Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра систем управления

Д.А. Ганьшин, М.А. Антипова

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ

Конспект лекций

для студентов специальностей
I-53 01 03 «Автоматическое управление в технических системах»
и I-53 01 07 «Информационные технологии
и управление в технических системах»
всех форм обучения

Минск 2006

УДК 681.518 (076)
ББК 32.965 я 7
Г 19

Р е ц е н з е н т :
профессор кафедры электротехники Военной академии Республики Беларусь,
канд. техн. наук В.Е. Гурский

Ганьшин Д.А.

Г 19 Информационное обеспечение систем управления: Конспект лекций для студ. спец. I-53 01 03 «Автоматическое управление в технических системах» и I-53 01 07 «Информационные технологии и управление в технических системах» всех форм обуч. / Д.А. Ганьшин, М.А. Антипова.– Мн.: БГУИР, 2006. – 60 с.

ISBN 985–444–869–X

Конспект лекций включает следующие разделы: предисловие, введение, дореляционная организация БД, основные понятия и функции СУБД, внутренняя организация СУБД, реляционный подход к организации БД, описание языка программирования БД – SQL, вопросы организации доступа прикладных программ к серверу БД, основы проектирования реляционных БД классическим методом нормализации отношений и более современным методом семантического моделирования.

Материал соответствует рабочей программе курса и излагается в соответствии с предоставленным количеством часов на изучение каждого раздела.

УДК 681.518 (076)
ББК 32.965 я 7

ISBN 985–444–869–X

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Предметом изучения курса «Информационное обеспечение систем управления» являются системы управления базами данных (СУБД).

В настоящее время программное обеспечение систем автоматизации, в частности, и систем управления в целом, в большинстве случаев разрабатывается на базе или с использованием современных СУБД. Эта тенденция связана с потребностью в обработке и хранении больших объемов различных видов информации. Наибольшее практическое применение имеет создание СУБД для научной, отраслевой, экономической, технической, производственной информации.

Цель изучения данного курса – получение и систематизация знаний об основных идеях и методах создания современных реляционных систем управления базами данных. В курсе лекций не рассматривается какая-либо одна популярная СУБД, а представленный материал в равной степени относится к любой современной системе. Как показывает опыт, без знания основ проектирования баз данных трудно на профессиональном уровне работать с конкретными системами, как бы хорошо они не были документированы.

Вводная часть курса посвящена основным различиям между файловыми системами и системами управления базами данных. На основе анализа возможностей современных файловых систем выделяются области применения, в которых достаточно использовать только файлы, а также те области, для которых необходимы базы данных.

В первой части рассматриваются основные характеристики ранних дореляционных систем, вводятся некоторые ключевые для баз данных понятия.

Во второй части курса обсуждаются базовые функции системы управления базами данных и приводится ее типовая организация, а также затрагиваются вопросы организации современных СУБД.

Третья часть курса содержит информацию о реляционном подходе к проектированию БД. Вводятся основные понятия теории создания реляционных моделей данных, рассматриваются свойства отношений и два базовых механизма манипулирования данными: реляционная алгебра и реляционное исчисление.

Четвертая часть курса посвящается внутренней организации современных многопользовательских реляционных СУБД. Рассматриваются методы организации внешней памяти баз данных и применяемые структуры данных. Вводится понятие транзакции и анализируются известные способы управления асинхронно выполняемыми транзакциями. Обсуждаются потребности в журнализации изменений баз данных и связь алгоритмов журнализации с политикой управления буферами оперативной памяти. Наконец, рассматриваются способы применения журнальной и архивной информации для восстановления баз данных после различных сбоев.

В пятой части в общих чертах рассматривается язык запросов реляционных баз данных – SQL (Structured Query Language). Обсуждаются способы использования SQL при программировании прикладных систем и приводятся основные операторы для создания схем баз данных и манипулирования непосредственно самими данными.

В шестом разделе частично рассмотрены вопросы практического программирования, освещены проблемы использования специализированных библиотек и встраиваемого SQL, описаны некоторые открытые интерфейсы, созданные для разработчиков на платформе MS Windows.

В седьмом разделе затрагиваются вопросы проектирования реляционных БД, в ней излагаются основные принципы нормализации, на которых основан классический метод разработки БД, а также описывается и более современный подход к проектированию реляционных баз данных, основанный на использовании семантических моделей.

ВВЕДЕНИЕ

С начала развития вычислительной техники образовались два основных направления ее применения.

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, установлению обратной связи с разработчиками новых архитектур ЭВМ.

Второе направление (которое непосредственно касается темы 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.

Второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями в области памяти. Понятно, что можно говорить о надежном и долговременном хранении информации только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная память этим свойством обычно не обладает. Вначале 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 данным, но были ограниченного размера.

Указанные ограничения не были очень существенными для 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 явилось неудовлетворительным фактором. Достаточно представить себе покупателя билета, который, стоя у кассы, должен дождаться полной перемотки магнитной ленты. Также одним из естественных требований к информационным системам является средняя скорость выполнения операций.

Вероятно, именно требования к вычислительной технике со стороны нечисленных приложений вызвали появление съемных магнитных дисков с подвижными головками, что явилось революцией в истории развития 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). Однако 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. Конечно, недостатки очевидны, но первый шаг был сделан и именно он послужил толчком к дальнейшему развитию автоматизированных систем управления информацией, разработке теоретического материала и прикладных программных продуктов в данной области.

1 ДОРЕЛЯЦИОННАЯ ОРГАНИЗАЦИЯ БД

Как уже было сказано, первым шагом на пути развития методов организации баз данных стали файловые системы. Основным объектом любой файловой системы является файл. 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.

Во всех файловых системах явно или неявно выделяется некоторый базовый уровень, обеспечивающий работу с файлами, представляющими набор прямо адресуемых в адресном пространстве файла блоков. В некоторых файловых системах базовый уровень доступен пользователю, но более часто прикрывается некоторым более высоким уровнем, стандартным для пользователей (набор стандартных библиотечных функций).

Файловые системы обычно обеспечивают хранение слабо структурированной информации, оставляя дальнейшую структуризацию прикладным программам. В большинстве 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.

Однако ситуация существенно отличается для ***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 индивидуальные дополнительные средства управления данными являлись существенной частью информационных систем и практически повторялись от одной системы к другой. Очень скоро стало понятно, что невозможно обойтись общей библиотекой программ, реализующей над стандартной базовой файловой системой более сложные методы хранения данных. Стремление выделить и обобщить общую часть информационных систем, ответственную за управление сложно структурированными данными, возможно, и явилось, первой побудительной причиной создания ***систем управления базами данных*** (СУБД).

Покажем это на примере. Допустим, что есть необходимость реализовать простую информационную систему, поддерживающую учет сотрудников некоторой 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*), информация о его соответствии занимаемой должности (для простоты – «да» или «нет») (*СОТР_СТАТ*), размер зарплаты (*СОТР_ЗАРП*), номер отдела (*СОТР_ОТД_НОМЕР*)? Поскольку необходимо 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 выбрать все записи о сотрудниках отдела и посчитать соответствующие общие значения.

Таким образом, видно, что даже для такой простой системы ее реализация на базе файловой системы, во-первых, требует создания достаточно сложной надстройки для многоключевого доступа к файлам. Во-вторых, вызывает необходимость в избыточности хранения данных (для каждого сотрудника одного отдела повторяется имя руководителя). Кроме того, для получения суммарной информации об отделах необходимо выполнение массовой выборки и многочисленных вычислений. Далее, если в ходе эксплуатации системы необходимо, например, выдать 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 тогда должен содержать сле-

дующие 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 будут преодолены. Каждый из файлов будет содержать только не дублируемую 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 первый шаг в сторону схемы базы данных). Она также должна определить структуру и смысл каждого поля (например, что *СОТР_ОТД_НОМЕР* в файле *СОТРУДНИКИ* и *ОТД_НОМЕР* в файле *ОТДЕЛЫ* означают одно и то же) и 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 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 рассматриваемом 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

воспользуется 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 в 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, что необходимо обеспечить параллельную (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. СУБД обеспечивают более тонкую синхронизацию параллельного доступа к данным.

Таким образом, СУБД решают множество проблем, которые затруднительно или вообще невозможно решить при использовании файловых систем.

2 ПОНЯТИЕ СУБД. ФУНКЦИИ. ВНУТРЕННЯЯ АРХИТЕКТУРА

2.1 Функции СУБД. Типовая организация СУБД

СУБД принято называть прикладную информационную систему (комплекс программных средств), опирающуюся на некоторую систему управления данными, и обладающую следующим минимальным набором функций:

1) управление данными во внешней памяти;
2) управление буферами оперативной памяти;
3) управление транзакциями;
4) журнализация;
5) поддержка языков БД.

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 служебных целей, например для ускорения доступа к данным (обычно для этого используются индексы).

В некоторых реализациях СУБД активно используются возможности существующих файловых систем, в других же задачи решаются даже на уровне устройств внешней памяти. В любом случае в хорошо развитых СУБД пользователи не обязаны знать, использует ли СУБД файловую систему и, если использует, то каким образом в ней организованы файлы.

Управление буферами оперативной памяти. СУБД обычно работают с БД значительного размера; по крайней мере этот размер 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.

Отметим, 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.

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. Транзакции необходимы для поддержания

логической целостности БД и наибольшее значение приобретают в многопользовательских СУБД.

Каждая транзакция начинается при целостном состоянии БД и оставляет это состояние целостным после своего завершения. Это свойство 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.

С управлением транзакциями в многопользовательской СУБД также 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 последовательного 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.

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. Примеры программных сбоев: 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). 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. Для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций 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 в 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 остановимся на этом при рассмотрении внутренней организации баз данных.

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Архивная копия – это полная копия БД к моменту начала заполнения журнала (имеется много вариантов и 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 повышенные 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.

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. Можно особо выделить 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. Рассмотрим более подробно языки ранних СУБД в последующих лекциях.

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), который более подробно будет рассмотрен в 5-м разделе. Перечислим пока только 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. 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 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 поддерживается на языковом уровне.

В заключение хотелось бы еще раз обратить внимание на то, что полная поддержка всех пяти функций является обязательным требованием к любой современной СУБД.

2.2 Типовая организация современной СУБД

Организация типичной СУБД и состав ее компонентов полностью соответствуют рассмотренному набору функций.

Логически в современной реляционной СУБД можно выделить внутреннюю часть – ядро СУБД (часто его называют Data Base Engine), компилятор языка БД (обычно SQL), подсистему поддержки времени выполнения и 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. Функции этих компонентов взаимосвязаны, и для обеспечения корректной работы СУБД все они должны взаимодействовать по тщательно продуманным и проверенным 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 (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. Поэтому компилятор должен решить, каким образом выполнять оператор языка, прежде чем произвести программу. 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 РЕЛЯЦИОННЫЙ ПОДХОД К ОРГАНИЗАЦИИ БАЗ ДАННЫХ

Исторически можно выделить несколько подходов к организации баз данных. Каждый из подходов описывает определенную логическую модель, в соответствии с которой СУБД осуществляет хранение и взаимодействие объектов, входящих в базу данных. Данная модель определяет все основные алгоритмы ядра и является главной характеристикой любой СУБД.

В настоящее время, наиболее распространенным является реляционный подход к организации БД. Первые прототипы реляционных СУБД появились еще в 70-х гг. XX в. Очевидные преимущества и постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже к середине 80-х годов реляционные системы практически полностью вытеснили с мирового рынка ранние СУБД (иерархические и сетевые).

Рассмотрим основные понятия и принципы, лежащие в основе реляционного подхода.

3.1 Общие понятия реляционного подхода к организации БД

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

Раскроем смысл этих понятий на примере отношения *СОТРУДНИКИ*, содержащего информацию о сотрудниках некоторой организации.

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».

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.

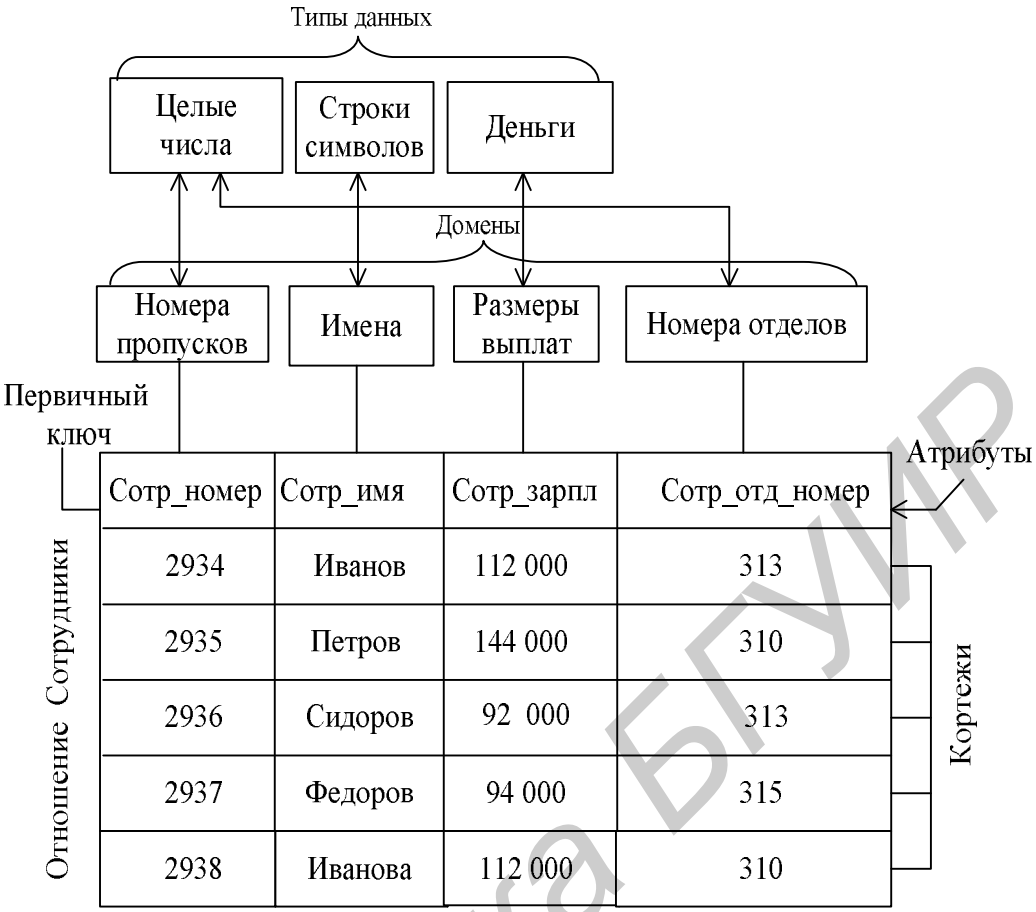

Рисунок.1 – Раскрытие основных понятий реляционных БД на примере отношения *СОТРУДНИКИ*

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.

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.

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 «отношение-экземпляр», иногда схему отношения называют заголовком отношения, а отношение как набор кортежей – телом отношения.

Обычным житейским представлением отношения является таблица, заголовком которой является схема отношения, строками – кортежи отношения-экземпляра, а имена атрибутов – столбцы этой таблицы. При рассмотрении практических вопросов организации реляционных баз данных и средств управления, 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, несмотря на то что в теории реляционных БД все они определяются абсолютно формально и точно.

3.2 Фундаментальные свойства отношений

Перечислим некоторые наиболее важные свойства отношений:

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. В качестве первичного ключа выступает набор 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 ос-

новного свойства. Понятие первичного ключа является исключительно важным в связи с понятием целостности баз данных.

2. ***Отсутствие упорядоченности кортежей.*** Свойство отсутствия упорядоченности кортежей отношения также является 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.

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.

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).

3.3 Реляционная модель данных

Вернемся и более подробно остановимся на понятии 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 об иерархической, сетевой, некоторой семантической и т.д. моделях данных, необходимо отметить, что это понятие было введено в обиход применительно к реляционным системам и наиболее эффективно используется именно в этом контексте.

Наиболее распространенная трактовка реляционной модели данных, по-видимому, принадлежит Дейту. Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной, манипуляционной и целостной частей.

В ***структурной части*** модели фиксируется, что единственной структурой данных, используемой в реляционных БД, является нормализованное n-арное отношение.

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.

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. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Второе требование называется **требованием целостности по ссылкам.** При соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений, связанных между собой посредством первичных и вторичных (внешних) ключей.

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).

3.4 Базисные средства манипулирования реляционными данными

Все механизмы, лежащие в основе манипуляционной части реляционной модели данных, 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.

Реляционные 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 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, 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. Но 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 ставит 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.

3.4.1 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. Опишем 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 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 вы-

ражений, и операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

3.4.2 Общая интерпретация реляционных операций

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 отношение, включающее кортежи отношения-операнда, удовлетворяющие 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 с 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.

3.4.3 Реляционное исчисление

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

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

1) выполнить соединение отношений *СОТРУДНИКИ* и *ОТДЕЛЫ* по условию *СОТР_НОМ = ОТД_НАЧ*;

2) ограничить полученное отношение по условию *ОТД_КОЛ* > 50;

3) спроецировать результат предыдущей операции на атрибут *СОТР_ИМЯ*, *СОТР_НОМ*.

Перед нами четко сформулированная 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 получилась бы формула, 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 указаны лишь характеристики результирующего отношения, но ничего не сказано 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. На самом деле эти два механизма эквивалентны, и существуют не очень сложные правила преобразования одного формализма в другой.

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.

3.5 Достоинства и недостатки реляционного подхода к организации БД

Реляционный подход к организации баз данных в настоящее время является наиболее распространенным, хотя наряду с общепризнанными достоинствами обладает и рядом недостатков.

К числу достоинств реляционного подхода можно отнести:

- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными;
- наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации баз данных;
- возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

В настоящее время основным предметом критики реляционных СУБД является их недостаточная эффективность при использовании в нетрадиционных областях (наиболее распространенными примерами которых являются системы автоматизации проектирования), в которых требуются предельно сложные структуры данных. Еще одним часто отмечаемым недостатком реляционных баз данных является невозможность адекватного отражения семантики предметной области. Другими словами, возможности представления знаний о семантической специфике предметной области в реляционных системах ограничены. Современные исследования в области постреляционных систем главным образом посвящены именно устранению этих недостатков.

4 ВНУТРЕННЯЯ ОРГАНИЗАЦИЯ РЕЛЯЦИОННЫХ СУБД

4.1 Организация внешней памяти

Реляционные СУБД обладают рядом особенностей, влияющих на организацию внешней памяти. К наиболее важным можно отнести следующие:

1. Наличие двух уровней системы: уровня непосредственного управления данными во внешней памяти и языкового уровня. При такой организации подсистема нижнего уровня должна поддерживать во внешней памяти набор базовых структур, конкретная интерпретация которых входит в число функций подсистемы верхнего уровня.

2. Поддержание отношений-каталогов. Информация, связанная с именованием объектов базы данных и их конкретными свойствами (например структура ключа индекса), поддерживается подсистемой языкового уровня. С точки зрения структур внешней памяти отношение-каталог ничем не отличается от обычного отношения базы данных.

3. Регулярность структур данных. Поскольку основным объектом реляционной модели данных является плоская таблица, главный набор объектов внешней памяти может иметь очень простую регулярную структуру. При этом необходимо обеспечить возможность эффективного выполнения операторов языкового уровня как над одним отношением (простые селекция и проекция), так и над несколькими отношениями (наиболее распространено и трудоемко соединение нескольких отношений). Для этого во внешней памяти должны поддерживаться дополнительные «управляющие» структуры - индексы.

4. Для выполнения требования надежного хранения баз данных необходимо поддерживать избыточность хранения данных, что обычно реализуется в виде журнала изменений базы данных.

Соответственно возникают следующие разновидности объектов во внешней памяти базы данных:

- строки отношений – основная часть базы данных, большей частью непосредственно видимая пользователям;
- управляющие структуры – индексы, создаваемые по инициативе пользователя (администратора) или верхнего уровня системы из соображений повышения эффективности выполнения запросов и обычно автоматически поддерживаемые нижним уровнем системы;
- журнальная информация, поддерживаемая для удовлетворения потребности в надежном хранении данных;
- служебная информация, поддерживаемая для удовлетворения внутренних потребностей нижнего уровня системы (например информация о свободной памяти).

Хранение отношений. Существуют два принципиальных подхода к физическому хранению отношений. Наиболее распространенным является покортежное хранение отношений (кортеж является единицей физического хране-

ния). Естественно, это обеспечивает быстрый доступ к целому кортежу, но при этом во внешней памяти дублируются общие значения разных кортежей одного отношения и, вообще говоря, могут потребоваться лишние обмены с внешней памятью, если нужна часть кортежа.

Альтернативным (менее распространенным) подходом является хранение отношения по столбцам, т.е. единицей хранения является столбец отношения с исключенными дубликатами. Естественно, что при такой организации суммарно в среднем тратится меньше внешней памяти, поскольку дубликаты значений не хранятся; за один обмен с внешней памятью в общем случае считывается больше полезной информации. Дополнительным преимуществом является возможность использования значений столбца отношения для оптимизации выполнения операций соединения. Но при этом требуются существенные дополнительные действия для сборки целого кортежа (или его части).

Индексы. Как бы не были организованы индексы в конкретной СУБД, их основное назначение состоит в обеспечении эффективного прямого доступа к кортежу отношения по ключу. Обычно индекс определяется для одного отношения, и ключом является значение атрибута (возможно, составного). Если ключом индекса является возможный ключ отношения, то индекс должен обладать свойством уникальности, т.е. не содержать дубликатов ключа. На практике ситуация выглядит обычно противоположно: при объявлении первичного ключа отношения автоматически заводится уникальный индекс, а единственным способом объявления возможного ключа, отличного от первичного, является явное создание уникального индекса. Это связано с тем, что для проверки сохранения свойства уникальности возможного ключа так или иначе требуется индексная поддержка.

Поскольку при выполнении многих операций языкового уровня требуется сортировка отношений в соответствии со значениями некоторых атрибутов, полезным свойством индекса является обеспечение последовательного просмотра кортежей отношения в диапазоне значений ключа в порядке возрастания или убывания этих значений.

Наконец, одним из способов оптимизации выполнения эквисоединения отношений (наиболее распространенная из числа дорогостоящих операций) является организация так называемых мультииндексов для нескольких отношений, обладающих общими атрибутами. Любой из этих атрибутов (или их набор) может выступать в качестве ключа мультииндекса. Значению ключа сопоставляется набор кортежей всех связанных мультииндексом отношений, значения выделенных атрибутов которых совпадают со значением ключа.

Общей идеей любой организации индекса, поддерживающего прямой доступ по ключу и последовательный просмотр в порядке возрастания или убывания значений ключа, является хранение упорядоченного списка значений ключа с привязкой к каждому значению ключа списка идентификаторов кортежей. Одна организация индекса отличается от другой главным образом в способе поиска ключа с заданным значением.

B-дерево. Видимо, наиболее популярным подходом к организации индексов в базах данных является использование техники B-деревьев. С точки зрения внешнего логического представления B-дерево – это сбалансированное сильно ветвистое дерево во внешней памяти. Сбалансированность означает, что длина пути от корня дерева к любому его листу одна и та же. Ветвистость дерева – это свойство каждого узла дерева ссылаться на большое число узлов-потомков. С точки зрения физической организации B-дерево представляется как мультисписочная структура страниц внешней памяти, т.е. каждому узлу дерева соответствует блок внешней памяти (страница). Внутренние и листовые страницы обычно имеют разную структуру.

Поиск в B-дереве – это прохождение от корня к листу в соответствии с заданным значением ключа. Заметим, что поскольку деревья сильноветвистые и сбалансированные, то для выполнения поиска по любому значению ключа потребуется одно и то же (и обычно небольшое) число обменов с внешней памятью. Более точно: в сбалансированном дереве, где длины всех путей от корня к листу одни и те же, если во внутренней странице помещается n ключей, то при хранении m записей требуется дерево глубиной logn(m), где logn вычисляет логарифм по основанию n. Если n достаточно велико (обычный случай), то глубина дерева невелика, и производится быстрый поиск.

Основной «изюминкой» B-деревьев является автоматическое поддержание свойства сбалансированности. При выполнении операций вставки и удаления свойство сбалансированности B-дерева сохраняется, а внешняя память расходуется достаточно экономно.

Проблемой является то, что при выполнении операций модификации слишком часто могут возникать расщепления и слияния.

Хэширование. Альтернативным и все более популярным подходом к организации индексов является использование техники хэширования. Это очень обширная тема, которая заслуживает отдельного рассмотрения. Ограничимся лишь несколькими замечаниями. Общей идеей методов хэширования является применение к значению ключа некоторой функции свертки (хэш-функции), вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи.

В самом простом, классическом случае свертка ключа используется как адрес в таблице, содержащей ключи и записи. Основным требованием к хэш-функции является равномерное распределение значения свертки. При возникновении коллизий (одна и та же свертка для нескольких значений ключа) образуются цепочки переполнения. Главным ограничением этого метода является фиксированный размер таблицы. Если таблица заполнена слишком сильно или переполнена, то возникнет слишком много цепочек переполнения и главное преимущество хэширования – доступ к записи почти всегда за одно обращение к таблице – будет утрачено. Расширение таблицы требует ее полной переделки на основе новой хэш-функции (со значением свертки большего размера).

В случае баз данных такие действия являются абсолютно неприемлемыми. Поэтому обычно вводят промежуточные таблицы-справочники, содержа-

щие значения ключей и адреса записей, а сами записи хранятся отдельно. Тогда при переполнении справочника требуется только его переделка, что вызывает меньше накладных расходов.

Чтобы избежать потребности полной переделки справочников, при их организации часто используют технику B-деревьев с расщеплениями и слияниями. Хэш-функция при этом меняется динамически, в зависимости от глубины B-дерева. Путем дополнительных технических ухищрений удается добиться сохранения порядка записей в соответствии со значениями ключа. В целом методы B-деревьев и хэширования все более сближаются.

Журнальная информация. Структура журнала обычно является сугубо частным делом конкретной реализации. Отметим только самые общие свойства.

Журнал обычно представляет собой чисто последовательный файл с записями переменного размера, которые можно просматривать в прямом или обратном порядке. Обмены производятся стандартными порциями (страницами) с использованием буфера оперативной памяти. В грамотно организованных системах структура (и тем более смысл) журнальных записей известна только компонентам СУБД, ответственным за журнализацию и восстановление. Поскольку содержимое журнала является критичным при восстановлении базы данных после сбоев, к ведению файла журнала предъявляются особые требования по части надежности. В частности, обычно стремятся поддерживать две идентичные копии журнала на разных устройствах внешней памяти.

Служебная информация. Для корректной работы подсистемы управления данными во внешней памяти необходимо поддерживать информацию, которая используется только этой подсистемой и не видна подсистеме языкового уровня. Набор структур служебной информации зависит от общей организации системы, но обычно требуется поддержание следующих служебных данных:

1. Внутренние каталоги, описывающие физические свойства объектов базы данных, например, число атрибутов отношения, их размер и, возможно, типы данных; описание индексов, определенных для данного отношения и т.д.

2. Описатели свободной и занятой памяти в страницах отношения. Такая информация требуется для нахождения свободного места при занесении кортежа. Отдельно приходится решать задачу поиска свободного места в случаях некластеризованных и кластеризованных отношений (в последнем случае приходится дополнительно использовать кластеризованный индекс).

3. Связывание страниц одного отношения. Если в одном файле внешней памяти могут располагаться страницы нескольких отношений (обычно к этому стремятся), то нужно каким-то образом связать страницы одного отношения. Тривиальный способ использования прямых ссылок между страницами часто приводит к затруднениям при синхронизации транзакций (например особенно трудно освобождать и заводить новые страницы отношения). Поэтому стараются использовать косвенное связывание страниц с использованием служебных индексов. В частности, известен общий механизм для описания свободной памяти и связывания страниц на основе B-деревьев.

4.2 Управление транзакциями

Поддержание механизма транзакций – показатель уровня развитости СУБД. Корректное поддержание транзакций одновременно является основой обеспечения целостности баз данных (и поэтому транзакции вполне уместны и в однопользовательских персональных СУБД), а также составляет базис изолированности пользователей в многопользовательских системах. Часто эти два аспекта рассматриваются по отдельности, но на самом деле они взаимосвязаны.

Заметим, что хотя с точки зрения обеспечения целостности баз данных механизм транзакций следовало бы поддерживать в персональных СУБД, на практике это обычно не выполняется. Поэтому при переходе от персональных к многопользовательским СУБД пользователи сталкиваются с необходимостью четкого понимания природы транзакций.

Как уже было сказано, под транзакцией понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации). Причем либо результаты всех операторов, входящих в транзакцию, отображаются в БД, либо воздействие всех этих операторов полностью отсутствует. Лозунг транзакции – «Все или ничего». При завершении транзакции оператором *COMMIT* результаты гарантированно фиксируются во внешней памяти (commit – «зафиксировать» результаты транзакции); при завершении транзакции оператором *ROLLBACK* результаты гарантировано отсутствуют во внешней памяти (rollback – «ликвидировать» результаты транзакции).

Понятие транзакции имеет непосредственную связь с понятием целостности БД. Очень часто БД может обладать такими ограничениями целостности, которые просто невозможно не нарушить, выполняя только один оператор изменения БД. Поэтому для поддержания подобных ограничений целостности допускается их нарушение внутри транзакции с тем условием, чтобы к моменту завершения транзакции условия целостности были соблюдены. В системах с развитыми средствами ограничения и контроля целостности каждая транзакция начинается при целостном состоянии БД и должна оставить это состояние целостными после своего завершения. Несоблюдение этого условия приводит к тому, что вместо фиксации результатов транзакции происходит ее откат (т.е. вместо оператора *COMMIT* выполняется оператор *ROLLBACK*), и БД остается в таком состоянии, в котором находилась к моменту начала транзакции, т.е. в целостном состоянии.

Различаются два вида ограничений целостности: немедленно проверяемые и откладываемые. К немедленно проверяемым ограничениям целостности относятся такие ограничения, проверку которых бессмысленно или даже невозможно откладывать. Примером ограничения, проверку которого откладывать бессмысленно, являются ограничения домена (возраст сотрудника не может превышать 85 лет). Более сложным ограничением, проверку которого невозможно отложить, является следующее: зарплата сотрудника не может быть

увеличена за одну операцию более, чем на 100.000 рублей. Немедленно проверяемые ограничения целостности соответствуют уровню отдельных операторов языкового уровня СУБД. При их нарушениях не производится откат транзакции, а лишь отвергается соответствующий оператор.

Откладываемые ограничения целостности – это ограничения на базу данных, а не на какие-либо отдельные операции. По умолчанию такие ограничения проверяются при конце транзакции, и их нарушение вызывает автоматическую замену оператора *COMMIT* на оператор *ROLLBACK*. Однако в некоторых системах поддерживается специальный оператор насильственной проверки ограничений целостности внутри транзакции. Если после выполнения такого оператора обнаруживается, что условия целостности не выполнены, пользователь может сам выполнить оператор *ROLLBACK* или постараться устранить причины нецелостного состояния базы данных внутри транзакции (видимо, это осмысленно только при использовании интерактивного режима работы).

С точки зрения внешнего представления в момент завершения транзакции проверяются все откладываемые ограничения целостности, определенные в этой базе данных. Однако при реализации стремятся при выполнении транзакции динамически выделить те ограничения целостности, которые действительно могли бы быть нарушены. Например, если при выполнении транзакции над базой данных *СОТРУДНИКИ-ОТДЕЛЫ* в ней не выполнялись операторы вставки или удаления кортежей из отношения *СОТРУДНИКИ,* то проверять упоминавшееся выше ограничение целостности не требуется (а проверка подобных ограничений вызывает достаточно большую работу).

4.3 Изолированность пользователей

В многопользовательских системах с одной базой данных одновременно могут работать несколько пользователей или прикладных программ. Предельной задачей системы является обеспечение изолированности пользователей, т.е. создание достоверной и надежной иллюзии того, что каждый из пользователей работает с БД в одиночку.

В связи со свойством сохранения целостности БД транзакции являются подходящими единицами изолированности пользователей. Действительно, если с каждым сеансом работы с базой данных ассоциируется транзакция, то каждый пользователь начинает работу с согласованным состоянием базы данных, т.е. с таким состоянием, в котором база данных могла бы находиться, даже если бы пользователь работал с ней в одиночку.

Понятно, что для того, чтобы добиться изолированности транзакций, в СУБД должны использоваться какие-либо методы регулирования совместного выполнения транзакций.

План (способ) выполнения набора транзакций называется сериальным, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций.

Сериализация транзакций – это механизм их выполнения по некоторому сериальному плану. Обеспечение такого механизма является основной функцией компонента СУБД, ответственного за управление транзакциями. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей.

Основная реализационная проблема состоит в выборе метода сериализации набора транзакций, который не слишком ограничивал бы их параллельность. Тривиальным решением является действительно последовательное выполнение транзакций. Но существуют ситуации, в которых можно выполнять операторы разных транзакций в любом порядке с сохранением сериальности. Примерами могут служить только читающие транзакции, а также транзакции, не конфликтующие по объектам базы данных.

Существуют два базовых подхода к сериализации транзакций – основанный на синхронизационных захватах объектов базы данных и на использовании временных меток.

4.3.1 Синхронизационные захваты

Наиболее распространенным в централизованных СУБД (включающих системы, основанные на архитектуре «клиент-сервер») является подход, основанный на соблюдении двухфазного протокола синхронизационных захватов объектов БД. В общих чертах протокол состоит в том, что перед выполнением любой операции в транзакции T над объектом базы данных r от имени транзакции T запрашивается синхронизационный захват объекта r в соответствующем режиме (в зависимости от вида операции).

Основными режимами синхронизационных захватов являются:

1. *Совместный* – S (Shared), означающий разделяемый захват объекта и требуемый для выполнения операции чтения объекта.
2. *Монопольный* – X (eXclusive), означающий монопольный захват объекта и требуемый для выполнения операций занесения, удаления и модификации.

Захваты объектов несколькими транзакциями по чтению совместимы, т.е. нескольким транзакциям допускается читать один и тот же объект, захват объекта одной транзакцией по чтению не совместим с захватом другой транзакцией того же объекта по записи, и захваты одного объекта разными транзакциями по записи не совместимы. Правила совместимости захватов одного объекта разными транзакциями изображены в таблице 1:

Таблица 1 – Правила совместимости захватов

Состояния объекта	**X**	**S**
Захват не установлен	да	да
X	нет	нет
S	нет	да

В первом столбце приведены возможные состояния объекта с точки зрения синхронизационных захватов. Транзакция, запросившая синхронизационный захват объекта БД, уже захваченный другой транзакцией в несовместимом режиме, блокируется до тех пор, пока захват с этого объекта не будет снят.

При соблюдении двухфазного протокола синхронизационных захватов действительно обеспечивается сериализация транзакций на третьем уровне изолированности. Основная проблема состоит в том, что следует считать объектом для синхронизационного захвата.

В контексте реляционных баз данных возможны следующие альтернативы:

- файл – физический (с точки зрения базы данных) объект, область хранения нескольких отношений и, возможно, индексов;
- отношение – логический объект, соответствующий множеству кортежей данного отношения;
- страница данных – физический объект, хранящий кортежи одного или нескольких отношений, индексную или служебную информацию;
- кортеж – элементарный физический объект базы данных.

Чем крупнее объект синхронизационного захвата (не существенно, какой природы этот объект – логический или физический), тем меньше синхронизационных захватов будет поддерживаться в системе, и на это, соответственно, будут тратиться меньшие накладные расходы. Но вся беда в том, что при использовании для захватов крупных объектов возрастает вероятность конфликтов транзакций и тем самым уменьшается допускаемая степень их параллельного выполнения.

В большинстве современных систем используются покортежные синхронизационные захваты.

Но если единицей захвата является кортеж, то какие синхронизационные захваты потребуются при выполнении таких операций, как уничтожение отношения? Было бы довольно нелепо перед выполнением такой операции потребовать захвата всех существующих кортежей отношения. Кроме того, это не предотвратило бы возможности параллельной вставки в другой транзакции нового кортежа в уничтожаемое отношение.

Подобные рассуждения привели к разработке аппарата гранулированных синхронизационных захватов. При применении этого подхода синхронизационные захваты могут запрашиваться по отношению к объектам разного уровня: файлам, отношениям и кортежам. Требуемый уровень объекта определяется тем, какая операция выполняется (например, для выполнения операции уничтожения отношения объектом синхронизационного захвата должно быть все отношение, а для выполнения операции удаления кортежа – этот кортеж). Объект любого уровня может быть захвачен в режиме S или X.

Теперь введем наиболее важное отличие, на котором, собственно, и держится соответствие захватов разного уровня. Это специальные протоколы гранулированных захватов и новые типы захватов: перед захватом объекта в ре-

жиме S или X соответствующий объект более верхнего уровня должен быть захвачен в режиме IS, IX или SIX:

- IS (Intented for Shared lock) по отношению к некоторому составному объекту O означает намерение захватить некоторый входящий в O объект в совместном режиме. Например, при намерении читать кортежи из отношения R это отношение должно быть захвачено в режиме IS (а до этого в таком же режиме должен быть захвачен файл);
- IX (Intented for eXclusive lock) по отношению к некоторому составному объекту O означает намерение захватить некоторый входящий в O объект в монопольном режиме. Например, при намерении удалять кортежи из отношения R это отношение должно быть захвачено в режиме IX (а до этого в таком же режиме должен быть захвачен файл);
- SIX (Shared, Intented for eXclusive lock) по отношению к некоторому составному объекту O означает совместный захват всего этого объекта с намерением впоследствии захватывать какие-либо входящие в него объекты в монопольном режиме. Например, если выполняется длинная операция просмотра отношения с возможностью удаления некоторых просматриваемых кортежей, то экономичнее всего захватить это отношение в режиме SIX (а до этого захватить файл в режиме IS).

Возможные ситуации описаны в таблице 2.

Таблица 2 – Совместимость захватов

	X	**S**	**IX**	**IS**	**SIX**
Захват не установлен	да	да	да	да	да
X	нет	нет	нет	нет	нет
S	нет	да	нет	да	нет
IX	нет	нет	да	да	нет
IS	нет	да	да	да	да
SIX	нет	нет	нет	да	нет

Одним из наиболее чувствительных недостатков метода сериализации транзакций на основе синхронизационных захватов является возможность возникновения тупиков (deadlocks) между транзакциями. Тупики возможны при применении любого из рассмотренных вариантов.

Вот простой пример возникновения тупика между транзакциями T1 и T2.

Транзакции T1 и T2 установили монопольные захваты объектов r1 и r2 соответственно, после чего T1 требуется совместный захват r2, а T2 – совместный захват r1. Ни одна из транзакций не может продолжаться, следовательно, монопольные захваты не будут сняты, а совместные – не будут удовлетворены.

Поскольку тупики возможны и никакого естественного выхода из тупиковой ситуации не существует, то эти ситуации необходимо обнаруживать и искусственно устранять.

Основой обнаружения тупиковых ситуаций является построение (или постоянное поддержание) графа ожидания транзакций. Граф ожидания транзакций – это ориентированный двудольный граф, в котором существует два типа вершин: вершины, соответствующие транзакциям, и вершины, соответствующие объектам захвата. В этом графе существует дуга, ведущая из вершины-транзакции к вершине-объекту, если для этой транзакции существует удовлетворенный захват объекта. В графе существует дуга из вершины-объекта к вершине-транзакции, если транзакция ожидает удовлетворения захвата объекта.

Легко показать, что в системе существует ситуация тупика, если в графе ожидания транзакций имеется хотя бы один цикл. Для распознавания тупика периодически производится построение графа ожидания транзакций (как уже отмечалось, иногда граф ожидания поддерживается постоянно), и в этом графе ищутся циклы.

Если удалось найти цикл в графе ожидания транзакций, нужно каким-то образом обеспечить возможность продолжения работы хотя бы для части транзакций, попавших в тупик. Разрушение тупика начинается с выбора в цикле транзакций так называемой транзакции-жертвы, т.е. транзакции, которой решено пожертвовать, чтобы обеспечить возможность продолжения работы других транзакций. Критерием выбора является стоимость транзакции; жертвой выбирается самая дешевая транзакция. Стоимость транзакции определяется на основе многофакторной оценки, в которую с разными весами входят время выполнения, число накопленных захватов, приоритет и т.д.

После выбора транзакции-жертвы выполняется откат этой транзакции, который может носить полный или частичный характер. При этом, естественно, освобождаются захваты и может быть продолжено выполнение других транзакций.

Естественно, такое насильственное устранение тупиковых ситуаций является нарушением принципа изолированности пользователей, которого невозможно избежать.

Чтобы минимизировать число конфликтов между транзакциями, в некоторых СУБД (например в Oracle) используется следующее развитие подхода. Монопольный захват объекта блокирует только изменяющие транзакции. После выполнения операции модификации предыдущая версия объекта остается доступной для чтения в других транзакциях. Кратковременная блокировка чтения требуется только на период фиксации изменяющей транзакции, когда обновленные объекты становятся текущими.

4.3.2 Метод временных меток

Альтернативный метод сериализации транзакций, хорошо работающий в условиях редких конфликтов транзакций и не требующий построения графа ожидания транзакций, основан на использовании временных меток.

Основная идея метода состоит в следующем: если транзакция T1 началась раньше транзакции T2, то система обеспечивает такой режим выполнения, как если бы T1 была целиком выполнена до начала T2.

Для этого каждой транзакции T предписывается временная метка t, соответствующая времени начала T. При выполнении операции над объектом r транзакция T помечает его своей временной меткой и типом операции (чтение или изменение). Перед выполнением операции над объектом r транзакция T1 выполняет следующие действия:

1. Проверяет, не закончилась ли транзакция T, пометившая этот объект. Если T закончилась, T1 помечает объект r и выполняет свою операцию. Если транзакция T не завершилась, то T1 проверяет конфликтность операций. Если операции неконфликтны, при объекте r остается или проставляется временная метка с меньшим значением и транзакция T1 выполняет свою операцию.

2. Если операции T1 и T конфликтуют:

если t(T) > t(T1) (транзакция T является более «молодой», чем T), производится откат T и T1 продолжает работу;

если t(T) < t(T1) (T "старше" T1), то T1 получает новую временную метку и начинается заново.

К недостаткам метода временных меток относятся потенциально более частые откаты транзакций, чем в случае использования синхронизационных захватов. Это связано с тем, что конфликтность транзакций определяется более грубо.

4.4 Журнализация изменений БД

Одним из основных требований к развитым СУБД является надежность хранения баз данных. Это требование предполагает, в частности, возможность восстановления согласованного состояния базы данных после любого рода аппаратных и программных сбоев. Очевидно, что для выполнения восстановлений необходима некоторая дополнительная информация. В подавляющем большинстве современных реляционных СУБД такая избыточная дополнительная информация поддерживается в виде журнала изменений базы данных.

Итак, общей целью журнализации изменений баз данных является обеспечение возможности восстановления согласованного состояния базы данных после любого сбоя. Поскольку основой поддержания целостного состояния базы данных является механизм транзакций, журнализация и восстановление тесно связаны с понятием транзакции. Общими принципами восстановления являются следующие:

- результаты зафиксированных транзакций должны быть сохранены в восстановленном состоянии базы данных;
- результаты незафиксированных транзакций должны отсутствовать в восстановленном состоянии базы данных.

Это, собственно, и означает, что восстанавливается последнее по времени согласованное состояние базы данных.

Возможны следующие ситуации, при которых требуется производить восстановление состояния базы данных:

1. Индивидуальный откат транзакции. Тривиальной ситуацией отката транзакции является ее явное завершение оператором ROLLBACK. Возможны также ситуации, когда откат транзакции инициируется системой. Примерами могут быть возникновение исключительной ситуации в прикладной программе (например деление на ноль) или выбор транзакции в качестве жертвы при обнаружении синхронизационного тупика. Для восстановления согласованного состояния базы данных при индивидуальном откате транзакции нужно устранить последствия операторов модификации базы данных, которые выполнялись в этой транзакции.

2. Восстановление после внезапной потери содержимого оперативной памяти (мягкий сбой). Такая ситуация может возникнуть при аварийном выключении электрического питания, при возникновении неустранимого сбоя процессора (например срабатывании контроля оперативной памяти) и т.д. Ситуация характеризуется потерей той части базы данных, которая к моменту сбоя содержалась в буферах оперативной памяти.

3. Восстановление после поломки основного внешнего носителя базы данных (жесткий сбой). Эта ситуация при достаточно высокой надежности современных устройств внешней памяти может возникать сравнительно редко, но тем не менее СУБД должна быть в состоянии восстановить базу данных даже и в этом случае. Основой восстановления является архивная копия и журнал изменений базы данных.

Во всех трех случаях основой восстановления является избыточное хранение данных. Эти избыточные данные хранятся в журнале, содержащем последовательность записей об изменении базы данных.

Возможны два основных варианта ведения журнальной информации. В первом варианте для каждой транзакции поддерживается отдельный локальный журнал изменений базы данных этой транзакцией. Эти локальные журналы используются для индивидуальных откатов транзакций и могут поддерживаться в оперативной (правильнее сказать, в виртуальной) памяти. Кроме того, поддерживается общий журнал изменений базы данных, используемый для восстановления состояния базы данных после мягких и жестких сбоев.

Этот подход позволяет быстро выполнять индивидуальные откаты транзакций, но приводит к дублированию информации в локальных и общем журналах. Поэтому чаще используется второй вариант – поддержание только общего журнала изменений базы данных, который используется и при выполнении индивидуальных откатов.

5 ЯЗЫК SQL

5.1 Общие сведения

Рассматривая вопросы, связанные с БД и СУБД, было бы нелогично оставить в стороне язык баз данных – SQL. Чтобы получить общее представление об его особенностях и возможностях, вполне достаточно ограничиться перечислением основных операторов языка.

Язык SQL (эта аббревиатура должна произноситься как «си-ку-ель», однако все чаще говорят «эс-ку-эль») в настоящее время является промышленным стандартом, который в большей или меньшей степени поддерживает любая СУБД, претендующая на звание «реляционной».

В начале 70-х гг. в компании IBM была разработана экспериментальная СУБД SystemR на основе языка SEQUEL (Structured English Qeury Language – структурированный английский язык запросов), который можно считать непосредственным предшественником SQL. Целью разработки было создание простого непроцедурного языка, которым мог воспользоваться любой пользователь, даже не имеющий навыков программирования. В 1981 г. IBM объявила о своем первом основанном на SQL программном продукте – SQL/DS. Чуть позже к ней присоединились Oracle и другие производители. Первый стандарт языка SQL был принят Американским национальным институтом стандартизации (ANSI) в 1987 г. (так называемый SQL level 1) и несколько уточнен в 1989 г. (SQL level 2). Дальнейшее развитие языка поставщиками СУБД потребовало принятия в 1992 г. нового расширенного стандарта (ANSI SQL-92 или просто SQL-2). В настоящее время ведется работа по подготовке третьего стандарта SQL, который должен включать элементы объектно-ориентрованного доступа к данным.

В SQL определены два подмножества языка:

1. **SQL-DDL** (Data Definition Language) – язык определения структур и ограничений целостности баз данных. Сюда относятся команды создания и удаления баз данных; создания, изменения и удаления таблиц; управления пользователями и т.д.
2. **SQL-DML** (Data Manipulation Language) – язык манипулирования данными: добавление, изменение, удаление и извлечение данных, управления транзакциями

Как и большинство языков программирования, SQL реализует стандартный набор типов данных дополненный парой специфических для баз данных типов значений. Перечислим основные типы данных языка:

- символьные типы данных – содержат буквы, цифры и специальные символы: *CHAR* или *CHAR(n),VARCHAR(n)* ;
- целые типы данных – поддерживают только целые числа (дробные части и десятичные точки не допускаются): *INTEGER* или *INT,SMALLINT* ;
- вещественные типы данных – описывают числа с дробной частью: *FLOAT* и *SMALLFLOAT* ,*DECIMAL(p), DECIMAL(p,n)* ;

- денежные типы данных – описывают, естественно, денежные величины: *MONEY(p,n)* ;.
- дата и время – используются для хранения даты, времени и их комбинаций: *DATE,TIME,INTERVAL,DATETIME* ;
- двоичные типы данных – позволяют хранить данные любого объема в двоичном коде (оцифрованные изображения, исполняемые файлы и т.д.): *BINARY ,BYTE, BLOB* ;
- последовательные типы данных – используются для представления возрастающих числовых последовательностей: *SERIAL*;

5.2 DDL: Операторы создания схемы базы данных

Перечислим основные операторы DDL (таблицы 3 – 5).

Таблица 3 – Операторы базы данных

Команда	Описание
Создание базы данных.	**CREATE DATABASE <имя_базы_данных>**
Удаление базы данных.	**DROP DATABASE <имя_базы_данных>**

Таблица 4 – Операторы модификации таблиц

Команда	Описание
Добавить столбцы	**ALTER TABLE <имя_таблицы> ADD (<имя_столбца> <тип_столбца> [NOT NULL] [UNIQUE | PRIMARY KEY] [REFERENCES <имя_мастер_таблицы> [<имя_столбца>]] ,...)**
Удалить столбцы	**ALTER TABLE <имя_таблицы> DROP (<имя_столбца>,...)**
Модификация типа столбцов	**ALTER TABLE <имя_таблицы> MODIFY (<имя_столбца> <тип_столбца> [NOT NULL] [UNIQUE | PRIMARY KEY] [REFERENCES <имя_мастер_таблицы> <имя_столбца>]] ,...)**

Таблица 5 – Операторы работы с индексами

Команда	Описание
Создание индекса	**CREATE [UNIQUE] INDEX <имя_индекса> ON <имя_таблицы> (<имя_столбца>,...) [REFERENCES <имя_мастер_таблицы> [<имя_столбца>]],...)**
Удаление индекса	**DROP INDEX <имя_индекса>**

5.3 DML: Операторы манипулирования данными

Основные операторы манипулирования данными приведены в таблице 6.

Таблица 6 – Команды модификации данных

Добавление записи в таблицу	**INSERT INTO <имя_таблицы> [(<имя_столбца>,<имя_столбца>,...)] VALUES (<значение>,<значение>,..)**
Модификация записи	**UPDATE <имя_таблицы> SET <имя_столбца>=<значение>,... [WHERE <условие>]**
Удаление записи	**DELETE FROM <имя_таблицы> [WHERE <условие>]**

Выборка данных. Для извлечения записей из таблиц в SQL определен оператор *SELECT*. С помощью этой команды осуществляется не только операция реляционной алгебры «выборка» (горизонтальное подмножество), но и предварительное соединение (join) двух и более таблиц. Это наиболее сложное и мощное средство SQL; полный синтаксис оператора *SELECT* имеет вид

SELECT [ALL | DISTINCT] <список_выбора>
FROM <имя_таблицы>, ...
[WHERE <условие>]
[GROUP BY <имя_столбца>,...]
[HAVING <условие>]
[ORDER BY <имя_столбца> [ASC | DESC],...]

Порядок предложений в операторе *SELECT* должен строго соблюдаться (например, *GROUP BY* должно всегда предшествовать *ORDER BY*), иначе это приведет к появлению ошибок.

Как видно, SQL предоставляет все необходимые средства не только для работы со структурой модели данных и отдельных отношений, но для манипулирования хранимой в отношениях информацией. Однако главное назначение языка кроется в другом. С его помощью организуется процесс общения (обмена информацией) между серверной частью (собственно сама СУБД) и клиентской частью (пользовательское приложение). Некоторые вопросы данного взаимодействия будут освещены далее.

6. ОРГАНИЗАЦИЯ ДОСТУПА ПРИКЛАДНОЙ ПРОГРАММЫ К СЕРВЕРУ БАЗЫ ДАННЫХ

6.1 Общие сведения

Как правило, любой поставщик СУБД предоставляет вместе со своей системой внешнюю утилиту, которая позволяет вводить операторы SQL в режиме командной строки и выдает на консоль результаты их выполнения. Недостатки такого режима работы очевидны: необходимо знать SQL, необходимо помнить схему БД, отсутствует возможность удобного просмотра результатов выполнения запросов. Поэтому подобные утилиты стали инструментами администраторов баз данных, а для создания пользовательских приложений используются универсальные и специализированные языки программирования. Приложения, написанные таким образом, позволяют пользователю сосредоточиться на решении собственных задач, а не на структурах данных.

Почти все способы организации взаимодействия пользователя с базой данных основаны на модели «клиент-сервер». Каждое приложение обработки данных разбито, как минимум, на две части:

1. Клиента, который отвечает за организацию пользовательского интерфейса
2. Сервер, который собственно хранит данные, обрабатывает запросы и посылает их результаты клиенту для отображения.

Предположим, что каждая часть приложения функционирует на отдельном компьютере, т.е. к выделенному серверу БД с помощью локальной сети подключены персональные компьютеры пользователей (клиенты).

Язык SQL позволяет только манипулировать данными, но в нем отсутствуют средства создания экранного интерфейса, что необходимо для пользовательских приложений. Для создания этого интерфейса служат универсальные языки третьего поколения (C, C++ и т.д.) или проблемно-ориентированные языки четвертого поколения (xBase, Informix 4Gl, Progress, Jam и т.д.). Все они содержат необходимые операторы ввода/вывода на экран, а также операторы структурного программирования (цикла ветвения и т.д.). Также эти языки допускают определение структур, соответствующих записям таблиц обрабатываемой базы данных. В исходный текст программы включаются операторы языка SQL, во время исполнения передающиеся серверу БД, который собственно и производит манипулирование данными. Отношения, полученные в результате выполнения сервером SQL-запросов, возвращаются прикладной программе, которая заполняет строками этих отношений заранее определенные структуры. Дальнейшая работа клиентской программы (отображение, корректировка записей) ведется с этими структурами.

Рассмотрим различные способы организации доступа прикладной программы к серверу базы данных.

6.2 Использование специализированных библиотек и встраиваемого SQL

Каждая СУБД помимо интерактивной SQL-утилиты обязательно имеет библиотеку доступа и набор драйверов для различных операционных систем. Схема взаимодействия клиентского приложения с сервером базы данных в этом случае приведена на рисунке 2.

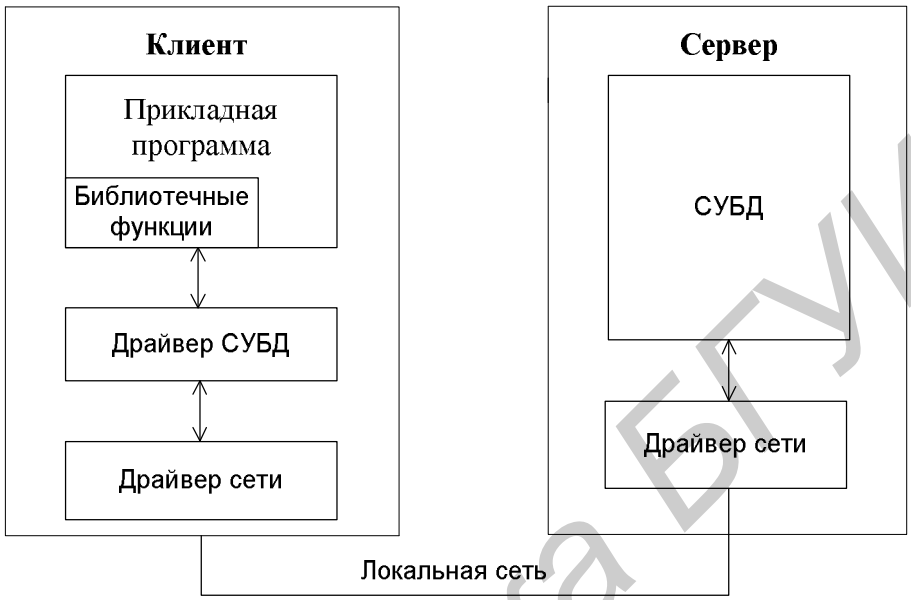

Рисунок 2 – Схема взаимодействия клиентского приложения с сервером базы данных

Библиотека доступа – это, как правило, объектный файл, исходный код которого создан на универсальном языке типа С. Эта библиотека содержит набор функций, позволяющих пользовательскому приложению соединяться с базой данных, передавать запросы серверу и получать ответные данные. Типичный набор функций такой библиотеки (имена функций зависят от используемой библиотеки):

- **DB_connect(char *имя_базы_данных, char *имя_пользователя, char *пароль)** – установить соединение с базой данной. Возвращает указатель на структуру **db**, описывающую характеристики этого соединения;
- **DB_exec(db, char *запрос)** – выполнить запрос к базе данных, определяемой структурой **db**. Применяется для любых запросов, кроме SELECT. Возвращает код выполнения запроса (0 – удачно, либо код ошибки);
- **DB_select(db, char *запрос)** – выполнить запрос на извлечение данных (SELECT). Возвращает структуру **result**, содержащую результаты выполнения запроса (реляционное отношение);
- **DB_fetch(result)** – извлечь следующую запись из структуры **result;**
- **DB_close(db)** – закрыть соединение с базой данных.

Разумеется, это минимальный набор функций для работы с базой данных. Обычно в библиотеке присутствуют также функции, позволяющие определить характеристики структуры **result** (число, порядок и имена столбцов, число строк, номер текущей строки), передвигаться по этой структуре не только вперед, но и назад (**DB_next**, **DB_prev**) и т.д.

Пример программы, использующей библиотеку связи с базой данных:

```
#include <dblib.h>     /* Файл, содержащий описание функций библиотеки          */
.....
/* Организация интерфейса с пользователем, запрос его имени и пароля           */
/* Присвоение значений переменным:  dbname – имя базы данных                    */
/*                          username – имя пользователя      */
/*                          password – пароль                 */
db=DB_connect(dbname,username,password); /* Установление соединения          */
if (db == NULL) {
    error_message();  /* Выдача сообщения об ошибке на монитор пользователя    */
    exit(1);              /* Завершение работы */
}
/* Ожидание запроса пользователя. Формирование строки s_query – запроса        */
/* на выборку данных                                                           */
result=DB_select(db,s_query);/* Пересылка запроса на сервер                    */
if (result==NULL) {
     error_message();             /* Ошибка выполнения запроса. Выдача сообщения */
     exit(2);                     /* Завершение работы                          */
}
/* Вывод результатов запроса на монитор пользователя. Ожидание следующего     */
/* запроса. Подготовка строки u_query="UPDATE ... SET ...", содержащей         */
/* запрос на изменение данных.                                                 */
res=DB_exec(db,u_query);     /* Пересылка запроса на сервер */
if (res != 0 ) {
     error_message();             /* Ошибка выполнения запроса. Выдача сообщения */
     exit(2);                     /* Завершение работы */
}
DB_close(db);                     /*Завершение работы */
```

Данная программа, обеспечивающая взаимодействие пользователя с СУБД, компилируется совместно с библиотекой доступа. Библиотечные вызовы преобразуются драйвером базы данных в сетевые вызовы и передаются сетевым программным обеспечением на сервер.

На сервере происходит обратный процесс преобразования: сетевые пакеты, функции библиотеки, SQL-запросы, обрабатываются, их результаты передаются клиенту.

Такой способ создания приложений чрезвычайно гибок, позволяет реализовать практически любое приложение, но в то же время имеет явные недостатки:

- разработка клиентской программы возможна только для той операционной системы и на том языке программирования, который поддерживается библиотекой;

- необходим драйвер базы данных, который определяет допустимые типы сетевых интерфейсов;
- большой объем кодирования ;
- нестандартизованные библиотечные функции.

В результате получаем приложение, которое привязано как к сетевой среде, так и к программно-аппаратной платформе и используемой базе данных.

Некоторой модификацией данного способа является использование «встроенного» языка SQL. В этом случае в текст программы на языке третьего поколения включаются не вызовы библиотек, а непосредственно предложения SQL, которые предваряются ключевым выражением «EXEC SQL». Перед компиляцией в машинный код такая программа обрабатывается препроцессором, который транслирует смесь операторов «собственного» языка СУБД и операторов SQL в «чистый» исходный код. Затем коды SQL замещаются вызовами соответствующих процедур из библиотек исполняемых модулей, служащих для поддержки конкретного варианта СУБД.

Такой подход позволил несколько снизить степень привязанности к СУБД, например, при переключении прикладной программы на работу с другим сервером базы данных достаточно было заново обработать ее исходный текст новым препроцессором и перекомпилировать.

6.3 CLI – интерфейс уровня вызовов

Большим достижением явилось появление (1994 г.) в стандарте SQL интерфейса уровня вызова – CLI (Call Level Interface), в котором стандартизован общий набор рабочих процедур, обеспечивающий совместимость со всеми основными серверами баз данных. Ключевой элемент CLI – специальная библиотека для компьютера-клиента, в которой хранятся вызовы процедур и большинство часто используемых сетевых компонентов для организации связи с сервером. Это ПО поставляется разработчиком средств SQL, не является универсальным и поддерживает разнообразные транспортные протоколы.

Использование программных вызовов позволяет свести к минимуму операции на компьютере-клиенте. В общем случае клиент формирует оператор языка SQL в виде строки и пересылает ее на сервер посредством процедуры исполнения (execute). Когда же сервер в качестве ответа возвращает несколько строк данных, клиент считывает результат с помощью серии вызовов процедуры выборки данных. Далее информация из столбцов полученной таблицы может быть связана с соответствующими переменными приложения. Вызов специальной процедуры позволяет клиенту определить считанное число строк, столбцов и типы данных в каждом столбце.

Интерфейс CLI построен таким образом, что перед передачей запроса серверу клиент не должен заботиться о типе оператора SQL, будь то выборка, обновление, удаление или вставка.

6.4 ODBC – открытый интерфейс к БД на платформе MS Windows

Очень важный шаг к созданию переносимых приложений обработки данных сделала фирма Microsoft, опубликовавшая в 1992 г. спецификацию ODBC (Open Database Connetcivity – открытого интерфейса к базам данных), предназначенную для унификации доступа к данным с персональных компьютеров, работающих под управлением операционной системы Windows. (Заметим, что ODBC опирается на спецификации CLI.) Структурная схема доступа к данным с использованием ODBC приведена на рисунке 3.

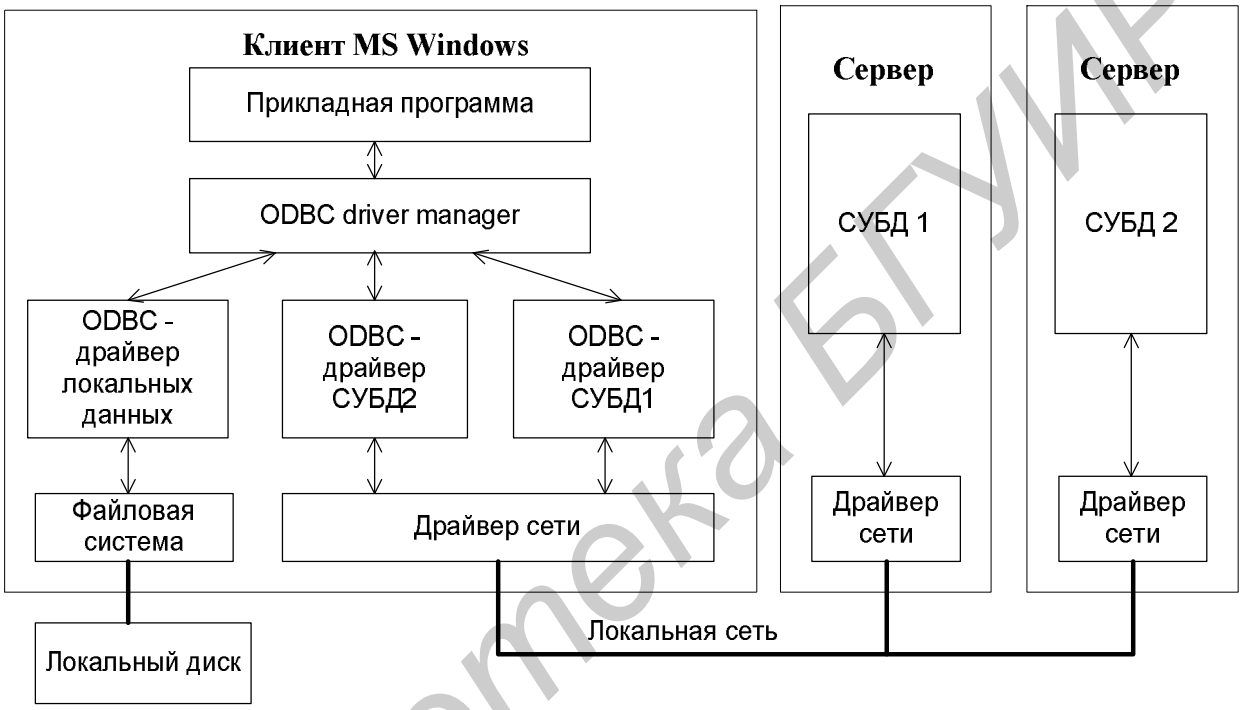

Рисунок 3 – Структурная схема доступа к данным с использованием ODBC

ODBC представляет из себя программный слой, унифицирующий интерфейс приложений с базами данных. За реализацию особенностей доступа к каждой отдельной СУБД отвечает специальный ODBC - драйвер. Пользовательское приложение этих особенностей не видит, т.к. взаимодействует с универсальным программным слоем более высокого уровня. Таким образом, приложение становится в значительной степени не зависимым от СУБД. Однако этот способ также не лишен недостатков:

- приложения становятся привязанными к платформе MS Windows;
- увеличивается время обработки запросов (как следствие введения дополнительного программного слоя);
- необходима предварительная инсталляция ODBC – драйвера и настройка ODBC (указание драйвера, сетевого пути к серверу, базы данных и т.д.) на каждом рабочем месте. Параметры этой настройки являются статическими, т.е. приложение их самостоятельно изменить не может.

6.5 JDBC - мобильный интерфейс к базам данных на платформе Java

JDBC (Java DataBase Connectivity) – это интерфейс прикладного программирования (API) для выполнения SQL-запросов к базам данных из программ, написанных на языке Java. Напомним, что язык Java, созданный компанией Sun, является платформенно-независимым и позволяет создавать как собственно приложения (standalone application), так и программы (апплеты), встраиваемые в web-страницы. Более подробная информация о Java и связанных с ним технологиях находится на серверах java.sun.ru.

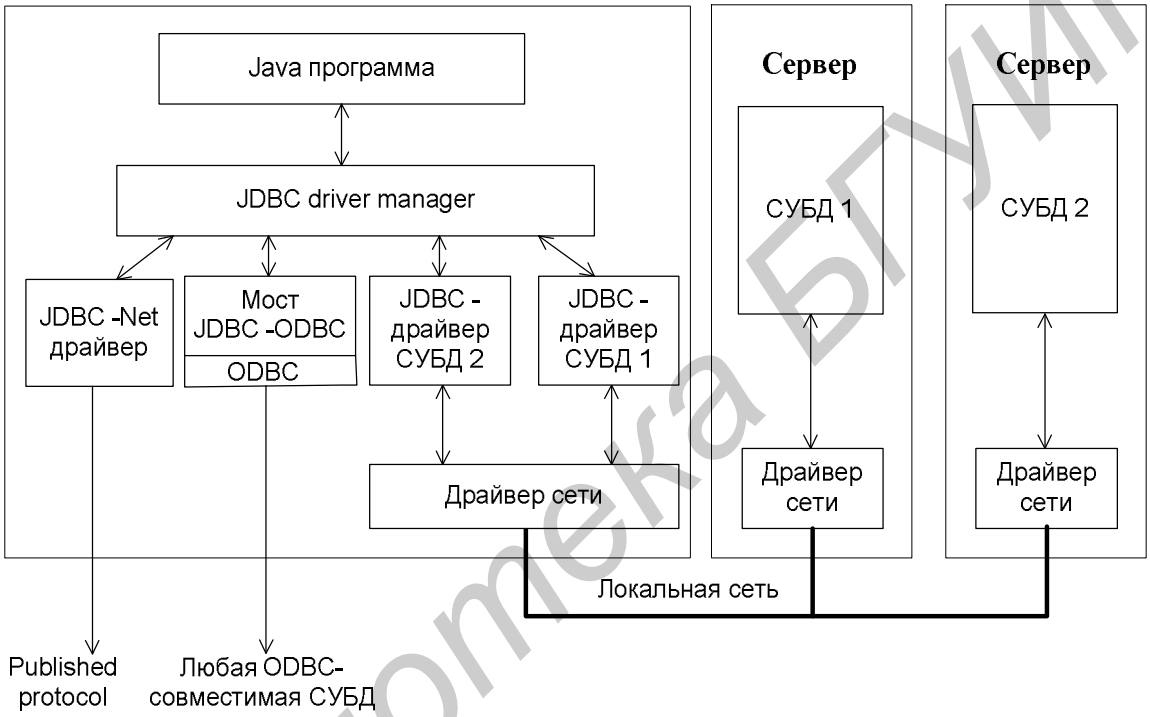

Рисунок 4 – Структурная схема доступа к данным с использованием JDBC

JDBC во многом подобен ODBC (см. рисунок 3),он также построен на основе спецификации CLI, однако имеет ряд замечательных отличий.

Во-первых, приложение загружает JDBC-драйвер динамически, а следовательно, администрирование клиентов упрощается, более того, появляется возможность переключаться на работу с другой СУБД без перенастройки клиентского рабочего места.

Во-вторых, JDBC, как и Java в целом, не привязан к конкретной аппаратной платформе, и, как следствие, проблемы с переносимостью приложений практически снимаются.

В-третьих, использование Java-приложений и связанной с ними идеологии «тонких клиентов» обещает снизить требования к оборудованию клиентских рабочих мест.

7 ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БД

7.1 Постановка задачи проектирования

При проектировании базы данных решаются две основные проблемы:

1. Каким образом отобразить объекты предметной области в абстрактные объекты модели данных так, 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.

2. Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создания 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. Мы ограничимся вопросами логического проектирования, которые существенны при использовании любой реляционной СУБД.

Основная 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.

7.2 Проектирование реляционных баз данных с использованием нормализации

Сначала рассмотрим 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, будем считать, что исходный набор отношений уже соответствует этому требованию.

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

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

Основными свойствами нормальных форм можно назвать следующие**:**

- 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.Y.

Определение 2. Полная функциональная зависимость.

Функциональная зависимость R.X ® 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) R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X (r) R.Z и R.Z (r) R.Y и отсутствует функциональная зависимость R.Z → →R.X. (При отсутствии последнего требования получились 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.

Определение 5. Взаимно независимые атрибуты.

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

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 (r) СОТР_ЗАРП

СОТР_НОМЕР (r) ОТД_НОМЕР

ОТД_НОМЕР (r) СОТР_ЗАРП

СОТР_НОМЕР, ПРО_НОМЕР (r) 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 невозможно 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 не только разрушается связь данного сотрудника с данным проектом, но утрачивается информация о том, что он работает в некотором отделе. При переводе сотрудника в другой отдел необходимо 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 (r) СОТР_ЗАРП

СОТР_НОМЕР (r) ОТД_НОМЕР

ОТД_НОМЕР (r) СОТР_ЗАРП

СОТРУДНИКИ-ПРОЕКТЫ(СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН)

Первичный ключ:

СОТР_НОМЕР, ПРО_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР, ПРО_НОМЕР (r) СОТР_ЗАДАН

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

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

Определение 6.1. *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 не приводятся примеры для отношений с несколькими ключами. Они слишком громоздки и относятся к ситуациям, редко встречающимся на практике.

Третья нормальная форма. Рассмотрим еще раз отношение *СОТРУДНИКИ-ОТДЕЛЫ,* находящееся в 2NF. Заметим, что функциональная зависимость *СОТР_НОМЕР (r) СОТР_ЗАРП* является транзитивной; она является следствием функциональных зависимостей *СОТР_НОМЕР (r) ОТД_НОМЕР и ОТД_НОМЕР (r) 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 не представляется возможным 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, исчезает информация 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 (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 (r) ОТД_НОМЕР

ОТДЕЛЫ (ОТД_НОМЕР, СОТР_ЗАРП)

Первичный ключ:

ОТД_НОМЕР

Функциональные зависимости:

ОТД_НОМЕР (r) 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 7.1. *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. Однако иногда полезно продолжить процесс нормализации.

Нормальная форма Бойса - Кодда. Рассмотрим следующий пример схемы отношения:

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, СОТР_ИМЯ, ПРО_НОМЕР, СОТР_ЗАДАН)

Возможные ключи:

СОТР_НОМЕР, ПРО_НОМЕР

СОТР_ИМЯ, ПРО_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР (r) СОТР_ИМЯ

СОТР_НОМЕР (r) ПРО_НОМЕР

СОТР_ИМЯ (r) СОТР_НОМЕР

СОТР_ИМЯ (r) ПРО_НОМЕР

СОТР_НОМЕР, ПРО_НОМЕР (r) СОТР_ЗАДАН

СОТР_ИМЯ, ПРО_НОМЕР (r) СОТР_ЗАДАН

В этом примере предполагаем, что личность сотрудника полностью определяется как его номером, так и именем.

В соответствии с определением 7.1 отношение *СОТРУДНИКИ-ПРОЕКТЫ* находится в 3NF. Однако тот факт, что имеются функциональные зависимости атрибутов отношения от атрибута, являющегося частью первичного ключа, приводит к аномалиям. Например, для того, чтобы изменить имя сотрудника с данным номером согласованным образом, потребуется модифицировать все кортежи, включающие его номер.

Определение 8. *Детерминант – любой атрибут, от которого полностью функционально зависит некоторый другой атрибут*.

Определение 9. Нормальная форма Бойса - Кодда. *Отношение R находится в нормальной форме Бойса - Кодда (BCNF) в том и только в том случае, если каждый детерминант является возможным ключом.*

Очевидно, что это требование не выполнено для отношения *СОТРУДНИКИ-ПРОЕКТЫ*. Можно произвести его декомпозицию к отношениям *СОТРУДНИКИ и СОТРУДНИКИ-ПРОЕКТЫ*:

СОТРУДНИКИ (СОТР_НОМЕР, СОТР_ИМЯ)

Возможные ключи:

СОТР_НОМЕР

СОТР_ИМЯ

Функциональные зависимости:

СОТР_НОМЕР (r) *СОТР_ИМЯ*

СОТР_ИМЯ (r) *СОТР_НОМЕР*

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР, СОТР_ЗАДАН

Возможный ключ:

СОТР_НОМЕР, ПРО_НОМЕР

Функциональные зависимости:

СОТР_НОМЕР, ПРО_НОМЕР (r) СОТР_ЗАДАН

Возможна альтернативная декомпозиция, если выбрать за основу *СОТР_ИМЯ*. В обоих случаях получаемые отношения *СОТРУДНИКИ* и *СОТРУДНИКИ-ПРОЕКТЫ* находятся в BCNF, и им не свойственны отмеченные аномалии.

Четвертая нормальная форма. Рассмотрим пример следующей схемы отношения:

ПРОЕКТЫ (ПРО_НОМЕР, ПРО_СОТР, ПРО_ЗАДАН)

Отношение *ПРОЕКТЫ* содержит номера проектов, для каждого проекта список сотрудников, которые могут выполнять проект, и список заданий, предусматриваемых проектом. Сотрудники могут участвовать в нескольких проектах, и разные проекты могут включать одинаковые задания.

Каждый кортеж отношения связывает некоторый проект с сотрудником, участвующим в этом проекте, и заданием, который сотрудник выполняет в рамках данного проекта (предполагаем, что любой сотрудник, участвующий в проекте, выполняет все задания, предусмотренные этим проектом). По причине сформулированных выше условий единственным возможным ключем отношения является составной атрибут *ПРО_НОМЕР*, *ПРО_СОТР*, *ПРО_ЗАДАН*, и нет никаких других детерминантов. Следовательно, отношение *ПРОЕКТЫ* находится в BCNF. Но при этом оно обладает недостатками: если, например, некоторый сотрудник присоединяется к данному проекту, необходимо вставить в отношение *ПРОЕКТЫ* столько кортежей, сколько заданий в нем предусмотрено.

Определение 10. Многозначные зависимости. *В отношении R (A, B, C) существует многозначная зависимость R.A (r) (r) R.B в том и только в том случае, если множество значений B, соответ-ствующее паре значений A и C, зависит только от A и не зависит от C.*

В отношении *ПРОЕКТЫ* существуют следующие две многозначные зависимости:

ПРО_НОМЕР (r) (r) ПРО_СОТР
ПРО_НОМЕР (r) (r) ПРО_ЗАДАН

Легко показать, что в общем случае в отношении R (A, B, C) существует многозначная зависимость R.A (r) (r) R.B в том и только в том случае, когда существует многозначная зависимость R.A (r) (r) R.C.

Дальнейшая нормализация отношений, подобных отношению *ПРОЕКТЫ*, основывается на следующей теореме.

Теорема Фейджина: Отношение R (A, B, C) можно спроецировать без потерь в отношения R1 (A, B) и R2 (A, C) в том и только в том случае, когда существует MVD A (r) (r) B | C.

Под проецированием без потерь понимается такой способ декомпозиции отношения, при котором исходное отношение полностью и без избыточности восстанавливается путем естественного соединения полученных отношений.

Определение 11. Четвертая нормальная форма. *Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости A (r) (r) B все остальные атрибуты R функционально зависят от A.*

В нашем примере можно произвести декомпозицию отношения *ПРОЕКТЫ* в два отношения *ПРОЕКТЫ-СОТРУДНИКИ* и *ПРОЕКТЫ-ЗАДАНИЯ*:

ПРОЕКТЫ-СОТРУДНИКИ (ПРО_НОМЕР, ПРО_СОТР)
ПРОЕКТЫ-ЗАДАНИЯ (ПРО_НОМЕР, ПРО_ЗАДАН)

Оба эти отношения находятся в 4NF и свободны от отмеченных аномалий.

Пятая нормальная форма. Во всех рассмотренных до этого момента нормализациях производилась декомпозиция одного отношения в два. Иногда это сделать не удается, но возможна декомпозиция в большее число отношений, каждое из которых обладает лучшими свойствами.

Рассмотрим, например, отношение *СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ (СОТР_НОМЕР, ОТД_НОМЕР, ПРО_НОМЕР).*

Предположим, что один и тот же сотрудник может работать в нескольких отделах и работать в каждом отделе над несколькими проектами. Первичным ключем этого отношения является полная совокупность его атрибутов, отсутствуют функциональные и многозначные зависимости.

Поэтому отношение находится в 4NF. Однако в нем могут существовать аномалии, которые можно устранить путем декомпозиции в три отношения.

Определение 12. Зависимость соединения. *Отношение R (X, Y, ..., Z) удовлетворяет зависимости соединения * (X, Y, ..., Z) в том и только в том случае, когда R восстанавливается без потерь путем соединения своих проекций на X, Y, ..., Z.*

Определение 13. Пятая нормальная форма. *Отношение R находится в пятой нормальной форме (нормальной форме проекции-соединения - PJ/NF) в том и только в том случае, когда любая зависимость соединения в R следует из существования некоторого возможного ключа в R.*

Введем следующие имена составных атрибутов:

СО = {СОТР_НОМЕР, ОТД_НОМЕР}

СП = {СОТР_НОМЕР, ПРО_НОМЕР}

ОП = {ОТД_НОМЕР, ПРО_НОМЕР}

Предположим, что в отношении *СОТРУДНИКИ-ОТДЕЛЫ-ПРОЕКТЫ* существует зависимость соединения:

** (СО, СП, ОП)*

На примерах легко показать, что при вставках и удалениях кортежей могут возникнуть проблемы. Их можно устранить путем декомпозии исходного отношения в три новых отношения:

СОТРУДНИКИ-ОТДЕЛЫ (СОТР_НОМЕР, ОТД_НОМЕР)

СОТРУДНИКИ-ПРОЕКТЫ (СОТР_НОМЕР, ПРО_НОМЕР)

ОТДЕЛЫ-ПРОЕКТЫ (ОТД_НОМЕР, ПРО_НОМЕР)

Пятая нормальная форма – это последняя нормальная форма, которую можно получить путем декомпозиции. Ее условия достаточно нетривиальны, и на практике 5NF не используется. Заметим, что зависимость соединения является обобщением как многозначной, так и функциональной зависимости.

7.3 Семантическое моделирование данных, ER-диаграммы

Предметная область – часть реального мира, подлежащая изучению с целью организации управления и в конечном счете автоматизации**.** Предметная область представляется множеством фрагментов, например, предприятие – цехами, дирекцией, бухгалтерией и т.д. Каждый фрагмент предметной области характеризуется множеством объектов и процессов, использующих объекты, а также множеством пользователей, характеризующихся различными взглядами на предметную область.

Данное определение прежде всего свидетельствует о том, что, приступая к созданию системы автоматизированной обработки информации, разработчик должен сформировать понятия о предметах, фактах и событиях, которыми будет оперировать данная система. Проектирование предметной области 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 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 опи-

сать 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.

Таким образом, хотя процесс проектирования и начинается с выделения некоторых существенных для приложения объектов предметной области («сущностей») и выявления связей между этими сущностями, в итоге реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

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

В терминах семантической модели проектируется 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. Первый подход основан на явном представлении концептуальной схемы как исходной информации для компилятора. Второй предполагает построение 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 мало известны системы, обеспечивающие более высокий уровень нормализации).

Наконец, есть и 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, основывается 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).

Основные понятия ER-модели.

Приведем основные определения терминов, используемых для ER(Entity-Relationship) – модели или, как еще ее называют, модели «сущность-связь»:

Сущность (entity) – это объект, который может быть идентифицирован неким способом, отличающим его от других объектов. Примеры: конкретный человек, предприятие, событие и т.д.

Набор сущностей (entity set) − множество сущностей одного типа (обладающих одинаковыми свойствами). Примеры: все люди, предприятия, праздники и т.д. Наборы сущностей необязательно должны быть непересекающимися. Например, сущность, принадлежащая к набору *МУЖЧИНЫ*, также принадлежит набору *ЛЮДИ*.

Связь (relationship) – это ассоциация, установленная между несколькими сущностями. Например:

- поскольку каждый сотрудник работает в каком-либо отделе, между сущностями *СОТРУДНИК* и *ОТДЕЛ* существует связь «работает в …» или *ОТДЕЛ−РАБОТНИК*;
- так как один из работников отдела является его руководителем, то между сущностями *СОТРУДНИК* и *ОТДЕЛ* имеется связь «руководит …» или *ОТДЕЛ−РУКОВОДИТЕЛЬ*;
- могут существовать и связи между сущностями одного типа, например связь *РОДИТЕЛЬ – ПОТОМОК* между двумя сущностями *ЧЕЛОВЕК*.

Связь также может иметь атрибуты. Например, для связи *ОТДЕЛ −РАБОТНИК* можно задать атрибут *СТАЖ_РАБОТЫ_В_ОТДЕЛЕ*.

Очень важным свойством модели «сущность-связь» является то, что она может быть представлена в виде графической схемы. Это значительно облегчает анализ предметной области. Рассмотрим в качестве примера использование нотации Чена - Мартина для создания семантической модели, отображающей связь на предприятии сотрудников, отделов и должностей. В таблице 8 приведен список принятых условных обозначений.

Атрибуты с сущностями и сущности со связями соединяются прямыми линиями. То число сущностей, которое может быть ассоциировано через набор связей с другой сущностью, называют степенью связи. Существуют следующие степени бинарных связей:

- один к одному (обозначается 1 : 1). Это означает, что в такой связи сущности с одной ролью всегда соответствуют не более одной сущности с другой ролью. В рассматриваемом примере эта связь «руководит», поскольку в каждом отделе может быть только один начальник, а сотрудник может руководить только в одном отделе. Так как степень связи для каждой сущности равна 1, то они соединяются одной линией.

Таблица 8 – Условные обозначения нотации Чена-Мартина для построения ER-модели

Обозначение	Значение
Имя сущности	Набор независимых сущностей
Имя сущности	Набор зависимых сущностей
Имя атрибута	Атрибут
Имя атрибута	Ключевой атрибут
Имя связи	Набор связей

Важной характеристикой является класс принадлежности входящих в нее сущностей, или **кардинальность связи**. Так как в каждом отделе обязательно должен быть руководитель, то каждой сущности *ОТДЕЛ* непременно должна соответствовать сущность *СОТРУДНИК*. Однако не каждый сотрудник является руководителем отдела, следовательно, в данной связи не каждая сущность *СОТРУДНИК* имеет ассоциированную с ней сущность *ОТДЕЛ* (рисунок 5).

Рисунок 5 – Пример связи «один к одному»

Таким образом, говорят, что сущность *СОТРУДНИК* имеет обязательный класс принадлежности – ⊢, а сущность *ОТДЕЛ* имеет необязательный класс принадлежности – ○—;

- много к одному (n : 1). Предположим, что предприятие строит свою деятельность на основании контрактов, заключаемых с заказчиками. Этот факт отображается в модели «сущность-связь» с помощью связи *КОНТРАКТ-ЗАКАЗЧИК*, объединяющей сущности *КОНТРАКТ (НОМЕР, СРОК ИСПОЛНЕНИЯ, СУММА)* и *ЗАКАЗЧИК (НАИМЕНОВАНИЕ, АДРЕС)*. Так как с одним заказчиком может быть заключено более одного контракта, то связь *КОНТРАКТ-ЗАКАЗЧИК* между этими сущностями имеет степень n : 1 (рисунок 6);

Рисунок 6. – Пример связи «много к одному»

- много ко многим (n : n). В этом случае каждая из ассоциированных сущностей может быть представлена любым количеством экземпляров. Пусть на рассматриваемом предприятии для выполнения каждого контракта создается рабочая группа, в которую входят сотрудники разных отделов. Поскольку каждый сотрудник может входить в несколько (или ни в одну из) рабочих групп, а каждая группа должна включать не менее одного сотрудника, то связь между сущностями *СОТРУДНИК* и *РАБОЧАЯ ГРУППА* имеет степень n : n (рисунок 7).

Рисунок 7 – Пример связи «много ко многим»

Сотрудник может иметь более чем одну должность (работать более чем в одном отделе), причем может занимать неполную ставку. В то же время одну и ту же должность могут занимать одновременно несколько сотрудников. В результате необходимо ввести наборы сущностей *СОТРУДНИК, ОТДЕЛ, ДОЛЖНОСТЬ* с их атрибутами и набор связей «работает в». Также для связи «работает в» можно установить атрибут *СТАВКА* (рисунок 8). Тогда связь «работает в» будет являться тринарной (связывать три сущности). При реализации в большинстве СУБД, например в MicrosoftAccess, требуется, чтобы присутствовали только бинарные связи. Кроме того, атрибуты могут принадлежать только сущностям. Поэтому вводят дополнительные сущности, в рассматриваемом примере – *ШТАТНАЯ ЕДИНИЦА* с атрибутом *СТАВКА*. В результате получим семантическую модель, представленную на рисунке 8.

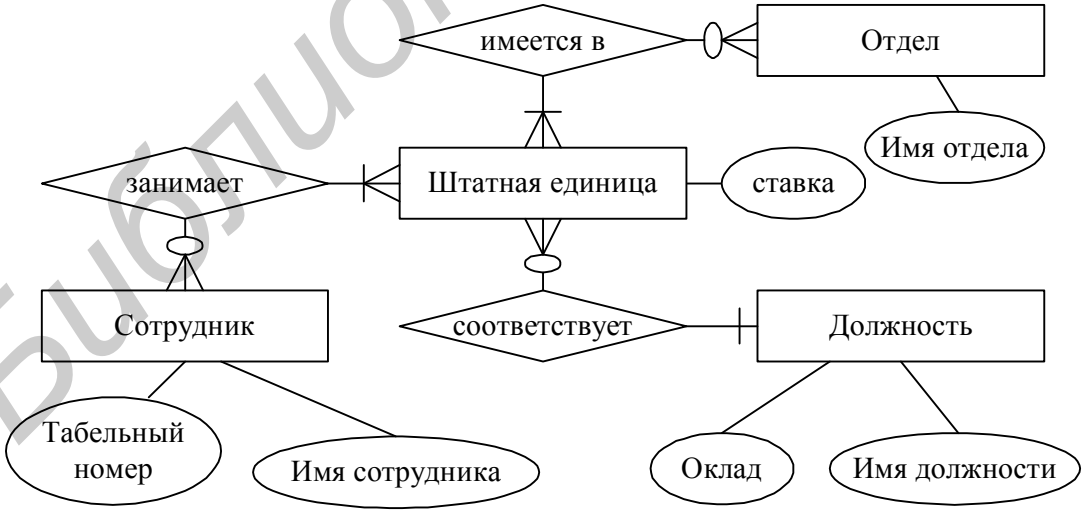

Рисунок 8 – Пример диаграммы «сущность-связь»

Представленную семантическую модель достаточно легко преобразовать в реляционную, используя принятые для выбранной семантики правила. Однако наилучшего результата можно достичь, если разрабатывать предметную область на основе совместного использования рассмотренных методов, т.е. начинать проектирование БД с создания семантической модели, а затем производить проверку нормализации отношений. Необходимо отметить, что только концептуальная модель наиболее полно описывает предметную область, поэтому ее поддержание и сопровождение осуществляется не только на последующих этапах проектирования, но и в период эксплуатации готового программного продукта.

ЛИТЕРАТУРА

1. Бойко В.В., Савинков В.М. Проектирование баз данных информационных систем. – М.: Финансы и статистика, 1989. – 351 с.
2. Боуман Д., Эмерсон С., Дарновски М. Практическое руководство по SQL. – Киев: Диалектика, 1997.
3. Васкевич Д. Стратегии клиент/сервер. – Киев: Диалектика, 1997.
4. Кузнецов С.Д. Введение в системы управления базами данных //СУБД. – 1995. № 1,2,3,4, 1996.
5. Кузнецов С.Д. Стандарты языка реляционных баз данных SQL: краткий обзор //СУБД. – 1996. № 2. – с. 6–36.
6. Дейт К. Руководство по реляционной СУБД DB2. – М.: Финансы и статистика, 1988. – 320 с.
7. Дейт К. Введение в системы баз данных. 6-изд. – Киев: Диалектика, 1998. – 784 с.
8. Джексон Г. Проектирование реляционных баз данных для использования с микроЭВМ. – М.: Мир, 1991. – 252 с.
9. Кириллов В.В. Основы проектирования реляционных баз данных: Учеб. пособие. – СПб.: ИТМО, 1994. – 90 с.
10. Кириллов В.В. Структуризованный язык запросов (SQL). – СПб.: ИТМО, 1994. – 80 с.
11. Мейер М. Теория реляционных баз данных. – М.: Мир. 1987. – 608 с.

Св. план 2005, поз. 33

Учебное издание

Ганьшин Дмитрий Алексеевич,
Антипова Марина Александровна

ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ

Конспект лекций

для студентов специальностей
I-53 01 03 «Автоматическое управление в технических системах»
и I-53 01 07 «Информационные технологии и управление
в технических системах»
всех форм обучения

Редактор Т.П. Андрейченко
Корректор Н.В. Гриневич

Подписано в печать 2.03.2006. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л. 3,72.
Уч.-изд. л. 3,7. Тираж 250 экз. Заказ 235.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
Лицензия на осуществление издательской деятельности №02330/0056964 от 01.04.2004.
Лицензия на осуществление полиграфической деятельности №02330/0131518 от 30.04.2004.
220013, Минск, П. Бровки, 6