникает конфликт именования атрибутов в отношениях - операндах одной реляционной операции. Тогда к одному из операндов сначала применяется операция переименования, а затем основная операция выполняется уже безо всяких проблем.

В дальнейшем изложении мы будем предполагать применение операции переименования во всех конфликтных случаях.

4.1.3. Особенности теоретико-множественных операций реляционной алгебры

Хотя в основе теоретико-множественной части реляционной алгебры лежит классическая теория множеств, соответствующие операции реляционной алгебры обладают некоторыми особенностями.

Начнем с операции объединения (все, что будет говориться по поводу объединения, переносится на операции пересечения и взятия разности). Смысл операции объединения в реляционной алгебре в целом остается теоретико-множественным. Но если в теории множеств операция объединения осмысленна для любых двух множеств-операндов, то в случае реляционной алгебры результатом операции объединения должно являться отношение. Если допустить в реляционной алгебре возможность теоретико-множественного объединения произвольных двух отношений (с разными схемами), то, конечно, результатом операции будет множество, но множество разнотипных кортежей, т.е. не отношение. Если исходить из требования замкнутости реляционной алгебры относительно понятия отношения, то такая операция объединения является бессмысленной.

Все эти соображения приводят к появлению понятия совместимости отношений по объединению: два отношения совместимы по объединению в том и только в том случае, когда обладают одинаковыми заголовками. Более точно, это означает, что в заголовках обоих отношений содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене.

Если два отношения совместимы по объединению, то при обычном выполнении над ними операций объединения, пересечения и взятия разности результатом операции является отношение с корректно определенным заголовком, совпадающим с заголовком каждого из отношений-операндов. Напомним,

что если два отношения "почти" совместимы по объединению, т.е. совместимы во всем, кроме имен атрибутов, то до выполнения операции типа соединения эти отношения можно сделать полностью совместимыми по объединению путем применения операции переименования.

Заметим, что включение в состав операций реляционной алгебры трех операций объединения, пересечения и взятия разности является очевидно избыточным, поскольку известно, что любая из этих операций выражается через две других. Тем не менее, Кодд в свое время решил включить все три операции, исходя из интуитивных потребностей потенциального пользователя системы реляционных БД, далекого от математики.

Другие проблемы связаны с операцией взятия прямого произведения двух отношений. В теории множеств прямое произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств. Поскольку отношения являются множествами, то и для любых двух отношений возможно получение прямого произведения. Но результат не будет отношением! Элементами результата будут являться не кортежи, а пары кортежей.

Поэтому в реляционной алгебре используется специализированная форма операции взятия прямого произведения - расширенное прямое произведение отношений. При взятии расширенного прямого произведения двух отношений элементом результирующего отношения является кортеж, являющийся конкатенацией (или слиянием) одного кортежа первого отношения и одного кортежа второго отношения.

Но теперь возникает второй вопрос - как получить корректно сформированный заголовок отношения-результата? Очевидно, что проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами.

Эти соображения приводят к появлению понятия совместимости по взя-

тию расширенного прямого произведения. Два отношения совместимы по взятию прямого произведения в том и только в том случае, если множества имен атрибутов этих отношений не пересекаются. Любые два отношения могут быть сделаны совместимыми по взятию прямого произведения путем применения операции переименования к одному из этих отношений.

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

По поводу теоретико-множественных операций реляционной алгебры следует еще заметить, что все четыре операции являются ассоциативными. Т. е., если обозначить через OP любую из четырех операций, то (A OP B) OP C = A (B OP C), и следовательно, без введения двусмысленности можно писать A OP B OP C (A, B и C - отношения, обладающие свойствами, требуемыми для корректного выполнения соответствующей операции). Все операции, кроме взятия разности, являются коммутативными, т.е. A OP B = B OP A.

4.1.4. Специальные реляционные операции

В этом подразделе мы несколько подробнее рассмотрим специальные реляционные операции реляционной алгебры: ограничение, проекция, соединение и деление.

Операция ограничения

Операция ограничения требует наличия двух операндов: ограничиваемого отношения и простого условия ограничения. Простое условие ограничения

может иметь либо вид (a comp-op b), где a и b - имена атрибутов ограничиваемого отношения, для которых осмысленна операция сравнения comp-op, либо вид (a comp-op const), где a - имя атрибута ограничиваемого отношения, а const - литерально заданная константа.

В результате выполнения операции ограничения производится отношение, заголовок которого совпадает с заголовком отношения-операнда, а в тело входят те кортежи отношения-операнда, для которых значением условия ограничения является true.

Пусть UNION обозначает операцию объединения, INTERSECT - операцию пересечения, а MINUS - операцию взятия разности. Для обозначения операции ограничения будем использовать конструкцию A WHERE comp, где A - ограничиваемое отношение, а comp - простое условие сравнения. Пусть comp1 и comp2 - два простых условия ограничения. Тогда по определению:

- A WHERE comp1 AND comp2 обозначает то же самое, что и (A WHERE comp1) INTERSECT (A WHERE comp2)
- A WHERE comp1 OR comp2 обозначает то же самое, что и (A WHERE comp1) UNION (A WHERE comp2)
- A WHERE NOT comp1 обозначает то же самое, что и A MINUS (A WHERE comp1)

С использованием этих определений можно использовать операции ограничения, в которых условием ограничения является произвольное булевское выражение, составленное из простых условий с использованием логических связок AND, OR, NOT и скобок.

На интуитивном уровне операцию ограничения лучше всего представлять как взятие некоторой "горизонтальной" вырезки из отношения-операнда.

Операция взятия проекции

Операция взятия проекции также требует наличия двух операндов -

проецируемого отношения A и списка имен атрибутов, входящих в заголовок отношения A.

Результатом проекции отношения A по списку атрибутов a1, a2, ..., an является отношение, с заголовком, определяемым множеством атрибутов a1, a2, ..., an, и с телом, состоящим из кортежей вида таких, что в отношении A имеется кортеж, атрибут a1 которого имеет значение v1, атрибут a2 имеет значение v2, ..., атрибут an имеет значение vn. Тем самым, при выполнении операции проекции выделяется "вертикальная" вырезка отношения-операнда с естественным уничтожением потенциально возникающих кортежей-дубликатов.

Операция соединения отношений

Общая операция соединения (называемая также соединением по условию) требует наличия двух операндов - соединяемых отношений и третьего операнда - простого условия. Пусть соединяются отношения A и B. Как и в случае операции ограничения, условие соединения comp имеет вид либо (a comp-op b), либо (a comp-op const), где a и b - имена атрибутов отношений A и B, const - литерально заданная константа, а comp-op - допустимая в данном контексте операция сравнения.

Тогда по определению результатом операции сравнения является отношение, получаемое путем выполнения операции ограничения по условию comp прямого произведения отношений A и B.

Если внимательно осмыслить это определение, то станет ясно, что в общем случае применение условия соединения существенно уменьшит мощность результата промежуточного прямого произведения отношений-операндов только в том случае, когда условие соединения имеет вид (a comp-op b), где a и b - имена атрибутов разных отношений-операндов. Поэтому на практике обычно считают реальными операциями соединения именно те операции, которые основываются на условии соединения приведенного вида.

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

Имеется важный частный случай соединения - эквисоединение и простое, но важное расширение операции эквисоединения - естественное соединение. Операция соединения называется операцией эквисоединения, если условие соединения имеет вид (a = b), где a и b - атрибуты разных операндов соединения. Этот случай важен потому, что (a) он часто встречается на практике, и (b) для него существуют эффективные алгоритмы реализации.

Операция естественного соединения применяется к паре отношений A и B, обладающих (возможно составным) общим атрибутом c (т.е. атрибутом с одним и тем же именем и определенным на одном и том же домене). Пусть ab обозначает объединение заголовков отношений A и B. Тогда естественное соединение A и B - это спроектированный на ab результат эквисоединения A и B по A/c и BBC. Если вспомнить введенное нами в конце предыдущей главы определение внешнего ключа отношения, то должно стать понятно, что основной смысл операции естественного соединения - возможность восстановления сложной сущности, декомпозированной по причине требования первой нормальной формы. Операция естественного соединения не включается прямо в состав набора операций реляционной алгебры, но она имеет очень важное практическое значение.

Операция деления отношений

Эта операция наименее очевидна из всех операций реляционной алгебры

и поэтому нуждается в более подробном объяснении. Пусть заданы два отношения - A с заголовком {a1, a2, ..., an, b1, b2, ..., bm} и B с заголовком {b1, b2, ..., bm}. Будем считать, что атрибут bi отношения A и атрибут bi отношения B не только обладают одним и тем же именем, но и определены на одном и том же домене. Назовем множество атрибутов {aj} составным атрибутом a, а множество атрибутов {bj} - составным атрибутом b. После этого будем говорить о реляционном делении бинарного отношения A(a,b) на унарное отношение B(b).

Результатом деления A на B является унарное отношение C(a), состоящее из кортежей v таких, что в отношении A имеются кортежи такие, что множество значений {w} включает множество значений атрибута b в отношении B.

Предположим, что в базе данных сотрудников поддерживаются два отношения: СОТРУДНИКИ (ИМЯ, ОТД_НОМЕР) и ИМЕНА (ИМЯ), причем унарное отношение ИМЕНА содержит все фамилии, которыми обладают сотрудники организации. Тогда после выполнения операции реляционного деления отношения СОТРУДНИКИ на отношение ИМЕНА будет получено унарное отношение, содержащее номера отделов, сотрудники которых обладают всеми возможными в этой организации именами.

4.2. Реляционное исчисление

Предположим, что мы работаем с базой данных, обладающей схемой СОТРУДНИКИ (СОТР_НОМ, СОТР_ИМЯ, СОТР_ЗАРП, ОТД_НОМ) и ОТДЕЛЫ (ОТД_НОМ, ОТД_КОЛ, ОТД_НАЧ), и хотим узнать имена и номера сотрудников, являющихся начальниками отделов с количеством сотрудников больше 50.

Если бы для формулировки такого запроса использовалась реляционная алгебра, то мы получили бы алгебраическое выражение, которое читалось бы, например, следующим образом:

- выполнить соединение отношений СОТРУДНИКИ и ОТДЕЛЫ по усло-

вию СОТР_НОМ = ОТД_НАЧ;

- ограничить полученное отношение по условию ОТД_КОЛ > 50;
- спроецировать результат предыдущей операции на атрибут СОТР_ИМЯ, СОТР_НОМ.

Мы четко сформулировали последовательность шагов выполнения запроса, каждый из которых соответствует одной реляционной операции. Если же сформулировать тот же запрос с использованием реляционного исчисления, которому посвящается этот раздел, то мы получили бы формулу, которую можно было бы прочитать, например, следующим образом: Выдать СОТР_ИМЯ и СОТР_НОМ для сотрудников таких, что существует отдел с таким же значением ОТД_НАЧ и значением ОТД_КОЛ большим 50.

Во второй формулировке мы указали лишь характеристики результирующего отношения, но ничего не сказали о способе его формирования. В этом случае система должна сама решить, какие операции и в каком порядке нужно выполнить над отношениями СОТРУДНИКИ и ОТДЕЛЫ. Обычно говорят, что алгебраическая формулировка является процедурной, т.е. задающей правила выполнения запроса, а логическая - описательной (или декларативной), поскольку она всего лишь описывает свойства желаемого результата. Как мы указывали в начале лекции, на самом деле эти два механизма эквивалентны и существуют не очень сложные правила преобразования одного формализма в другой.

4.2.1. Кортежные переменные и правильно построенные формулы

Реляционное исчисление является прикладной ветвью формального механизма исчисления предикатов первого порядка. Базисными понятиями исчисления являются понятие переменной с определенной для нее областью допустимых значений и понятие правильно построенной формулы, опирающейся на переменные, предикаты и кванторы.

В зависимости от того, что является областью определения переменной, различаются исчисление кортежей и исчисление доменов. В исчислении кортежей областями определения переменных являются отношения базы данных, т.е. допустимым значением каждой переменной является кортеж некоторого отношения. В исчислении доменов областями определения переменных являются домены, на которых определены атрибуты отношений базы данных, т.е. допустимым значением каждой переменной является значение некоторого домена. Мы рассмотрим более подробно исчисление кортежей, а в конце лекции коротко опишем особенности исчисления доменов.

В отличие от раздела, посвященного реляционной алгебре, в этом разделе нам не удастся избежать использования некоторого конкретного синтаксиса, который мы, тем не менее, формально определять не будем. Необходимые синтаксические конструкции будут вводиться по мере необходимости. В совокупности, используемый синтаксис близок, но не полностью совпадает с синтаксисом языка баз данных QUEL, который долгое время являлся основным языком СУБД Ingres.

Для определения кортежной переменной используется оператор RANGE. Например, для того, чтобы определить переменную СОТРУДНИК, областью определения которой является отношение СОТРУДНИКИ, нужно употребить конструкцию

```
RANGE СОТРУДНИК IS СОТРУДНИКИ
```

Как мы уже говорили, из этого определения следует, что в любой момент времени переменная СОТРУДНИК представляет некоторый кортеж отношения СОТРУДНИКИ. При использовании кортежных переменных в формулах можно ссылаться на значение атрибута переменной (это аналогично тому, как, например, при программировании на языке Си можно сослаться на значение поля структурной переменной). Например, для того, чтобы сослаться на значение атрибута СОТР_ИМЯ переменной СОТРУДНИК, нужно употребить

конструкцию СОТРУДНИК.СОТР_ИМЯ.

Правильно построенные формулы (WFF - Well-Formed Formula) служат для выражения условий, накладываемых на кортежные переменные. Основой WFF являются простые сравнения (comparison), представляющие собой операции сравнения скалярных значений (значений атрибутов переменных или литерально заданных констант). Например, конструкция "СОТРУДНИК.СОТР_НОМ = 140" является простым сравнением. По определению, простое сравнение является WFF, а WFF, заключенная в круглые скобки, является простым сравнением.

Более сложные варианты WFF строятся с помощью логических связок NOT, AND, OR и IF ... THEN. Так, если form - WFF, a comp - простое сравнение, то NOT form, comp AND form, comp OR form и IF comp THEN form являются WFF.

Наконец, допускается построение WFF с помощью кванторов. Если form - это WFF, в которой участвует переменная var, то конструкции EXISTS var (form) и FORALL var (form) представляют wff.

Переменные, входящие в WFF, могут быть свободными или связанными. Все переменные, входящие в WFF, при построении которой не использовались кванторы, являются свободными. Фактически, это означает, что если для какого-то набора значений свободных кортежных переменных при вычислении WFF получено значение true, то эти значения кортежных переменных могут входить в результирующее отношение. Если же имя переменной использовано сразу после квантора при построении WFF вида EXISTS var (form) или FORALL var (form), то в этой WFF и во всех WFF, построенных с ее участием, var - это связанная переменная. Это означает, что такая переменная не видна за пределами минимальной WFF, связавшей эту переменную. При вычислении значения такой WFF используется не одно значение связанной переменной, а вся ее область определения.

Пусть СОТР1 и СОТР2 - две кортежные переменные, определенные на отношении СОТРУДНИКИ. Тогда, WFF EXISTS СОТР2 (СОТР1.СОТР_ЗАРП > СОТР2.СОТР_ЗАРП) для текущего кортежа переменной СОТР1 принимает значение true в том и только в том случае, если во всем отношении СОТРУДНИКИ найдется кортеж (связанный с переменной СОТР2) такой, что значение его атрибута СОТР_ЗАРП удовлетворяет внутреннему условию сравнения. WFF FORALL СОТР2 (СОТР1.СОТР_ЗАРП > СОТР2.СОТР_ЗАРП) для текущего кортежа переменной СОТР1 принимает значение true в том и только в том случае, если для всех кортежей отношения СОТРУДНИКИ (связанных с переменной СОТР2) значения атрибута СОТР_ЗАРП удовлетворяют условию сравнения.

На самом деле, правильнее говорить не о свободных и связанных переменных, а о свободных и связанных вхождениях переменных. Легко видеть, что если переменная var является связанной в WFF form, то во всех WFF, включающих данную, может использоваться имя переменной var, которая может быть свободной или связанной, но в любом случае не имеет никакого отношения к вхождению переменной var в WFF form. Вот пример:

```
  EXISTS СОТР2 (СОТР1.СОТР_ОТД_НОМ = СОТР2.СОТР_ОТД_НОМ)
AND

    FORALL СОТР2 (СОТР1.СОТР_ЗАРП > СОТР2.СОТР_ЗАРП)
```

Здесь мы имеем два связанных вхождения переменной СОТР2 с совершенно разным смыслом.

4.2.2. Целевые списки и выражения реляционного исчисления

Итак, WFF обеспечивают средства формулировки условия выборки из отношений БД. Чтобы можно было использовать исчисление для реальной рабо-

ты с БД, требуется еще один компонент, который определяет набор и имена столбцов результирующего отношения. Этот компонент называется целевым списком (target_list).

Целевой список строится из целевых элементов, каждый из которых может иметь следующий вид:

- var.attr, где var - имя свободной переменной соответствующей WFF, а attr - имя атрибута отношения, на котором определена переменная var;
- var, что эквивалентно наличию подсписка var.attr1, var.attr2, ..., var.attrn, где attr1, attr2, ..., attrn включает имена всех атрибутов определяющего отношения;
- new_name = var.attr; new_name - новое имя соответствующего атрибута результирующего отношения.

Последний вариант требуется в тех случаях, когда в WFF используются несколько свободных переменных с одинаковой областью определения.

Выражением реляционного исчисления кортежей называется конструкция вида target_list WHERE wff. Значением выражения является отношение, тело которого определяется WFF, а набор атрибутов и их имена - целевым списком.

4.2.3. Реляционное исчисление доменов

В исчислении доменов областью определения переменных являются не отношения, а домены. Применительно к базе данных СОТРУДНИКИ-ОТДЕЛЫ можно говорить, например, о доменных переменных ИМЯ (значения - допустимые имена) или НОСОТР (значения - допустимые номера сотрудников).

Основным формальным отличием исчисления доменов от исчисления кортежей является наличие дополнительного набора предикатов, позволяющих выражать так называемые условия членства. Если R - это n-арное отношение с

атрибутами a1, a2, ..., an, то условие членства имеет вид

```
R (ai1:vi1, ai2:vi2, ..., aim:vim) (m <= n),
```

где vij - это либо литерально задаваемая константа, либо имя кортежной переменной. Условие членства принимает значение true в том и только в том случае, если в отношении R существует кортеж, содержащий указанные значения указанных атрибутов. Если vij - константа, то на атрибут aij задается жесткое условие, не зависящее от текущих значений доменных переменных; если же vij - имя доменной переменной, то условие членства может принимать разные значения при разных значениях этой переменной.

Во всех остальных отношениях формулы и выражения исчисления доменов выглядят похожими на формулы и выражения исчисления кортежей. В частности, конечно, различаются свободные и связанные вхождения доменных переменных.

Для примера сформулируем с использованием исчисления доменов запрос "Выдать номера и имена сотрудников, не получающих минимальную заработную плату" (будем считать для простоты, что мы определили доменные переменные, имена которых совпадают с именами атрибутов отношения СОТРУДНИКИ, а в случае, когда требуется несколько доменных переменных, определенных на одном домене, мы будем добавлять в конце имени цифры):

```
СОТР_НОМ, СОТР_ИМЯ WHERE EXISTS СОТР_ЗАРП1
  (СОТРУДНИКИ (СОТР_ЗАРП1) AND
   СОТРУДНИКИ (СОТР_НОМ, СОТР_ИМЯ, СОТР_ЗАРП) AND
   СОТР_ЗАРП > СОТР_ЗАРП1)
```

Реляционное исчисление доменов является основой большинства языков запросов, основанных на использовании форм. В частности, на этом исчислении базировался известный язык Query-by-Example, который был первым (и наиболее интересным) языком в семействе языков, основанных на табличных

формах.

5. Проектирование реляционных БД

При проектировании базы данных решаются две основных проблемы:

- Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было по возможности лучшим (эффективным, удобным и т.д.)? Часто эту проблему называют проблемой логического проектирования баз данных.
- Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур (например, индексов) потребовать и т.д.? Эту проблему называют проблемой физического проектирования баз данных.

В случае реляционных баз данных трудно представить какие-либо общие рецепты по части физического проектирования. Здесь слишком много зависит от используемой СУБД. Например, при работе с СУБД Ingres можно выбирать один из предлагаемых способов физической организации отношений, при работе с System R следовало бы прежде всего подумать о кластеризации отношений и требуемом наборе индексов и т.д. Поэтому мы ограничимся вопросами логического проектирования реляционных баз данных, которые существенны при использовании любой реляционной СУБД.

Более того, мы не будем касаться очень важного аспекта проектирования - определения ограничений целостности (за исключением ограничения первичного ключа). Дело в том, что при использовании СУБД с развитыми механизмами ограничений целостности (например, SQL-ориентированных систем) трудно предложить какой-либо общий подход к определению ограничений целостно-

сти. Эти ограничения могут иметь очень общий вид, и их формулировка пока относится скорее к области искусства, чем инженерного мастерства. Самое большее, что предлагается по этому поводу в литературе, это автоматическая проверка непротиворечивости набора ограничений целостности.

Так что будем считать, что проблема проектирования реляционной базы данных состоит в обоснованном принятии решений о том,

- из каких отношений должна состоять БД и
- какие атрибуты должны быть у этих отношений.

5.1. Проектирование реляционных баз данных с использованием нормализации

Сначала будет рассмотрен классический подход, при котором весь процесс проектирования производится в терминах реляционной модели данных методом последовательных приближений к удовлетворительному набору схем отношений. Исходной точкой является представление предметной области в виде одного или нескольких отношений, и на каждом шаге проектирования производится некоторый набор схем отношений, обладающих лучшими свойствами. Процесс проектирования представляет собой процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы - значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, мы будем считать, что исходный набор отношений уже соответствует этому требо-

ванию.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм:

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);
- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

- каждая следующая нормальная форма в некотором смысле лучше предыдущей;
- при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

В основе процесса проектирования лежит метод нормализации, декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости. Для дальнейшего изложения нам потребуются несколько определений.

Определение 1. Функциональная зависимость

В отношении R атрибут Y функционально зависит от атрибута X (X и Y могут быть составными) в том и только в том случае, если каждому значению X соответствует в точности одно значение Y: R.X (r) R.Y.

Определение 2. Полная функциональная зависимость

Функциональная зависимость R.X (r) R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

Определение 3. Транзитивная функциональная зависимость

Функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X -> R.Z и R.Z -> R.Y и отсутствует функциональная зависимость R.Z --> R.X. (При отсутствии последнего требования мы имели бы "неинтересные" транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

Определение 4. Неключевой атрибут

Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа (в частности, первичного).

Определение 5. Взаимно независимые атрибуты

Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

5.1.1. Вторая нормальная форма

Рассмотрим следующий пример схемы отношения:

```
СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ
```

```
(СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)
```

Первичный ключ:

```
СОТР_НОМЕР, ПРО_НОМЕР
```

Функциональные зависимости:

```
СОТР_НОМЕР -> СОТР_ЗАРП
СОТР_НОМЕР -> ОТД_НОМЕР
ОТД_НОМЕР -> СОТР_ЗАРП
СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН
```

Как видно, хотя первичным ключом является составной атрибут СОТР_НОМЕР, ПРО_НОМЕР, атрибуты СОТР_ЗАРП и ОТД_НОМЕР функционально зависят от части первичного ключа, атрибута СОТР_НОМЕР. В результате мы не сможем вставить в отношение СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ кортеж, описывающий сотрудника, который еще не выполняет никакого проекта (первичный ключ не может содержать неопределенное значение). При удалении кортежа мы не только разрушаем связь данного сотрудника с данным проектом, но утрачиваем информацию о том, что он работает в некотором отделе. При переводе сотрудника в другой отдел мы будем вынуждены модифицировать все кортежи, описывающие этого сотрудника, или получим несогласованный результат. Такие неприятные явления называются аномалиями схемы отношения. Они устраняются путем нормализации.

Определение 6. Вторая нормальная форма (в этом определении предполагается, что единственным ключом отношения является первичный ключ)

Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда находится в 1NF, и каждый неключевой атрибут полностью зависит от первичного ключа.

Можно произвести следующую декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ в два отношения СОТРУДНИКИ-ОТДЕЛЫ и СОТРУДНИКИ-ПРОЕКТЫ:

```
СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, СОТР_ЗАРП, ОТД_НОМЕР)
Первичный ключ:
СОТР_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР -> СОТР_ЗАРП
СОТР_НОМЕР -> ОТД_НОМЕР
ОТД_НОМЕР -> СОТР_ЗАРП

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)
Первичный ключ:
СОТР_НОМЕР, ПРО_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР, ПРО_НОМЕР -> СОТР_ЗАДАН
```

Каждое из этих двух отношений находится в 2NF, и в них устранены отмеченные выше аномалии (легко проверить, что все указанные операции выполняются без проблем).

Если допустить наличие нескольких ключей, то определение 6 примет следующий вид:

Определение 6. Отношение R находится во второй нормальной форме (2NF) в том и только в том случае, когда оно находится в 1NF, и каждый неключевой атрибут полностью зависит от каждого ключа R.

Здесь и далее мы не будем приводить примеры для отношений с несколькими ключами. Они слишком громоздки и относятся к ситуациям, редко встречающимся на практике.

5.1.2. Третья нормальная форма

Рассмотрим еще раз отношение СОТРУДНИКИ-ОТДЕЛЫ, находящееся в

2NF. Заметим, что функциональная зависимость СОТР_НОМЕР -> СОТР_ЗАРП является транзитивной; она является следствием функциональных зависимостей СОТР_НОМЕР -> ОТД_НОМЕР и ОТД_НОМЕР -> СОТР_ЗАРП. Другими словами, заработная плата сотрудника на самом деле является характеристикой не сотрудника, а отдела, в котором он работает (это не очень естественное предположение, но достаточное для примера).

В результате мы не сможем занести в базу данных информацию, характеризующую заработную плату отдела, до тех пор, пока в этом отделе не появится хотя бы один сотрудник (первичный ключ не может содержать неопределенное значение). При удалении кортежа, описывающего последнего сотрудника данного отдела, мы лишимся информации о заработной плате отдела. Чтобы согласованным образом изменить заработную плату отдела, мы будем вынуждены предварительно найти все кортежи, описывающие сотрудников этого отдела. Т.е. в отношении СОТРУДИКИ-ОТДЕЛЫ по-прежнему существуют аномалии. Их можно устранить путем дальнейшей нормализации.

Определение 7. Третья нормальная форма. (Снова определение дается в предположении существования единственного ключа.)

Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 2NF и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Можно произвести декомпозицию отношения СОТРУДНИКИ-ОТДЕЛЫ в два отношения СОТРУДНИКИ и ОТДЕЛЫ:

```
СОТРУДНИКИ (СОТР_НОМЕР, ОТД_НОМЕР)
Первичный ключ:
СОТР_НОМЕР
Функциональные зависимости:
СОТР_НОМЕР -> ОТД_НОМЕР
```

```
ОТДЕЛЫ (ОТД_НОМЕР, СОТР_ЗАРП)
Первичный ключ:
ОТД_НОМЕР
Функциональные зависимости:
ОТД_НОМЕР -> СОТР_ЗАРП
```

Каждое из этих двух отношений находится в 3NF и свободно от отмеченных аномалий.

Если отказаться от того ограничения, что отношение обладает единственным ключом, то определение 3NF примет следующую форму:

Определение 7. Отношение R находится в третьей нормальной форме (3NF) в том и только в том случае, если находится в 1NF, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

На практике третья нормальная форма схем отношений достаточна в большинстве случаев, и приведением к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается.

5.2. Семантическое моделирование данных, ER-диаграммы

Широкое распространение реляционных СУБД и их использование в самых разнообразных приложениях показывает, что реляционная модель данных достаточна для моделирования предметных областей. Однако проектирование реляционной базы данных в терминах отношений на основе кратко рассмотренного нами механизма нормализации часто представляет собой очень сложный и неудобный для проектировщика процесс.

При этом проявляется ограниченность реляционной модели данных в следующих аспектах:

- Модель не предоставляет достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к упоминавшейся нами проблеме представления ограничений целостности.
- Для многих приложений трудно моделировать предметную область на основе плоских таблиц. В ряде случаев на самой начальной стадии проектирования проектировщику приходится производить насилие над собой, чтобы описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.
- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.
- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области ("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

5.2.1. Семантические модели данных

Потребности проектировщиков баз данных в более удобных и мощных средствах моделирования предметной области вызвали к жизни направление семантических моделей данных. При том, что любая развитая семантическая модель данных, как и реляционная модель, включает структурную, манипуляционную и целостную части, главным назначением семантических моделей является обеспечение возможности выражения семантики данных.

Прежде, чем мы коротко рассмотрим особенности одной из распространенных семантических моделей, остановимся на их возможных применениях.

Наиболее часто на практике семантическое моделирование используется на первой стадии проектирования базы данных. При этом в терминах семантической модели производится концептуальная схема базы данных, которая затем вручную преобразуется к реляционной (или какой-либо другой) схеме. Этот процесс выполняется под управлением методик, в которых достаточно четко оговорены все этапы такого преобразования.

Менее часто реализуется автоматизированная компиляция концептуальной схемы в реляционную. При этом известны два подхода: на основе явного представления концептуальной схемы как исходной информации для компилятора и построения интегрированных систем проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области. И в том, и в другом случае в результате производится реляционная схема базы данных в третьей нормальной форме (более точно следовало бы сказать, что автору неизвестны системы, обеспечивающие более высокий уровень нормализации).

Наконец, третья возможность, которая еще не вышла (или только выходит) за пределы исследовательских и экспериментальных проектов, - это работа с базой данных в семантической модели, т.е. СУБД, основанные на семантических моделях данных. При этом снова рассматриваются два варианта: обеспечение пользовательского интерфейса на основе семантической модели данных с автоматическим отображением конструкций в реляционную модель данных (это задача примерно такого же уровня сложности, как автоматическая компиляция концептуальной схемы базы данных в реляционную схему) и прямая реализация СУБД, основанная на какой-либо семантической модели данных. Наиболее близко ко второму подходу находятся современные объектно-ориентированные СУБД, модели данных которых по многим параметрам близки к семантическим моделям (хотя в некоторых аспектах они более мощны, а в некоторых - более слабы).

5.2.2. Основные понятия модели Entity-Relationship (Сущность-Связи)

Далее мы кратко рассмотрим некоторые черты одной из наиболее популярных семантических моделей данных - модель "Сущность-Связи" (часто ее называют кратко ER-моделью).

На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных (главным образом, реляционных). Модель была предложена Ченом (Chen) в 1976 г. Моделирование предметной области базируется на использовании графических диаграмм, включающих небольшое число разнородных компонентов. В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в системах CASE, поддерживающих автоматизированное проектирование реляционных баз данных. Среди множества разновидностей ER-моделей одна из наиболее развитых применяется в системе CASE фирмы ORACLE. Ее мы и рассмотрим. Более точно, мы сосредоточимся на структурной части этой модели.

Основными понятиями ER-модели являются сущность, связь и атрибут.

Сущность - это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. В диаграммах ER-модели сущность представляется в виде прямоугольника, содержащего имя сущности. При этом имя сущности - это имя типа, а не некоторого конкретного экземпляра этого типа. Для большей выразительности и лучшего понимания имя сущности может сопровождаться примерами конкретных объектов этого типа.

Ниже изображена сущность АЭРОПОРТ с примерными объектами Шереметьево и Хитроу:

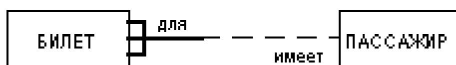
Каждый экземпляр сущности должен быть отличим от любого другого экземпляра той же сущности (это требование в некотором роде аналогично требованию отсутствия кортежей-дубликатов в реляционных таблицах).

Связь - это графически изображаемая ассоциация, устанавливаемая между двумя сущностями. Эта ассоциация всегда является бинарной и может существовать между двумя разными сущностями или между сущностью и ей же самой (рекурсивная связь). В любой связи выделяются два конца (в соответствии с существующей парой связываемых сущностей), на каждом из которых указывается имя конца связи, степень конца связи (сколько экземпляров данной сущности связывается), обязательность связи (т.е. любой ли экземпляр данной сущности должен участвовать в данной связи).

Связь представляется в виде линии, связывающей две сущности или ведущей от сущности к ней же самой. При это в месте "стыковки" связи с сущностью используются трехточечный вход в прямоугольник сущности, если для этой сущности в связи могут использоваться много (many) экземпляров сущности, и одноточечный вход, если в связи может участвовать только один экземпляр сущности. Обязательный конец связи изображается сплошной линией, а необязательный - прерывистой линией.

Как и сущность, связь - это типовое понятие, все экземпляры обеих пар связываемых сущностей подчиняются правилам связывания.

В изображенном ниже примере связь между сущностями БИЛЕТ и ПАССАЖИР связывает билеты и пассажиров. При том конец сущности с именем "для" позволяет связывать с одним пассажиром более одного билета, причем каждый билет должен быть связан с каким-либо пассажиром. Конец сущности с именем "имеет" означает, что каждый билет может принадлежать только одному пассажиру, причем пассажир не обязан иметь хотя бы один билет.



Лаконичной устной трактовкой изображенной диаграммы является следующая:

- Каждый БИЛЕТ предназначен для одного и только одного ПАССАЖИРА;
- Каждый ПАССАЖИР может иметь один или более БИЛЕТОВ.

На следующем примере изображена рекурсивная связь, связывающая сущность ЧЕЛОВЕК с ней же самой. Конец связи с именем "сын" определяет тот факт, что у одного отца может быть более чем один сын. Конец связи с именем "отец" означает, что не у каждого человека могут быть сыновья.

Лаконичной устной трактовкой изображенной диаграммы является следующая:

- Каждый ЧЕЛОВЕК является сыном одного и только одного ЧЕЛОВЕКА;
- Каждый ЧЕЛОВЕК может являться отцом для одного или более ЛЮДЕЙ ("ЧЕЛОВЕКОВ").

Атрибутом сущности является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности. Имена атрибутов заносятся в прямоугольник, изображающий сущность, под именем сущности и изображаются малыми буквами, возможно, с примерами.

Пример:

Уникальным идентификатором сущности является атрибут, комбинация атрибутов, комбинация связей или комбинация связей и атрибутов, уникально отличающая любой экземпляр сущности от других экземпляров сущности того же типа.

5.2.3. Нормальные формы ER-схем

Как и в реляционных схемах баз данных, в ER-схемах вводится понятие нормальных форм, причем их смысл очень близко соответствует смыслу реляционных нормальных форм. Заметим, что формулировки нормальных форм ER-схем делают более понятным смысл нормализации реляционных схем. Мы приведем только очень краткие и неформальные определения трех первых нормальных форм.

В первой нормальной форме ER-схемы устраняются повторяющиеся атрибуты или группы атрибутов, т.е. производится выявление неявных сущностей, "замаскированных" под атрибуты.

Во второй нормальной форме устраняются атрибуты, зависящие только от части уникального идентификатора. Эта часть уникального идентификатора определяет отдельную сущность.

В третьей нормальной форме устраняются атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор. Эти атрибуты являются основой отдельной сущности.

5.2.4. Более сложные элементы ER-модели

Мы остановились только на самых основных и наиболее очевидных понятиях ER-модели данных. К числу более сложных элементов модели относятся следующие:

- Подтипы и супертипы сущностей. Как в языках программирования с развитыми типовыми системами (например, в языках объектно-ориентированного программирования), вводится возможность наследования типа сущности, исходя из одного или нескольких супертипов. Интересные нюансы связаны с необходимостью графического изображения этого механизма.

- Связи "many-to-many". Иногда бывает необходимо связывать сущности таким образом, что с обоих концов связи могут присутствовать несколько экземпляров сущности (например, все члены кооператива сообща владеют имуществом кооператива). Для этого вводится разновидность связи "многие-со-многими".
- Уточняемые степени связи. Иногда бывает полезно определить возможное количество экземпляров сущности, участвующих в данной связи (например, служащему разрешается участвовать не более, чем в трех проектах одновременно). Для выражения этого семантического ограничения разрешается указывать на конце связи ее максимальную или обязательную степень.
- Каскадные удаления экземпляров сущностей. Некоторые связи бывают настолько сильными (конечно, в случае связи "один-ко-многим"), что при удалении опорного экземпляра сущности (соответствующего концу связи "один") нужно удалить и все экземпляры сущности, соответствующие концу связи "многие". Соответствующее требование "каскадного удаления" можно сформулировать при определении сущности.
- Домены. Как и в случае реляционной модели данных бывает полезна возможность определения потенциально допустимого множества значений атрибута сущности (домена).

Эти и другие более сложные элементы модели данных "Сущность-Связи" делают ее существенно более мощной, но одновременно несколько усложняют ее использование. Конечно, при реальном использовании ER-диаграмм для проектирования баз данных необходимо ознакомиться со всеми возможностями.

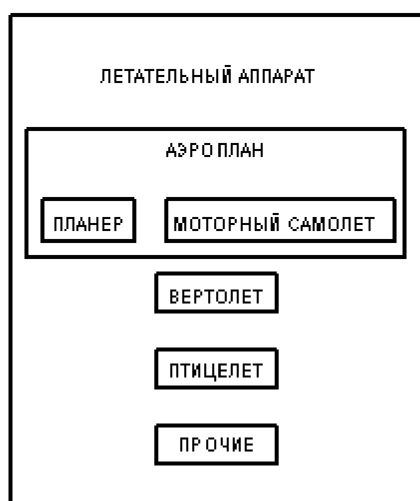
В данном материале мы немного подробнее разберем только один из упомянутых элементов - подтип сущности.

Сущность может быть расщеплена на два или более взаимно исключающих подтипа, каждый из которых включает общие атрибуты и/или связи. Эти

общие атрибуты и/или связи явно определяются один раз на более высоком уровне. В подтипах могут определяться собственные атрибуты и/или связи. В принципе подтипизация может продолжаться на более низких уровнях, но опыт показывает, что в большинстве случаев оказывается достаточно двух-трех уровней.

Сущность, на основе которой определяются подтипы, называется супертипом. Подтипы должны образовывать полное множество, т.е. любой экземпляр супертипа должен относиться к некоторому подтипу. Иногда для полноты приходится определять дополнительный подтип ПРОЧИЕ.

Пример: Супертип ЛЕТАТЕЛЬНЫЙ АППАРАТ



Как полагается это читать? От супертипа: ЛЕТАТЕЛЬНЫЙ АППАРАТ, который должен быть АЭРОПЛАНОМ, ВЕРТОЛЕТОМ, ПТИЦЕЛЕТОМ или ДРУГИМ ЛЕТАТЕЛЬНЫМ АППАРАТОМ. От подтипа: ВЕРТОЛЕТ, который относится к типу ЛЕТАТЕЛЬНОГО АППАРАТА. От подтипа, который является одновременно супертипа: АЭРОПЛАН, который относится к типу ЛЕТАТЕЛЬНОГО АППАРАТА и должен быть ПЛАНЕРОМ или МОТОРНЫМ САМОЛЕТОМ.

Иногда удобно иметь два или более разных разбиения сущности на подтипы. Например, сущность ЧЕЛОВЕК может быть разбита на подтипы по профессиональному признаку (ПРОГРАММИСТ, ДОЯРКА и т.д.), а может - по по-

ловому признаку (МУЖЧИНА, ЖЕНЩИНА).

5.2.5. Получение реляционной схемы из ER-схемы

Шаг 1. Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

Шаг 2. Каждый атрибут становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

Шаг 3. Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификатора, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

Шаг 4. Связи многие-к-одному (и один-к-одному) становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи "один", и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

Шаг 5. Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

Шаг 6. Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

- все подтипы в одной таблице (а)
- для каждого подтипа - отдельная таблица (б)

При применении способа (а) таблица создается для наиболее внешнего супертипа, а для подтипов могут создаваться представления. В таблицу добавляется по крайней мере один столбец, содержащий код ТИПА; он становится частью первичного ключа.

При использовании метода (б) для каждого подтипа первого уровня (для более нижних - представления) супертип воссоздается с помощью представления UNION (из всех таблиц подтипов выбираются общие столбцы - столбцы супертипа).

Все в одной таблице	**Таблица - на подтип**
Преимущества	
Все хранится вместе Легкий доступ к супертипу и подтипам Требуется меньше таблиц	Более ясны правила подтипов Программы работают только с нужными таблицами
Недостатки	
Слишком общее решение Требуется дополнительная логика работы с разными наборами столбцов и разными ограничениями Потенциальное узкое место (в связи с блокировками) Столбцы подтипов должны быть необязательными В некоторых СУБД для хранения неопределенных значений требуется дополнительная память	Слишком много таблиц Смущающие столбцы в представлении UNION Потенциальная потеря производительности при работе через UNION Над супертипом невозможны модификации

Шаг 7. Имеется два способа работы при наличии исключающих связей:

- общий домен (а)
- явные внешние ключи (б)

Если остающиеся внешние ключи все в одном домене, т.е. имеют общий формат (способ (а)), то создаются два столбца: идентификатор связи и идентификатор сущности. Столбец идентификатора связи используется для различения связей, покрываемых дугой исключения. Столбец идентификатора сущности используется для хранения значений уникального идентификатора сущности на дальнем конце соответствующей связи.

Если результирующие внешние ключи не относятся к одному домену, то для каждой связи, покрываемой дугой исключения, создаются явные столбцы внешних ключей; все эти столбцы могут содержать неопределенные значения.

Общий домен	**Явные внешние ключи**
Преимущества	
Нужно только два столбца	Условия соединения - явные
Недостатки	
Оба дополнительных атрибута должны использоваться в соединениях	Слишком много столбцов

6. Структурированный язык запросов SQL

Одним из основных преимуществ реляционного подхода к организации баз данных (БД) является то, что пользователи реляционных БД получают возможность эффективной работы в терминах простых и наглядных понятий таблиц, их строк и столбцов без потребности знания реальной организации данных во внешней памяти.

Реляционная модель данных, содержащая набор четких предписаний к базовой организации любой реляционной системы управления базами данных

(СУБД), позволяет пользователям работать в ненавигационной манере, т.е. для выборки информации из БД человек должен всего лишь указать список интересующих его таблиц и те условия, которым должны удовлетворять выбираемые данные. СУБД скрывает от пользователя выполняемые ей последовательные просмотры таблиц, выполняя их наиболее эффективным образом. Очень важная особенность реляционных систем состоит в том, что результатом выполнения любого запроса к таблицам БД является также таблица, которую можно сохранить в БД и/или по отношению к которой можно выполнять новые запросы.

Базовым требованием к реляционным СУБД является наличие мощного и в тоже время простого языка, позволяющего выполнять все необходимые пользователям операции. В последние годы таким повсеместно принятым языком стал язык реляционных БД SQL - Structured Query Language (теперь все чаще название языка понимается как Standard Query Language).

До появления SQL в СУБД (независимо от того, на какой модели они основывались) приходилось поддерживать по крайней мере три языка, которые обычно имели мало общего: язык определения данных (ЯОД), служащий для спецификации структур БД (обычно общую структуру БД называют схемой БД); язык манипулирования данными (ЯМД), позволяющий создавать прикладные программы, взаимодействующие с БД; и язык администрирования БД (ЯАДБ), с помощью которого можно было выполнять служебные действия (например, изменять структуру БД или производить ее настройку с целью повышения эффективности). Кроме того, если требовалось предоставить пользователям СУБД интерактивный доступ к БД, приходилось вводить еще один язык, операторы которого выполняются в диалоговом режиме. Язык SQL позволяет решать все эти задачи.

6.1. История языка баз данных SQL

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Ис-

ходное название языка SEQUEL (Structured English Query Language) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. В языке отсутствовали средства синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

В настоящее время SQL реализован практически во всех коммерческих реляционных СУБД, все фирмы провозглашают соответствие своей реализации стандарту SQL, и на самом деле реализованные диалекты SQL очень близки. Это произошло не сразу и не просто.

Наиболее близкими к System R являлись две системы фирмы IBM - SQL/DS и DB2. Как кажется (документальных подтверждений этому автор не имеет), обе эти системы прямо использовали реализацию System R. Отсюда предельная близость реализованных диалектов SQ к SQL System R. Из SQL System R были удалены только те части, которые были недостаточно проработаны (например, точки сохранения) или реализация которых вызывала слишком большие технические трудности (например, ограничения целостности и триггеры). Можно назвать этот путь к коммерческой реализации SQL движением сверху вниз.

Другой подход применялся в таких системах, как Oracle и Informix. Несмотря на различие в способе разработки этих систем, реализация SQL происходила "снизу вверх". В первых выпущенных на рынок реализациях SQL в этих системах использовалось минимальное и очень ограниченное подмножество

SQL System R. В частности, в первой реализации SQL в СУБД Oracle в операторах выборки не допускалось использование вложенных подзапросов.

Тем не менее, несмотря на эти ограничения и на очень слабую на первых порах эффективность, ориентация фирм на поддержание разных аппаратных платформ и заинтересованность пользователей в переходе к реляционным системам позволили фирмам добиться коммерческого успеха и приступить к совершенствованию своих реализаций. В текущих версиях Oracle и Informix поддерживаются достаточно мощные диалекты SQL, хотя реализация иногда вызывает сомнения.

Особенностью большинства современных коммерческих СУБД, затрудняющей анализ существующих диалектов SQL, является отсутствие полного описания языка. Обычно описание разбросано по разным руководствам и перемешано с описанием специфических для данной системы языковых средств, не имеющих отношения к SQL. Тем не менее можно сказать, что базовый набор операторов SQL, включающий операторы определения схемы БД, выборки и манипулирования данными, авторизации доступа к данным, поддержки встраивания SQL в языки программирования и операторы динамического SQL, в коммерческих реализациях относительно устоялся и более или менее соответствует стандарту.

6.2. Стандартизация SQL

Деятельность по стандартизации языка SQL началась практически одновременно с появлением первых его коммерческих реализаций. Первый из числа имеющихся у автора документ датирован октябрем 1985 г. и является уже очередным проектом стандарта ANSI/ISO.

Понятно, что в качестве стандарта нельзя было использовать SQL System R. Во-первых, этот вариант языка не был должным образом технически проработан. Во-вторых, его слишком сложно было бы реализовать (кто знает, как бы сложилась дальнейшая история SQL, если бы были полностью реализо-

ваны все идеи System R). С другой стороны, первые коммерческие реализации языка настолько различались, что ни один из реализованных диалектов не имел шансов быть принятым в качестве стандарта.

Анализ доступных документов показывает, что процесс происходил очень сложно с использованием далеко не только научных доводов. В результате, принятый в 1989 г. Международный стандарт SQL во многих частях имеет чрезвычайно общий характер и допускает очень широкое толкование. В этом стандарте полностью отсутствуют такие важные разделы, как манипулирование схемой БД и динамический SQL. Многие важные аспекты языка в соответствии со стандартом определяются в реализации.

Возможно, наиболее важными достижениями стандарта SQL являются четкая стандартизация синтаксиса и семантики операторов выборки и манипулирования данными и фиксация средств ограничения целостности БД, включающих возможности определения первичного и внешних ключей отношений и так называемых проверочных ограничений целостности. Средства определения внешних ключей позволяют легко формулировать требования так называемой целостности БД по ссылкам.

6.3. Современное состояние SQL

В этом разделе мы коротко рассмотрим основные особенности стандарта языка SQL 1989 года. Как было видно из приведенного краткого обзора наиболее продвинутых серверов баз данных, во всех них полностью реализован именно стандарт SQL-89. Поэтому при разработке достаточно сложных информационных систем необходимо хотя бы в общих чертах представлять основные свойства языка.

6.3.1. Типы данных

В языке SQL/89 поддерживаются следующие типы данных:

CHARACTER, NUMERIC, DECIMAL, INTEGER, SMALLINT, FLOAT, REAL, DOUBLE PRECISION. Эти типы данных классифицируются на типы строк символов, точных чисел и приблизительных чисел.

К первому классу относится CHARACTER. Спецификатор типа имеет вид CHARACTER (length), где length задает длину строк данного типа. Заметим, что в SQL/89 нет типа строк переменного размера, хотя во многих реализациях они допускаются. Литеральные строки символов изображаются в виде 'последовательность-символов' (например, 'example').

Представителями второго класса типов являются NUMERIC, DECIMAL (или DEC), INTEGER (или INT) и SMALLINT. Спецификатор типа NUMERIC имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представляемые с точностью precision и масштабом scale. Здесь и далее, если опущен масштаб, то он полагается равным 0, а если опущена точность, то ее значение по умолчанию определяется в реализации.

Спецификатор типа DECIMAL (или DEC) имеет вид NUMERIC [(precision [, scale]). Специфицируются точные числа, представленные с масштабом scale и точностью, равной или большей значения precision.

INTEGER специфицирует тип данных точных чисел с масштабом 0 и определяемой в реализации точностью. SMALLINT специфицирует тип данных точных чисел с масштабом 0 и определяемой в реализации точностью, не большей, чем точность чисел типа INTEGER.

Литеральные значения точных чисел в общем случае представляются в форме [+ -] <целое-без-знака> [.<целое-без-знака>].

Наконец, к классу типов данных приблизительных чисел относятся типы FLOAT, REAL и DOUBLE PRECISION. Спецификатор типа FLOAT имеет вид FLOAT [(precision)]. Специфицируются приблизительные числа с двоичной точностью, равной или большей значения precision.

REAL специфицирует тип данных приблизительных чисел с точностью, определенной в реализации. DOUBLE PRECISION специфицирует тип данных

приблизительных чисел с точностью, определенной в реализации, большей, чем точность типа REAL.

Литеральные значения приблизительных чисел в общем случае представляются в виде <литеральное-значение-точного-числа>E<целое-со-знаком>.

Заметим, что хотя с использованием языка SQL можно определить схему БД, содержащую данные любого из перечисленных типов, возможность использования этих данных в прикладных системах зависит от применяемого языка программирования. Весь набор типов данных можно использовать, только если программировать на ПЛ/1. Поэтому в некоторых реализациях SQL типы данных с масштабом и точностью вообще не поддерживаются.

Хотя правила встраивания SQL в программы на языке Си не определены в SQL/89, в большинстве реализаций, поддерживающих такое встраивание, имеется следующее соответствие между типами данных SQL и типами данных Си: CHARACTER соответствует строкам Си; INTEGER соответствует long; SMALLINT соответствует short; REAL соответствует float; DOUBLE PRECISION соответствует double (именно такое соответствие утверждено в стандарте SQL/92).

Заметим еще, что в большинстве реализаций SQL поддерживаются некоторые дополнительные типы данных, например, DATE, TIME, INTERVAL, MONEY. Некоторые из этих типов специфицированы в стандарте SQL/92, но в текущих реализациях синтаксические и семантические свойства таких типов могут различаться.

6.3.2. Средства определения схемы

Средства определения схемы БД в стандарте SQL/89 относятся к наиболее слабым и допускающим различную интерпретацию частям стандарта. Более того, неизвестна ни одна реализация, в которой поддерживался бы в точности такой набор средств определения схемы.

Поэтому, чтобы добиться мобильности прикладной системы в достаточно широком классе реализаций SQL/89, необходимо тщательно локализовать компоненты определения схемы БД. Видимо, лучше всего сосредоточить всю работу со схемой БД в одном модуле и иметь в виду, что при переходе к другой СУБД очень вероятно потребуется переделка этого модуля.

Особо отметим, что в SQL/89 вообще отсутствуют какие-либо средства изменения схемы БД: нет возможности удалить схему таблицы, добавить к схеме таблицы новый столбец и т.д. Во всех реализациях такие средства поддерживаются, но они могут различаться и синтаксисом, и семантикой.

Несмотря на отсутствие особых надежд на то, что удастся встретить реализацию, поддерживающую язык определения схем SQL/89, мы коротко опишем этот язык (без синтаксических деталей), чтобы оценить на содержательном уровне возможности SQL/89 в этой части и получить хотя бы какие-то средства сравнения разных реализаций.

6.3.3. Оператор определения схемы

В соответствии с правилами SQL/89 каждая таблица данной БД имеет простое и квалифицированное имена. В качестве квалификатора имени выступает "идентификатор полномочий" таблицы, который обычно в реализациях совпадает с именем некоторого пользователя, квалифицированное имя таблицы имеет вид:

```
<идентификатор полномочий>.<простое имя>
```

Подход к определению схемы в SQL/89 состоит в том, что все таблицы с одним идентификатором полномочий создаются (определяются) путем выполнения одного оператора определения схемы. При этом в стандарте не определяется способ выполнения оператора определения схемы: должен ли он выпол-

няться только в интерактивном режиме или может быть встроен в программу, написанную на традиционном языке программирования.

В операторе определения схемы содержится идентификатор полномочий и список элементов схемы, каждый из которых может быть определением таблицы, определением представления (view) или определением привилегий. Каждое из этих определений представляется отдельным оператором SQL/89, но все они, как уже говорилось, должны быть встроены в оператор определения схемы.

Для этих операторов мы приведем синтаксис, поскольку это позволит более четко описать их особенности.

Определение таблицы

Оператор определения таблицы имеет следующий синтаксис:

```
<table definition> ::=
 CREATE TABLE <table name> (<table element> [{,<table
 element>}...])
<table element> ::=
 <column definition>
 <table constraint definition>
```

Кроме имени таблицы, в операторе специфицируется список элементов таблицы, каждый из которых служит либо для определения столбца, либо для определения ограничения целостности определяемой таблицы. Требуется наличие хотя бы одного определения столбца. Оператор CREATE TABLE определяет так называемую базовую таблицу, то есть реальное хранилище данных.

Для определения столбцов таблицы и ограничений целостности используются специальные операторы, которые должны быть вложены в оператор определения таблицы.

Определение столбца

Оператор определения столбца описывается следующими синтаксическими правилами:

```
<column definition> ::=
 <column name> <data type>
 [<default clause>] [<column constraint>...]
<default clause> ::=
 DEFAULT { <literal> | USER | NULL }
<column constraint> ::=
 NOT | NULL [<unique specification>]
| <references specification>
| CHECK (<search condition>)
```

Как видно, кроме обязательной части, в которой определяется имя столбца и его тип данных, определение столбца может содержать два необязательных раздела: раздел значения столбца по умолчанию и раздел ограничений целостности столбца.

В разделе значения по умолчанию указывается значение, которое должно быть помещено в строку, заносимую в данную таблицу, если значение данного столбца явно не указано. Значение по умолчанию может быть указано в виде литеральной константы с типом, соответствующим типу столбца; путем задания ключевого слова USER, которому при выполнении оператора занесения строки соответствует символьная строка, содержащая имя текущего пользователя (в этом случае столбец должен иметь тип символьных строк); или путем задания ключевого слова NULL, означающего, что значением по умолчанию является неопределенное значение. Если значение столбца по умолчанию не специфицировано, и в разделе ограничений целостности столбца указано NOT NULL, то попытка занести в таблицу строку с неспецифицированным значени-

ем данного столбца приведет к ошибке.

Указание в разделе ограничений целостности NOT NULL приводит к неявному порождению проверочного ограничения целостности для всей таблицы (см. следующий подраздел) "CHECK (C IS NOT NULL)" (где C - имя данного столбца). Если ограничение NOT NULL не указано, и раздел умолчаний отсутствует, то неявно порождается раздел умолчаний DEFAULT NULL. Если указана спецификация уникальности, то порождается соответствующая спецификация уникальности для таблицы.

Если в разделе ограничений целостности указано ограничение по ссылкам данного столбца (<reference specification>), то порождается соответствующее определение ограничения по ссылкам для таблицы:

```
FOREIGN KEY(C) <reference specification>.
```

Наконец, если указано проверочное ограничение столбца, то условие поиска этого ограничения должно ссылаться только на данный столбец, и неявно порождается соответствующее проверочное ограничение для всей таблицы.

Определение ограничений целостности таблицы

Ограничения целостности таблицы обладают следующим синтаксисом:

```
<table constraint definition> ::=
 <unique constraint definition>
 <referential constraint definition>
 <check constraint definition>
<unique constraint definition> ::=
 <unique specification> (<unique column list>)
<unique specification> ::= UNIQUE PRIMARY KEY
<unique column list> ::= <column name> [{,<column
```

```
name>}...]
<referential constraint definition> ::=
 FOREIGN KEY (<referencing columns>) <references
specification>
<references specification> ::=
 REFERENCES <referenced table and columns>
<referencing columns> ::= <reference column list>
<referenced table and columns> ::=
 <table name> [(<reference column list>)]
<reference column list> ::= <column name> [{,<column
name>}...]
<check constraint definition> ::= CHECK (<search
condition>)
```

Для одной таблицы может быть задано несколько ограничений целостности, в том числе те, которые неявно порождаются ограничениями целостности столбцов. Стандарт SQL/89 устанавливает, что ограничения таблицы фактически проверяются при выполнении каждого оператора SQL.

Замечание: Наличие правильно подобранного набора ограничений БД очень важно для надежного функционирования прикладной информационной системы. Вместе с тем, в некоторых СУБД ограничения целостности практически не поддерживаются. Поэтому при проектировании прикладной системы необходимо принять решение о том, что более существенно: рассчитывать на поддержку ограничений целостности, но ограничить набор возможных СУБД, или отказаться от их использования на уровне SQL, сохранив возможность использования не самых современных СУБД.

Далее T обозначает таблицу, для которой определяются ограничения целостности.

Ограничение уникальности

Каждое имя столбца в списке уникальности должно именовать столбец T и не должно входить в этот список более одного раза. При определении столбца, входящего в список уникальности, должно быть указано ограничение столбца NO NULL. Среди ограничений уникальности T не должно быть более одного определения первичного ключа (ограничения уникальности с ключевым словом RIMARY KEY).

Действие ограничения уникальности состоит в том, что в таблице T не допускается появление двух или более строк, значения столбцов уникальности которых совпадают.

Ограничение по ссылкам

Ограничение по ссылкам от заданного набора столбцов CT таблицы T на заданный набор столбцов CT1 некоторой определенной к этому моменту таблицы T1 определяет условие на содержимое обеих этих таблиц, при котором ссылки можно считать корректными.

Если список столбцов CT1 явно специфицирован в определении ограничения по ссылкам, то требуется, чтобы этот список явно входил в какое-либо определение уникальности таблицы T1. Если же список CT1 не специфицирован явно в определении ограничения по ссылкам таблицы T, то требуется, чтобы в определении таблицы T1 присутствовало определение первичного ключа, и список CT1 неявно полагается совпадающим со списком имен столбцов из определения первичного ключа таблицы T1. Имена столбцов списков CT и CT1 должны именовать столбцы таблиц T и T1 соответственно и не должны появляться в списках более одного раза. Списки столбцов CT и CT1 должны содержать одинаковое число элементов, и столбец таблицы T, идентифицируемый i-ым элементом списка CT должен иметь тот же тип, что столбец таблицы T1, идентифицируемый i-ым элементом списка CT1.

По определению, таблицы T и T1 удовлетворяют заданному ограничению

по ссылкам, если для каждой строки s таблицы T такой, что все значения столбцов s, идентифицируемых списком CT, не являются неопределенными, существует строка s1 таблицы T1 такая, что значения столбцов s1, идентифицируемых списком CT1, позиционно равны значениям столбцов s, идентифицируемых списком CT. По человечески это можно сформулировать так: ограничение по ссылкам удовлетворяется, если для каждой корректной ссылки существует объект, на который она ссылается. В привычной программистам терминологии, ограничение по ссылкам не позволяет производить "висячие" ссылки, не ведущие ни к какому объекту.

Проверочное ограничение

Проверочное ограничение специфицирует условие, которому должна удовлетворять в отдельности каждая строка таблицы T. Это условие не должно содержать подзапросов, спецификаций агрегатных функций, а также ссылок на внешние переменные или параметры. В него могут входить только имена столбцов данной таблицы и литеральные константы.

Таблица удовлетворяет проверочному ограничению целостности в том и только в том случае, когда вычисление условия для каждой строки таблицы дает true.

Замечание: В некоторых реализациях допускаются расширенные механизмы ограничений по ссылкам и проверочных ограничений. Следует быть внимательным, если не желательно выходить за пределы возможностей стандарта.

6.3.4. Средства манипулирования данными

Структура запросов

Для того, чтобы можно было более или менее точно рассказать про структуру запросов в стандарте SQL/89, необходимо начать со сводки синтаксиче-

ских правил:

```
<cursor specification> ::=
 <query expression> [<order by clause>]
<query expression> ::=
 <query term>
 <query expression> UNION [ALL] <query term>
<query term> ::=
 <query specification>
 (<query expression>)
<query specification> ::=
 (SELECT [ALL DISTINCT] <select list> <table expression>)
<select statement> ::=
 SELECT [ALL DISTINCT] <select list> INTO <select target
 list> <table expression>
<subquery> ::=
 (SELECT [ALL DISTINCT] <result specification>
 <table expression>
<table expression> ::=
 <from clause>
 [<where clause>] [<group by clause>] [<having clause>]
```

Язык допускает три типа синтаксических конструкций, начинающихся с ключевого слова SELECT: спецификация курсора (cursor specification), оператор выборки (select statement) и подзапрос (subquery). Основой всех них является синтаксическая конструкция "табличное выражение (table expression)". Семантика табличного выражения состоит в том, что на основе последовательного применения разделов from, where, group by и having из заданных в разделе from таблиц строится некоторая новая результирующая таблица, порядок следования

строк которой неопределен и среди строк которой могут находиться дубликаты (т.е. в общем случае таблица-результат табличного выражения является мультимножеством строк). На самом деле именно структура табличного выражения наибольшим образом характеризует структуру запросов языка SQL/89. Мы рассмотрим структуру и смысл разделов табличного выражения ниже, но до этого немного подробнее обсудим три упомянутые конструкции, включающие табличные выражения.

Спецификация курсора

Наиболее общей является конструкция "спецификация курсора". Курсор - это понятие языка SQL, позволяющее с помощью набора специальных операторов получить построчный доступ к результату запроса к БД. К табличным выражениям, участвующим в спецификации курсора, не предъявляются какие-либо ограничения. Как видно из сводки синтаксических правил, при определении спецификации курсора используются три дополнительных конструкции: спецификация запроса, выражение запросов и раздел ORDER BY.

Спецификация запроса

В спецификации запроса задается список выборки (список арифметических выражений над значениями столбцов результата табличного выражения и констант). В результате применения списка выборки к результату табличного выражения производится построение новой таблицы, содержащей то же число строк, но вообще говоря другое число столбцов, включающих результаты вычисления соответствующих арифметических выражений из списка выборки. Кроме того, в спецификации запроса могут содержаться ключевые слова ALL или DISTINCT. При наличии ключевого слова DISTINCT из таблицы, полученной применением списка выборки к результату табличного выражения, удаляются строки-дубликаты; при указании ALL (или просто при отсутствии DISTINCT) удаление строк-дубликатов не производится.

Выражение запросов

Выражение запросов - это выражение, строящееся по указанным синтаксическим правилам на основе спецификаций запросов. Единственной операцией, которую разрешается использовать в выражениях запросов, является операция UNION (объединение таблиц) с возможной разновидностью UNION ALL. К таблицам-операндам выражения запросов предъявляется то требование, что все они должны содержать одно и то же число столбцов, и соответствующие столбцы всех операндов должны быть одного и того же типа. Выражение запросов вычисляется слева направо с учетом скобок. При выполнении операции UNION производится обычное теоретико-множественное объединение операндов, т.е. из результирующей таблицы удаляются дубликаты. При выполнении операции UNION ALL образуется результирующая таблица, в которой могут содержаться строки-дубликаты.

Раздел ORDER BY

Наконец, раздел ORDER BY позволяет установить желаемый порядок просмотра результата выражения запросов. Синтаксис ORDER BY следующий:

```
<order by clause> ::=
 ORDER BY <sort specification> [{,<sort
specification>}...]
<sort specification> ::=
 {<unsigned integer> <column specification>} [ASC DESC]
```

Как видно из этих синтаксических правил, фактически задается список столбцов результата выражения запросов, и для каждого столбца указывается порядок просмотра строк результата в зависимости от значений этого столбца (ASC - по возрастанию (умолчание) DESC - по убыванию). Столбцы можно задавать их именами в том и только в том случае, когда (1) выражение запросов

не содержит операций UNION или UNION ALL и (2) в списке выборки спецификации запроса этому столбцу соответствует арифметическое выражение, состоящее только из имени столбца. Во всех остальных случаях в разделе ORDER BY должен указываться порядковый номер столбца в таблице-результате выражения запросов.

Оператор выборки

Оператор выборки - это отдельный оператор языка SQL/89, позволяющий получить результат запроса в прикладной программе без привлечения курсора. Поэтому оператор выборки имеет синтаксис, отличающийся от синтаксиса спецификации курсора, и при его выполнении возникают ограничения на результат табличного выражения. Фактически, и то, и другое диктуется спецификой оператора выборки как одиночного оператора SQL: при его выполнении результат должен быть помещен в переменные прикладной программы. Поэтому в операторе появляется раздел INTO, содержащий список переменных прикладной программы, и возникает то ограничение, что результирующая таблица должна содержать не более одной строки. Соответственно, результат базового табличного выражения должен содержать не более одной строки, если оператор выборки не содержит спецификации DISTINCT, и таблица, полученная применением списка выборки к результату табличного выражения, не должна содержать более одной несовпадающих строк, если спецификация DISTINCT задана.

Замечание: В диалекте SQL СУБД Oracle поддерживается расширенный вариант оператора выборки, результатом которого не обязательно является таблица из одной строки. Такое расширение не поддерживается ни в SQL/89, ни в SQL/92.

Подзапрос

Наконец, последняя конструкция SQL/89, которая может содержать та-

бличные выражения, - это подзапрос, то есть запрос, который может входить в предикат условия выборки оператора SQL. В SQL/89 к подзапросам применяется то ограничение, что результирующая таблица должна содержать в точности один столбец. Поэтому в синтаксических правилах, определяющих подзапрос, вместо списка выборки указано "выражение, вычисляющее значение", т.е. арифметическое выражение. Заметим еще, что поскольку подзапрос всегда вложен в некоторый другой оператор SQL, то в качестве констант в арифметическом выражении выборки и логических выражениях разделов WHERE и HAVING можно использовать значения столбцов текущих строк таблиц, участвующих в (под)запросах более внешнего уровня. Более подробно об этом см. ниже, при описании семантики табличных выражений.

Табличное выражение

Стандарт SQL/89 рекомендует рассматривать вычисление табличного выражения как последовательное применение разделов FROM, WHERE, GROUP BY и HAVING к таблицам, заданным в списке FROM. Раздел FROM имеет следующий синтаксис:

```
<from clause> ::=
 FROM <table reference> ({,<table reference>}...]
<table reference> ::=
 <table name> [<correlation name>]
```

Раздел FROM

Результатом выполнения раздела FROM является расширенное декартово произведение таблиц, заданных списком таблиц раздела FROM. Расширенное декартово произведение (расширенное, потому что в качестве операндов и результата допускаются мультимножества) в стандарте определяется следующим образом:

"Расширенное произведение R есть мультимножество всех строк r таких, что r является конкатенацией строк из всех идентифицированных таблиц в том порядке, в котором они идентифицированы. Мощность R есть произведение мощностей идентифицированных таблиц. Порядковый номер столбца в R есть n+s, где n - порядковый номер порождающего столбца в именованной таблице T, а s - сумма степеней всех таблиц, идентифицированных до T в разделе FROM."

Как видно из синтаксиса, рядом с именем таблицы можно указывать еще одно имя "correlation name". Фактически, это некоторый синоним имени таблицы, который можно использовать в других разделах табличного выражения для ссылки на строки именно этого вхождения таблицы.

Если табличное выражение содержит только раздел FROM (это единственный обязательный раздел табличного выражения), то результат табличного выражения совпадает с результатом раздела FROM.

Раздел WHERE

Если в табличном выражении присутствует раздел WHERE, то следующим вычисляется он. Синтаксис раздела WHERE следующий:

```
<where clause> ::= WHERE <search condition>
<search condition> ::=
 <boolean term>
 ( <search condition> OR <boolean term>
<Boolean term> ::=
 <boolean factor>
 ( <boolean term> AND <boolean factor>
<boolean factor> ::= [NOT] <boolean primary>
<boolean primary> ::= <predicate> (<search condition>)
```

Вычисление раздела WHERE производится по следующим правилам:

Пусть R - результат вычисления раздела FROM. Тогда условие поиска применяется ко всем строкам R, и результатом раздела WHERE является таблица, состоящая из тех строк R, для которого результатом вычисления условия поиска является true. Если условие выборки включает подзапросы, то каждый подзапрос вычисляется для каждого кортежа таблицы R (в стандарте используется термин "effectively" в том смысле, что результат должен быть таким, как если бы каждый подзапрос действительно вычислялся заново для каждого кортежа R).

Заметим, что поскольку SQL/89 допускает наличие в базе данных неопределенных значений, то вычисление условия поиска производится не в булевой, а в трехзначной логике со значениями true, false и unknown (неизвестно). Для любого предиката определено, в каких ситуациях он может порождать значение unknown. Булевские операции AND, OR и NOT работают в трехзначной логике следующим образом:

```
true AND unknown = unknown
unknown AND true = unknown
unknown AND unknown = unknown
true OR unknown = true
unknown OR true = true
unknown OR unknown = unknown
NOT unknown = unknown
```

Среди предикатов условия поиска в соответствии с SQL/89 могут находиться следующие предикаты: предикат сравнения, предикат between, предикат in, предикат like, предикат null, предикат с квантором и предикат exists. Сразу заметим, что во всех реализациях SQL на эффективность выполнения запроса существенно влияет наличие в условии поиска простых предикатов сравнения

(предикатов, задающих сравнение столбца таблицы с константой). Наличие таких предикатов позволяет СУБД использовать индексы при выполнении запроса, т.е. избегать полного просмотра таблицы. Хотя в принципе язык SQL позволяет пользователям не заботиться о конкретном наборе предикатов в условии выборки (лишь бы они были синтаксически и семантически правильны), при реальном использовании SQL-ориентированных СУБД такие технические детали стоит иметь в виду.

Предикат сравнения

Синтаксис предиката сравнения определяется следующими правилами:

```
<comparison predicate> ::=
 <value expression> <comp op>
 {<value expression> <subquery>}
<comp op> ::=
 = <> < > <= >=
```

Через "<>" обозначается операция "неравенства". Арифметические выражения левой и правой частей предиката сравнения строятся по общим правилам построения арифметических выражений и могут включать в общем случае имена столбцов таблиц из раздела FROM и константы. Типы данных арифметических выражений должны быть сравнимыми (например, если тип столбца a таблицы A является типом символьных строк, то предикат "a = 5" недопустим).

Если правый операнд операции сравнения задается подзапросом, то дополнительным ограничением является то, что мощность результата подзапроса должна быть не более единицы. Если хотя бы один из операндов операции сравнения имеет неопределенное значение, или если правый операнд является подзапросом с пустым результатом, то значение предиката сравнения равно unknown.

Заметим, что значение арифметического выражения не определено, если в его вычислении участвует хотя бы одно неопределенное значение. Еще одно важное замечание из стандарта SQL/89: в контексте GROUP BY, DISTINCT и ORDER BY неопределенное значение выступает как специальный вид определенного значения, т.е. возможно, например, образование группы строк, значение указанного столбца которых является неопределенным. Для обеспечения переносимости прикладных программ нужно внимательно оценивать специфику работы с неопределенными значениями в конкретной СУБД.

Предикат between

Предикат between имеет следующий синтаксис:

```
<between predicate> ::=
 <value expression>
 [NOT] BETWEEN <value expression> AND <value expression>
```

Результат "x BETWEEN y AND z" тот же самый, что результат "x >= y AND x <= z". Результат "x NOT BETWEEN y AND z" тот же самый, что результат "NOT (x BETWEEN y AND z)".

Предикат in

Предикат in определяется следующими синтаксическими правилами:

```
<in predicate> ::=
 <value expression> [NOT] IN
 {<subquery> (<in value list>)}
<in value list> ::=
 <value specification>
 {,<value specification>}...
```

Типы левого операнда и значений из списка правого операнда (напомним, что результирующая таблица подзапроса должна содержать ровно один столбец) должны быть сравнимыми.

Значение предиката равно true в том и только в том случае, когда значение левого операнда совпадает хотя бы с одним значением списка правого операнда. Если список правого операнда пуст (так может быть, если правый операнд задается подзапросом), или значение "подразумеваемого" предиката сравнения x = y (где x - значение арифметического выражения левого операнда) равно false для каждого элемента y списка правого операнда, то значение предиката in равно false. В противном случае значение предиката in равно unknown. По определению значение предиката "x NOT IN S" равно значению предиката "NOT (x IN S)".

Предикат like

Предикат like имеет следующий синтаксис:

```
<like predicate> ::=
 <column specification> [NOT] LIKE <pattern>
 [ESCAPE <escape character>]
<pattern> ::= <value specification>
 <escape character> ::= <value specification>
```

Типы данных столбца левого операнда и образца должны быть типами символьных строк. В разделе ESCAPE должен специфицироваться одиночный символ.

Значение предиката равно true, если pattern является подстрокой заданного столбца. При этом, если раздел ESCAPE отсутствует, то при сопоставлении шаблона со строкой производится специальная интерпретация двух символов шаблона: символ подчеркивания ("_") обозначает любой одиночный символ;

символ процента ("%") обозначает последовательность произвольных символов произвольной длины (может быть, нулевой).

Если же раздел ESCAPE присутствует и специфицирует некоторый одиночный символ x, то пары символов "x_" и "x%" представляют одиночные символы "_" и "%" соответственно.

Значение предиката like есть unknown, если значение столбца, либо шаблона неопределено.

Значение предиката "x NOT LIKE y ESCAPE z" совпадает со значением "NOT x LIKE y ESCAPE z".

Предикат null

Предикат null описывается синтаксическим правилом:

```
<null predicate> ::=
 <column specification> IS [NOT] NULL
```

Этот предикат всегда принимает значения true или false. При этом значение "x IS NULL" равно true тогда и только тогда, когда значение x неопределено. Значение предиката "x NOT IS NULL" равно значению "NOT x IS NULL".

Предикат с квантором

Предикат с квантором имеет следующий синтаксис:

```
<quantified predicate> ::=
 <value expression> <comp op> <quantifier> <subquery>
<quantifier> ::=
 <all> <some>
<all> ::= ALL
<some> ::= SOME ANY
```

Обозначим через x результат вычисления арифметического выражения левой части предиката, а через S результат вычисления подзапроса.

Предикат "x <comp op> ALL S" имеет значение true, если S пусто или значение предиката "x <comp op> s" равно true для каждого s, входящего в S. Предикат "x <comp op> ALL S" имеет значение false, если значение предиката "x <comp op> s" равно false хотя бы для одного s, входящего в S. В остальных случаях значение предиката "x <comp op> ALL S" равно unknown.

Предикат "x <comp op> SOME S" имеет значение false, если S пусто или значение предиката "x <comp op> s" равно false для каждого s, входящего в S. Предикат "x <comp op> SOME S" имеет значение true, если значение предиката "x <comp op> s" равно true хотя бы для одного s, входящего в S. В остальных случаях значение предиката "x <comp op> SOME S" равно unknown.

Предикат exists

Предикат exists имеет следующий синтаксис:

```
<exists predicate> ::=
EXISTS <subquery>
```

Значением этого предиката всегда является true или false, и это значение равно true тогда и только тогда, когда результат вычисления подзапроса не пуст.

Раздел GROUP BY

Если в табличном выражении присутствует раздел GROUP BY, то следующим выполняется он. Синтаксис раздела GROUP BY следующий:

```
<group by clause> ::=
 GROUP BY <column specification> [{,<column
```

```
specification>}...]
```

Если обозначить через R таблицу, являющуюся результатом предыдущего раздела (FROM или WHERE), то результатом раздела GROUP BY является разбиение R на множество групп строк, состоящего из минимального числа групп таких, что для каждого столбца из списка столбцов раздела GROUP BY во всех строках каждой группы, включающей более одной строки, значения этого столбца равны. Для обозначения результата раздела GROUP BY в стандарте используется термин "сгруппированная таблица".

Раздел HAVING

Наконец, последним при вычислении табличного выражения используется раздел HAVING (если он присутствует). Синтаксис этого раздела следующий:

```
<having clause> ::=
 HAVING <search condition>
```

Раздел HAVING может осмысленно появиться в табличном выражении только в том случае, когда в нем присутствует раздел GROUP BY. Условие поиска этого раздела задает условие на группу строк сгруппированной таблицы. Формально раздел HAVING может присутствовать и в табличном выражении, не содержащем GROUP BY. В этом случае полагается, что результат вычисления предыдущих разделов представляет собой сгруппированную таблицу, состоящую из одной группы без выделенных столбцов группирования.

Условие поиска раздела HAVING строится по тем же синтаксическим правилам, что и условие поиска раздела WHERE, и может включать те же самые предикаты. Однако имеются специальные синтаксические ограничения по части использования в условии поиска спецификаций столбцов таблиц из раз-

дела FROM данного табличного выражения. Эти ограничения следуют из того, что условие поиска раздела HAVING задает условие на целую группу, а не на индивидуальные строки.

Поэтому в арифметических выражениях предикатов, входящих в условие выборки раздела HAVING, прямо можно использовать только спецификации столбцов, указанных в качестве столбцов группирования в разделе GROUP BY. Остальные столбцы можно специфицировать только внутри спецификаций агрегатных функций COUNT, SUM, AVG, MIN и MAX, вычисляющих в данном случае некоторое агрегатное значение для всей группы строк. Аналогично обстоит дело с подзапросами, входящими в предикаты условия выборки раздела HAVING: если в подзапросе используется характеристика текущей группы, то она может задаваться только путем ссылки на столбцы группирования.

Результатом выполнения раздела HAVING является сгруппированная таблица, содержащая только те группы строк, для которых результат вычисления условия поиска есть true. В частности, если раздел HAVING присутствует в табличном выражении, не содержащем GROUP BY, то результатом его выполнения будет либо пустая таблица, либо результат выполнения предыдущих разделов табличного выражения, рассматриваемый как одна группа без столбцов группирования.

Агрегатные функции и результаты запросов

Агрегатные функции (в стандарте SQL/89 они называются функциями над множествами) определяются в SQL/89 следующими синтаксическими правилами:

```
<set function specification> ::=
 COUNT(*) <distinct set function>
 <all set function>
<distinct set function> ::=
```

```
 { AVG MAX MIN SUM COUNT } (DISTNICT <column
specification>)
<all set function> ::=
 { AVG MAX MIN SUM } ([ALL] <value expression>)
```

Как видно из этих правил, в стандарте SQL/89 определены пять стандартных агрегатных функций: COUNT - число строк или значений, MAX - максимальное значение, MIN - минимальное значение, SUM - суммарное значение и AVG - среднее значение.

Семантика агрегатных функций

Агрегатные функции предназначены для того, чтобы вычислять некоторое значение для заданного множества строк. Таким множеством строк может быть группа строк, если агрегатная функция применяется к сгруппированной таблице, или вся таблица. Для всех агрегатных функций, кроме COUNT(*), фактический (то есть требуемый семантикой) порядок вычислений следующий: на основании параметров агрегатной функции из заданного множества строк производится список значений. Затем по этому списку значений производится вычисление функции. Если список оказался пустым, то значение функции COUNT для него есть 0, а значение всех остальных функций - null.

Пусть T обозначает тип значений из этого списка. Тогда результат вычисления функции COUNT - точное число с масштабом и точностью, определяемыми в реализации. Тип результата значений функций MAX и MIN совпадает с T. При вычислении функций SUM и AVG тип T не должен быть типом символьных строк, а тип результата функции - это тип точных чисел с определяемыми в реализации масштабом и точностью, если T - тип точных чисел, и тип приблизительных чисел с определяемой в реализации точностью, если T - тип приблизительных чисел.

Вычисление функции COUNT(*) производится путем подсчета числа

строк в заданном множестве. Все строки считаются различными, даже если они состоят из одного столбца со значением null во всех строках.

Если агрегатная функция специфицирована с ключевым словом DISTINCT, то список значений строится из значений указанного столбца. (Подчеркнем, что в этом случае не допускается вычисление арифметических выражений!) Далее из этого списка удаляются неопределенные значения, и в нем устраняются значения-дубликаты. Затем вычисляется указанная функция.

Если агрегатная функция специфицирована без ключевого слова DISTINCT (или с ключевым словом ALL), то список значений формируется из значений арифметического выражения, вычисляемого для каждой строки заданного множества. Далее из списка удаляются неопределенные значения и производится вычисление агрегатной функции. Обратите внимание, что в этом случае не допускается применение функции COUNT!

Замечание: оба ограничения, указанные в двух предыдущих абзацах, являются более техническими, чем принципиальными, и могут отсутствовать в конкретных реализациях. Тем не менее, это ограничения стандарта SQL/89, и их нужно придерживаться при мобильном программировании.

Результаты запросов

Агрегатные функции можно разумно использовать в спецификации курсора, операторе выборки и подзапросе после ключевого слова SELECT (будем называть в этом подразделе все такие конструкции списком выборки, не забывая о том, что в случае подзапроса этот список состоит только из одного элемента), и в условии выборки раздела HAVING. Стандарт допускает более экзотические случаи использования агрегатных функций в подзапросах (агрегатная функция на группе кортежей внешнего запроса), но на практике они встречаются очень редко.

Рассмотрим различные случаи применения агрегатных функций в списке выборки в зависимости от вида табличного выражения.

Если результат табличного выражения R не является сгруппированной таблицей, то появление хотя бы одной агрегатной функции от множества строк R в списке выборки приводит к тому, что R неявно рассматривается как сгруппированная таблица, состоящая из одной (или нуля) групп с отсутствующими столбцами группирования. Поэтому в этом случае в списке выборки не допускается прямое использование спецификаций строк R: все они должны находиться внутри спецификаций агрегатных функций. Результатом запроса является таблица, состоящая не более чем из одной строки, полученной путем применения агрегатных функций к R.

Аналогично обстоит дело в том случае, когда R представляет собой сгруппированную таблицу, но табличное выражение не содержит раздела GROUP BY (и, следовательно, не содержит раздел HAVING). Если в случае предыдущего абзаца было два варианта формирования списка выборки: только с прямым указанием столбцов R или только с указанием их внутри спецификаций агрегатных функций, то в данном случае возможен только второй вариант. Результат табличного выражения явно объявлен сгруппированной таблицей, состоящей из одной группы, и результат запроса можно формировать только путем применения агрегатных функций к этой группе строк. Опять результатом запроса является таблица, состоящая не более чем из одной строки, полученной путем применения агрегатных функций к R.

Наконец, рассмотрим случай, когда R представляет собой "настоящую" сгруппированную таблицу, т.е. табличное выражение содержит раздел GROUP BY и, следовательно, определен, по крайней мере, один столбец группирования. В этом случае правила формирования списка выборки полностью соответствуют правилам формирования условия выборки раздела HAVING: допускается прямое использование спецификации столбцов группирования, а спецификации остальных столбцов R могут появляться только внутри спецификаций агрегатных функций. Результатом запроса является таблица, число строк в которой равно числу групп в R, и каждая строка формируется на основе значений столб-

цов группирования и агрегатных функций для данной группы.

Список литературы

1. Дейт К. Введение в системы баз данных. Наука. Москва. 1980.
2. Кузнецов С. Основы современных баз данных. Учебное пособие. Центр Информационных технологий.