Министерство образования и науки Российской Федерации

Калужский филиал
федерального государственного бюджетного образовательного
учреждения высшего образования
**«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»**
(КФ МГТУ им. Н.Э. Баумана)

С.А. Глебов

БАЗЫ ДАННЫХ

Конспект лекций

Калуга – 2018

УДК 004.65
ББК 32.972.134
Г53

Конспект лекций составлен в соответствии с учебным планом КФ МГТУ им. Н.Э. Баумана по направлению подготовки 09.03.04 «Программная инженерия» кафедры «Программного обеспечения ЭВМ, информационных технологий и прикладной математики».

Конспект лекций рассмотрен и одобрен:

- Кафедрой «Программного обеспечения ЭВМ, информационных технологий и прикладной математики» (ФН1-КФ) протокол № 8 от «21» марта 2018 г.

И.о. зав. кафедрой ФН1-КФ _____к.т.н., доцент Ю.Е. Гагарин

- Методической комиссией факультета ФНК протокол №___ от «___» _____ 2018 г.

Председатель методической
комиссии факультета ФНК _____к.х.н., доцент К.Л. Анфилов

- Методической комиссией
КФ МГТУ им.Н.Э. Баумана протокол №___ от «___» _____ 2018 г.

Председатель методической комиссии
КФ МГТУ им.Н.Э. Баумана
_____д.э.н., профессор О.Л. Перерва

Рецензент:
к.ф.-м.н., директор по исследованиям и развитию ООО "НПФ "Эверест" _____ В.Ю. Кириллов

Авторы
к.ф.-м.н., доцент кафедры ФН1-КФ _____ С.А. Глебов

Аннотация

Конспект лекций включает сведения о подходах к организации баз данных, реляционных моделях баз данных, нормализации баз данных, языке SQL, а также технологиях физического хранения и доступа к данным.

Предназначены для студентов 3-го курса бакалавриата КФ МГТУ им. Н.Э. Баумана, обучающихся по направлению подготовки 09.03.04 «Программная инженерия».

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящий курс лекций составлен в соответствии с программой лекционных занятий по курсу «Базы данных» на кафедре «Программное обеспечение ЭВМ, информационные технологии и прикладная математика» факультета фундаментальных наук Калужского филиала МГТУ им. Н.Э. Баумана.

Курс лекций, ориентирован на студентов 3-го курса направления подготовки 09.03.04 «Программная инженерия», содержит сведения о подходах к организации баз данных, реляционных моделях баз данных, нормализации баз данных, языке SQL, а также технологиях физического хранения и доступа к данным.

Лекция 1. РАННИЕ ПОДХОДЫ К ОРГАНИЗАЦИИ БД

Файловые системы

Предшественниками СУБД были файловые системы. Они были первой попыткой компьютеризировать ручные картотеки.

Пусть требуется хранить информацию о торговых сделках: дата, сумма сделки, клиент, менеджер, заключивший сделку. Требуется подсчитывать, например, сумму комиссионных менеджеру, составляющих 2% от суммы.

Дата	Сумма	Клиент	Менеджер
15.03.99	100	ОАО "Звук"	Семенов А.Н.
17.03.99	150	ЧП "Петренко"	Нестеров П.К.
20.04.99	190	ОАО "Цвет"	Семенов А.Н.
24.04.99	120	ЗАО "КНП"	Уткина С.М.
25.05.99	130	ЧП "Васин"	Семенов А.Н.

Создается структура типа запись:

```
sale = record
   date:    string[8];
   summa:   longint;
   client:  string[30];
   manager: string[25];
end;
```

Объявляется файл типа sale.

```
f: file of sale;
```

Связав, файловую переменную f с файлом на диске, получим файл данных.

```
Assign (f, 'data.dat');
Rewrite (f);
```

Размер каждой записи (компонента файла) = 9+4+31+26=70 байт. Соответственно, размер файла = кол-во записей × 70.

Не зная структуры записи, из файла data.dat нельзя (достаточно сложно) получить хранимые в нем данных, поскольку неизвестно

сколько байт приходится на каждую запись, сколько полей и каков их размер внутри записи и т.д. Информацию об этом хранится исключительно в программе.

Предположим, что требуется добавить новое поле или удалить лишнее, или, например, использовать дату не в формате dd.mm.yy, а в формате dd.mm.yyyy. Это значит, что каждый компонент файла должен быть изменен на какое-то количество байт с соответвующим сдвигом последующих записей.

Просто переопределить файл data.dat невозможно, потому необходимо выполнить следующие процедуры:

- создать временный файл f1 с новой структурой;
- каждую запись из f изменить под новую структуру и записать в f1;
- удалить f;
- переименовать f1 в f.

Задача подсчета комиссионных достаточно проста: нужно перебрать все записи, выбирая нужные. В случае, если количество сделок достигает десятков тысяч, то это просто потребует больше времени как для человека, так и для компьютера.

Если этот же список сделок будет упорядочен по столбцу Менеджер, то процедуру можно значительно ускорить: найти первую запись для Менеджер = Семенов А.Н. и перебрать все записи подряд, пока значение Менеджер не изменится. Но для этого нужно пересортировать таблицу по полю Менеджер, а после подсчета — по полю Дата. А если еще нужна статистика по клиентам, то требуется еще сортировка и т.п., что значительно замедлит работу.

Достаточно реально может иметь место ситуация, когда в фирме может оказаться два сотрудника с одинаковыми ФИО (Семенов А.Н.). Как в таком случае разделить заключенные ими сделки? Однофамильцев необходимо как-то отличать. Например, по дате рождения.

Дата	Сумма	Клиент	Менеджер	д.р.
15.03.1999	100	ОАО "Звук"	Семенов А.Н.	03.10.1975
17.03.1999	150	ЧП "Петренко"	Нестеров П.К.	12.05.1970
20.04.1999	190	ОАО "Цвет"	Семенов А.Н.	03.10.1975
24.04.1999	120	ЗАО "КНП"	Уткина С.М.	31.12.1968
25.05.1999	130	ЧП "Васин"	Семенов А.Н.	28.02.1974

В этом случае, увеличивается т.н. избыточность таблицы и возрастает вероятность ошибки. Приходится хранить в каждой записи повторяющиеся ФИО и д.р. менеджера. Кроме того, чем больше информации приходится вводить тем вероятнее можно ошибиться. А для программы ошибка хотя бы в одном символе — и это будет новый менеджер, который получит свои комиссионные.

В этом случае поступают так: менеджеров выносят в отдельную таблицу и назначают каждому персональный номер (идентификатор):

ид	фио	д.р.
1	Нестеров П.К.	12.05.70
2	Семенов А.Н.	28.02.74
3	Семенов А.Н.	03.10.75
4	Уткина С.М.	31.12.68

И тогда таблица сделок будет выглядеть так:

дата	сумма	клиент	ид_менеджера
15.03.2003	100	ОАО "Звук"	3
17.03.2003	150	ЧП "Петренко"	1
20.04.2003	190	ОАО "Цвет"	3
24.04.2003	120	ЗАО "КНП"	4
25.05.2003	130	ЧП "Васин"	2

Эта процедура носит название нормализации. А имеющиеся две таб-лицы получаются связанными. При этом таблица СДЕЛКИ называется до-черней (подчиненной), т.к. зависит от таблицы

МЕНЕДЖЕРЫ, которая в свою очередь называется родительской (главной). В таблице МЕНЕДЖЕРЫ, поле ИД является первичным ключом, а в таблице СДЕЛКИ, поле ИД_МЕНЕДЖЕРА является вторичным ключом.

Другое преимущество: если, например, г-жа Уткина С.М. поменяет фамилию, то это никак не скажется на ее идентификаторе. И т.о. не при-дется менять записи в таблице сделок.

Может иметь место ситуация, когда БД будет содержать некорректные данные, т.е. информацию, которая никоим образом не отражает реальную ситуацию. В этом случае говорят о нарушении целостности БД, которое бывает двух типов:

- нарушение целостности данных. Например, отрицательная сумма сделки, дата сделки = 30.02.2003, клиент = "293234".
- нарушение ссылочной целостности. Например, в таблицу СДЕЛКИ будет введен идентификатор менеджера=9, отсутствующий в таблице МЕНЕДЖЕРЫ.

Далее оператор, который фиксирует сделки в файле, не должен, напрмер, иметь доступа к списку персонала, что в файловых системах сделать достаточно сложно.

Созданное приложение подсчитывает только комиссионные менеджеров. В случае появления необходимости в новых запросах к файлу данных нужно модифицировать старую или писать новую программу. Нет возможности выполнять любые, заранее неизвестные, динамические запросы из существующего приложения, не изменяя его.

Может понадобится написать новую программу, использующую тот же файл данных, но на другом языке программирования, в котором внутреннее представление типов данных существенно отличается или не существует подходящего типа данных. Для одного и того же типа отводится зачастую разное количество единиц памяти для хранения.

Исходя из вышеперечисленных проблем, появляются прикладные программы, которые в той или иной степени решали эти вопросы. Они получили название системы управления базами данных — СУБД.

Файловые системы	СУБД
Структура данных жестко зафиксирована в приложении	Описание структуры данных хранится в месте с самими данными
Изменить структуру данных нельзя	Специальные языки определения данных позволяют менять их структуру без каких-либо дополнительных действий
Проблема быстрой сортировки и поиска	Индексация обеспечивает быструю сортировку и поиск
Поддержка целостности целиком лежит на программисте	Средства контроля целостности БД
Разделение данных	Система безопасности позволяет ограничить доступ к конфиденциальным д-м
Фиксированные запросы	Специальные языки манипуляции д-ми позволяют выполнять динамические запросы, т.е. такие которые не планировались при создании приложения
Несовместимость форматов представления данных	Стандартизация типов и форматов д-х.

Все перечисленные ограничения файловых систем являются следствием двух факторов:

1. Определение данных хранится внутри приложений, а не содержится отдельно от них

2. Помимо приложений не предусмотрено никаких других инструментов доступа к данным и их обработки

Т.о. база данных это совместно импользуемый набор логически свя-занных данных и описание этих данных, предназначенный для удовлетво-рения информационных потребностей.

Дореляционные системы

В дореляционных системах связи между таблицами были основаны на физических указателях. Т.е. запись, которая является связанной с записью из другого файла содержит ее явный физический адрес. Очевидно, что такой подход весьма затрудняет операции над данными.

Кроме того, выбрать данные по связям, которые не были учтены при проектировании БД не представлялось возможным.

Навигационная природа ранних систем и доступ к данным на уровне записей заставляли пользователя самого производить всю оптимизацию доступа к БД, без какой-либо поддержки системы.

После появления реляционных систем большинство ранних систем было оснащено "реляционными" интерфейсами. Однако в большинстве случаев это не сделало их по-настоящему реляционными системами, поскольку оставалась возможность манипулировать данными в естественном для них режиме.

Иерархическая модель данных

Иерархическая БД состоит из упорядоченного набора деревьев.

Дерево состоит из одной «корневой» записи и упорядоченного набора из нуля или более поддеревьев. Тип дерева в целом представляет собой иерархически организованный набор типов записи. Каждый тип записи потомка может иметь только один тип записи предка.

Пример. Организация осуществляет продажу товаров (нас интересует только выписка счетов) в магазины клиентов, через конкретного менеджера (рис. 1.1).

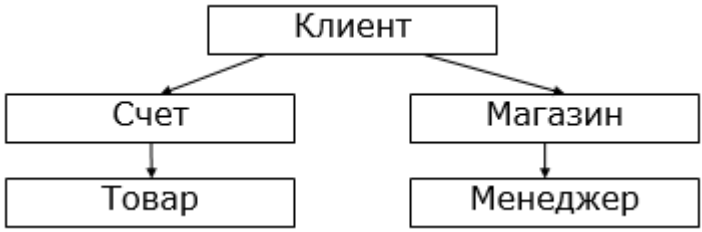


Рис. 1.1. Дерево организации

Экземпляр дерева выглядит следующим образом:

100; ООО "Звук"						
0001	29.12.2002				522	Строймат Труда-45
1	Плита	20	2000		901	Сидоров Н.У.
2	Гвозди	200	44		903	Нестеров П.К.
0003	11.01.2003				657	Алладин Мира-3
1	Песок	23	32		904	Семенов А.Н.
2	Щебень	32	23		906	Козлова Г.В.

Аналогией иерархических систем (как и любых древовидных) является картотека. В шкафу каждому клиенту нашей фирмы соответствует ящик, в нем два отделения: для счетов, выписанных клиенту (каждый счет - листок со списком выписанных товаров) и для магазинов этого клиента (каждый магазин — листок со списком его менеджеров).

Понятно, что не все связи можно представить в виде иерархии. При помощи иерархической модели нельзя моделировать связи "многие-ко-многим".

Другой пример типа дерева (схемы иерархической БД) представлен на рис. 1.2:

Рис. 1.2.Дерево структуры организации

Здесь ОТДЕЛ является предком для НАЧАЛЬНИК и СОТРУДНИКИ, а НАЧАЛЬНИК и СОТРУДНИКИ — потомки ОТДЕЛ. Между типами записи поддерживаются связи.

База данных с такой схемой могла бы выглядеть следующим образом (показан один экземпляр дерева):

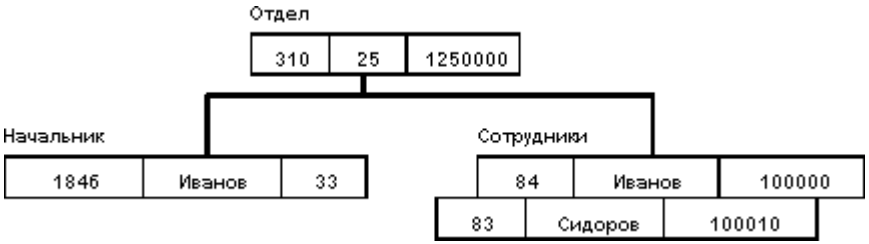


Рис. 1.3. Дерево базы данных

Все экземпляры данного типа потомка с общим экземпляром типа предка называются близнецами. Для БД определен полный порядок обхода — сверху-вниз, слева-направо.

Примерами типичных операторов манипулирования иерархически организованными данными могут быть следующие:

- Найти указанное дерево БД (например, клиент [100; ООО "Звук"]);
- Перейти от одного дерева к другому;
- Перейти от одной записи к другой внутри дерева (например, от отдела — к первому сотруднику);
- Перейти от одной записи к другой в порядке обхода иерархии;
- Вставить новую запись в указанную позицию или удалить текущую.

Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя.

Аналогичное поддержание целостности по ссылкам между записями, не входящими в одну иерархию, не поддерживается (например, в счете указать менеджера, который его выписал).

Сетевая модель данных

Сетевой подход к организации данных является расширением иерархического. В иерархических структурах запись-потомок должна

иметь в точности одного предка; в сетевой структуре данных потомок может иметь любое число предков.

Т.е. не все отношения можно представить в виде иерархии.

Например, сеть магазинов продает товары, через менеджеров, которые выписывают счета на товары. При этом список контрагентов этой сети включает как поставщиков, так и потребителей товара (рис. 1.4).

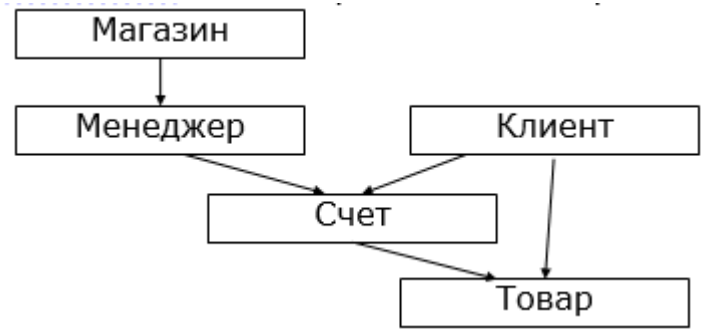


Рис. 1.4. Структура организации

Тип связи определяется для двух типов записи: предка и потомка. Экземпляр типа связи состоит из одного экземпляра типа записи предка и упорядоченного набора экземпляров типа записи потомка.

Типы связей в нашем примере: магазин — менеджер; менеджер — счет; счет — товар; клиент — счет; клиент — товар.

Экземпляры типа связи магазин — менеджер:

Экземпляры типа предка МАГАЗИН:	Набор экземпляров типа потомка МЕНЕДЖЕР:
Канцтовары	Иванов, Петров, Сидоров, Семенов
Охотник	Зайцев, Волков, Медведев, Гусев
Бакалея	Плюшкин, Блинов, Баранкин

Для данного типа связи L с типом записи предка P и типом записи по-томка C должны выполняться следующие два условия:

– Каждый экземпляр типа P является предком только в одном экземпляре L. Т.е. не может быть еще оного экземпляра магазина "Охотник" с своим штатом.
– Каждый экземпляр C является потомком не более, чем в одном экземпляре L. Т.е. менеджер не может работать более чем в одном магазине.

На формирование типов связи не накладываются особые ограничения; возможны, например, следующие ситуации:

– Тип записи потомка в одном типе связи L1 может быть типом записи предка в другом типе связи L2 (как в иерархии).
– Данный тип записи P может быть типом записи предка в любом числе типов связи.
– Данный тип записи P может быть типом записи потомка в любом числе типов связи.
– Может существовать любое число типов связи с одним и тем же типом записи предка и одним и тем же типом записи потомка; и если L1 и L2 — два типа связи с одним и тем же типом записи предка P и одним и тем же типом записи потомка C, то правила, по которым образуется родство, в разных связях могут различаться.
– Типы записи X и Y могут быть предком и потомком в одной связи и потомком, и предком — в другой.
– Предок и потомок могут быть одного типа записи.

Примерный набор операций может быть следующим:

– Найти конкретную запись в наборе однотипных записей (менеджера Петрова);
– Перейти от предка к первому потомку по некоторой связи (к первому менеджеру магазина "Бакалея");
– Перейти к следующему потомку в некоторой связи (от Петрова к Сидорову);
– Перейти от потомка к предку по некоторой связи (найти кто выписал счет № 388);
– Создать, уничтожить, модифицировать запись;
– Включить в связь, исключить из связи, переставить в др. связь и т.д.

Лекция 2. РЕЛЯЦИОННАЯ МОДЕЛЬ ДАННЫХ

Основные понятия

Реляционный подход, являющий сегодня доминирующим, начался со статьи Э.Кодда в 1970 г. Первые коммерческие СУБД, использующие реляционную модель появились только в начале 80-х.

Существовавшие ранее модели дореляционных СУБД, не сразу были вытеснены с рынка по причине огромных накопленных объемов информации.

В реляционной модели данных, основанной на логических отношениях, пользователь совершенно не заботится о физической структуре данных и не интересуется ею.

Реляционная модель основана на математическом понятии отношения, физическим представлением которого является двумерная таблица, состоящая из строк и столбцов. В этой модели широко используются математическая терминология из теории множеств.

К числу достоинств реляционной модели можно отнести:

- наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть 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 манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

К основным критическим положениям относится: их недостаточная эффективность и присущая этим системам некоторая ограниченность (прямое следствие простоты) при использовании в так называемых нетрадиционных областях (САПР). Другими словами, возможности представления знаний о семантической специфике предметной области в реляционных системах очень ограничены. Современные исследования в области по-стреляционных систем главным образом посвящены именно устранению этих недостатков.

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

Для начала покажем смысл этих понятий на примере отношения СОТРУДНИКИ, содержащего информацию о сотрудниках некоторой организации (рис. 2.1).

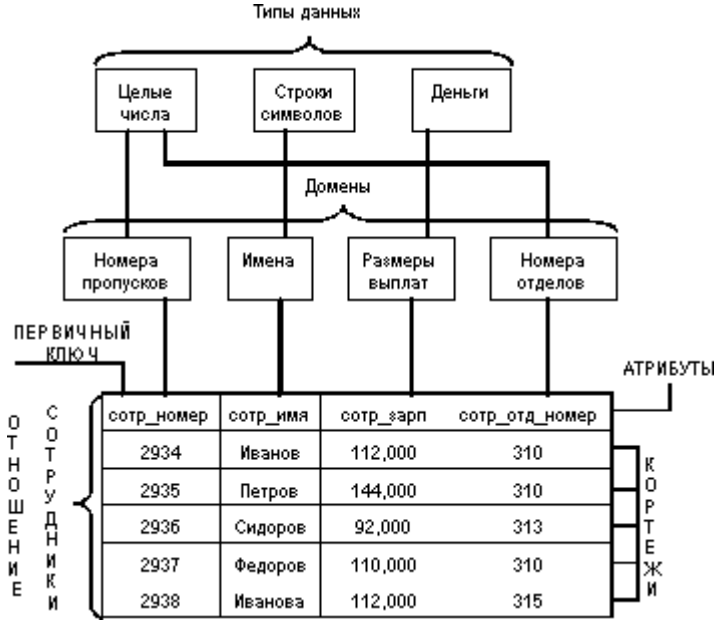


Рис. 2.1. Пример отношения

Отношение — плоская двумерная таблица, состоящая из строк и столбцов. Подобное восприятие не относится к физической структуре, которая может быть реализована по-разному.

Атрибут — это именованный столбец отношения.

Тип данных в реляционной модели данных полностью адекватен понятию типа данных в языках программирования. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, специализированных данных (таких как «деньги», дата, время, временной интервал). В нашем примере мы имеем дело с данными трех типов: строки символов, целые числа и «деньги».

Домен — это набор допустимых значений для одного или нескольких атрибутов. 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.

Для понятия домена важен и такой факт: данные считаются сравнимыми только в том случае, когда они относятся к одному домену. Н-р, атрибуты ФИО и АДРЕС относятся к строковому типу, но не являются сравнимыми.

Кортеж — это строка отношения. Другое определение: кортеж — 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). Кортеж — это набор именованных значений заданного типа.

Схема отношения — это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}.

{имя — множество допустимых имен | VARCHAR}

{др — множество дат | DATE}

{оклад — множество целых положительных чисел | INTEGER}

Степень или «арность» схемы отношения — мощность этого множества. Степень отношения СОТРУДНИКИ равна четырем, то есть оно является 4-арным. Грубо говоря, степень определяется количеством этих атрибу-тов.

Кардинальность отношения — количество кортежей, составляющих отношение.

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

Отношение (другое определение) — 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.

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

Альтернативная терминология

Отношение	Таблица	Файл
Кортеж	Строка	Запись
Атрибут	Столбец	Поле

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

Свойства отношений

1. Уникальность имени отношения и имен атрибутов в отношении
2. Отсутствие кортежей-дубликатов
3. Отсутствие упорядоченности кортежей
4. Отсутствие упорядоченности атрибутов
5. Атомарность значений атрибутов

Отсутствие кортежей-дубликатов. Это свойство 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 первичного ключа — набора атрибутов, значения которых однозначно определяют кортеж отношения. Для каждого отношения по крайней мере полный набор его атрибутов обладает этим свойством. Однако при формальном определении первичного ключа требуется обеспечение его «минимальности», т.е. в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства — однозначно определять кортеж.

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

Отсутствие упорядоченности кортежей 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. Это не противоречит тому, что при формулировании запроса к БД, например, на языке SQL можно потребовать сортировки результирующей таблицы в соответствии со значениями некоторых столбцов. Такой результат, вообще говоря, не отношение, а некоторый упорядоченный список кортежей.

Отсутствие упорядоченности атрибутов следует из определения схемы отношения как множество пар {имя атрибута, имя домена}. Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута. Это свойство теоретически позволяет, например, модифицировать схемы существующих отношений не только путем добавления новых атрибутов, но и путем удаления существующих атрибутов. Однако в большинстве существующих систем такая возможность не допускается, и хотя упорядо-ченность набора атрибутов отношения явно не требуется, часто в качестве неявного порядка атрибутов используется их порядок при определении схемы отношения.

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

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

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

Нормализованные отношения составляют основу классического реляционного подхода к организации баз данных. Они обладают некоторыми ограничениями (не любую информацию удобно представлять в виде плоских таблиц) и избыточностью, но существенно упрощают манипулирование данными. Рассмотрим, например, два идентичных оператора занесения кортежа:

Зачислить сотрудника Кузнецова (пропуск номер 3000, зарплата 115,000) в отдел номер 310 и Зачислить сотрудника Кузнецова (пропуск номер 3000, зарплата 115,000) в отдел номер 320.

Если информация о сотрудниках представлена в виде отношения СОТРУДНИКИ, оба оператора будут выполняться одинаково (вставить кортеж в отношение СОТРУДНИКИ). Если же работать с ненормализованным отношением ОТДЕЛЫ, то первый оператор добавляет информации о Кузнецове в множественное значение атрибута ОТДЕЛ кортежа с первичным ключом 310, а второй перед этим потребует создания нового кортежа.

Лекция 3. РЕЛЯЦИОННЫЕ КЛЮЧИ И СВЯЗИ. ЦЕЛОСТНОСТЬ

Суперключ — любой набор атрибутов, однозначно определяющий кортеж таблицы.

Потенциальный ключ — суперключ, который не содержит подмножества атрибутов, также являющегося суперключом данного отношения.

Отношение может иметь несколько потенциальных ключей. Если ключ состоит из нескольких атрибутов, то он называется составным.

Потенциальный ключ обладает двумя свойствами:

- Уникальность. В каждом кортеже отношения значение ключа единственным образом идентифицирует этот кортеж.
- Неприводимость. Никакое допустимое подмножество ключа не обладает свойством уникальности.

Первичный ключ — потенциальный ключ, который выбран для уникальной идентификации кортежей внутри отношения.

Потенциальные ключи, которые не выбраны в качестве первичного называются альтернативными.

Существует такое понятие как определитель NULL.

Определитель NULL указывает, что значение атрибута в настоящий момент неизвестно или неприемлемо для этого кортежа. Его логическое значение — “неизвестно”. Определитель NULL не следует понимать как нулевое численное значение или пустую строку. Нельзя также говорить, что аргумент имеет значение NULL, т.к. NULL не является значением, а лишь обозначает его отсутствие.

Ни один атрибут первичного ключа не может содержать отсутствующих значений, обозначаемых NULL.

По определению первичный ключ — минимальный набор атрибутов, который используется для уникальной идентификации кортежей. Это значит, что никакое подмножество ПК не может быть достаточным для уникальной идентификации кортежей. Если допустить присутствие определителя NULL в любой части первичного ключа, то это равносильно утверждению, что не все его атрибуты необходимы для уникальной идентификации кортежей, что противоречит определению первичного ключа.

Внешний ключ — атрибут или множество атрибутов внутри отношения, которое соответствует первичному ключу некоторого (м.б. того же самого /рекурсивный ключ/) отношения.

Например, таблица, содержащая сделки, заключенные менеджерами магазина:

Date_sdelki	Summa_sdelki	Client	ID_manager
15.03.2003	100	ОАО "Звук"	3
17.03.2003	150	ЧП "Петренко"	1
20.04.2003	190	ОАО "Цвет"	3
24.04.2003	120	ЗАО "КНП"	4
25.05.2003	130	ЧП "Васин"	2

И таблица МЕНЕДЖЕРЫ:

ID_m	FIO	BD	Adres
1	Нестеров П.К.	12.05.70	A
2	Семенов А.Н.	28.02.74	Z
3	Семенов А.Н.	03.10.75	C
4	Уткина С.М.	31.12.68	V

Здесь каждому менеджеру присваивается идентификатор (первичный ключ) и не допустима ситуация, когда разным менеджерам соответствует одинаковый идентификатор, который и указан в таблице СДЕЛКИ (ID_manager) в качестве вторичного ключа.

В данном случае ID_m — искусственно введенный атрибут, которого нет у сущности предметной области, т.н. суррогатный ключ. Конечно, можно его не вводить и в качестве первичного ключа выбрать составной — FIO + BD, но в этом случае вторичный ключ должен соответствовать первичному, т.е. также должен содержать FIO + BD. Это, безусловно, ведет к избыточности БД.

Можно ли в качестве ключевого поля выбрать FIO?

Возможна ситуация, когда для менеджера вводится естественный уникальный атрибут — номер паспорта, номер страхового

свидетельства, пенсионного, номер в/у и т.д. В этом случае уместно не вводить суррогатный ключ, а воспользоваться этим атрибутом.

Но!

Во-первых, в качестве типа ID выбирается тип longint (4 байта), а, например, водительское удостоверение 40 XX 123456, т.е. 12 символов (байт).

А, во-вторых, при вводе новой записи (устройство на работу нового менеджера) не всегда возможно сразу указать номер в/у и этот атрибут останется пустым, что для первичного ключа недопустимо. В то время как для суррогатного ключа средства СУБД автоматически генерируют новое уникальное значение (автоинкремент).

Говорят, также, что таблица СДЕЛКИ ссылается на таблицу (связана с таблицей) МЕНЕДЖЕРЫ по полю ID_manager.

Связь между сущностями (таблицами) является основополагающим понятием реляционной модели. Различают три вида связи: "один-к-одному" (1:1), "один-ко-многим" (1:N) и "многие-ко-многим" (N:M).

В нашем примере, один менеджер может заключить много сделок, а одна сделка не может быть совершена несколькими менеджерами, поэтому связь между сущностями СДЕЛКА и МЕНЕДЖЕР — "один-ко-многим". Это самый распространенный вид связи в реляционных БД.

В рамках связи "один-ко-многим" таблица с первичным ключом (МЕНЕДЖЕР) называется главной | master (родительской), а таблица ссылаю-щаяся на нее (СДЕЛКИ) — подчиненной | detail, (зависимой, дочерней).

Менее распространена связь "многие-ко-многим". Например, связь между сущностями АВТОР—КНИГА. Автор может написать несколько книг, и одна книга может быть написана группой авторов.

ID_a	FIO	BD	Adres
1	Иванов А.А.	12.03.65	А
2	Петров Р.А.	31.01.68	Б
3	Маслов К.Г.	23.12.71	В
4	Васин П.В.	19.10.55	Г
5	Зайцев Н.Н.	11.11.47	Д
6	Иванов А.А.	25.05.61	Е

ID_kn	Title	V	Tirag
54	Базы данных	15	1000
48	История СССР	24,5	5000
35	Калуга	3	500
01	DirectX	11	1500
91	История СССР	12	2000

Сколько книг может написать один автор? Если не больше 3, то можно добавить к таблице АВТОР три вторичных ключа BOOK1, BOOK2 и BOOK3, где и указать идентификаторы его книг (либо — NULL). Но реально мы не можем заранее знать сколько книг напишет автор и, соответственно, сколько полей добавить.

Такая же ситуация с книгами: сколько авторов участвует в ее написании?

Связь "многие-ко-многим" реализуется при помощи третьей таблицы, в которой указываются пары автор-книга. Например:

ID_a	ID_k
3	35
3	48
5	35
4	35
1	01
6	54
2	91

Два вторичных ключа этой таблицы ссылаются на первичные ключи таблиц АВТОР и КНИГА. Т.е. имеем две связи вида "один-ко-многим", в которых таблицы АВТОР и КНИГА будут главными, а третья таблица — зависимой.

При этом, в эту третью таблицу и автор и книга могут входить много раз.

В нашем примере, автор Маслов К.Г. написал книги "Калуга" и "История СССР", а книга "Калуга" написана двумя авторами: Масловым К.Г. и Зайцевым Н.Н.

Если автор пишет новую книгу, то добавляется новая запись. Если 2 автора написали новую книгу — 2 записи и т.д.

Таким образом, связь "многие-ко-многим" разбивается на две связи "один-ко-многим" и вспомогательную таблицу. Причем во вспомогательной таблице должно соблюдатся требование отсутствия дубликатов.

Как бы выглядела третья таблица, если не вводить идентификаторы (суррогатные ключи)? Что выбрать в качестве первичных ключей?

Связь "один-к-одному" используется наиболее редко.

Чаще всего, исходя из соображений повышения скорости доступа. Например, по книге нужно также хранить, например, такую информацию: формат, издательство, типография, бумага, переплет, аннотация, ISBN, ББК и т.д., но в повседневной работе с таблицей КНИГА эта информация используется очень редко, поэтому ее выносят в отдельную таблицу.

ID_kn	Title	V	Tirag
48	История СССР	24,5	5000
35	Калуга	3	500
01	DirectX	11	1500
91	История СССР	12	2000

→

ID_kn	Izdat	Typogr	Bum	Perep
48	F	Press1	Офс 60	7
35	G	Press2	Офс 70	7В
01	F	Press1	Газ	Б/шв
91	A	Press3	Офс 65	7

В результате данные в БД все-таки сохраняются, а оперативная работа ведется с таблицей 1 гораздо меньшего размера, что не может не сказаться на скорости, особенно при больших объемах записей.

Другой пример связи "один-к-одному": рекурсивный ключ. Требуется вести учет граждан и их семейного положения.

Здесь вторичные ключи таблицы (ID_supr, ID_f, ID_m) ссылаются на первичный ключ (ID) этой же таблицы.

Связь (ID_supr — ID) при моногамных браках будет вида "один-к-одному", при полигамных — "один-ко-многим".

Реляционная целостность

Реляционная модель имеет две части структурную (собственно таблицы) и управляющую, которая определяет типы допустимых операций с данными и набор ограничений, которые гарантируют корректность данных. Эти два основных правила реляционной модели называются целостностью данных и ссылочной целостностью.

Целостность данных означает непротиворечивость хранящихся данных. Например, две таблицы СТУДЕНТЫ (n_sb, fio, group) и ГРУППЫ (title, amount). При добавлении записи в таблицу СТУДЕНТЫ, у соответствующей группы должна изменятся численность.

Другой пример. Таблица БАНКОВСКИЙ СЧЕТ (n_s, summa). Требуется перевести деньги с одного счета на другой. Понятно, что на сколько уменьшилась сумма с одного счета, ровно на столько же должна увеличится сумма на другом счете.

Возраст не может быть отрицательным числом, пол может быть только одним значением из двух возможных и т.п. ограничения домена.

Кроме общих правил по поддержанию целостности данных, в каждом конкретном случае могут быть установлены дополнительные ограничения, которые еще называют корпоративными ограничениями или бизнес-правилами. Например, нельзя зачислить студента, возрастом старше 45 лет, нельзя использовать для поля КУРС значения <1 и >7.

Ссылочная целостность

Если в отношении есть внешний ключ, то значение внешнего ключа должно либо соответствовать значению потенциального ключа некоторого кортежа в базовом отношении, либо задаваться определение NULL.

В предыдущем примере родители Адама и Евы (ID_f, ID_m) будут задаваться как NULL, а также супруги Каина и Авеля нам тоже неизвестны.

Также не может быть ситуации, когда в качестве родителя указан ID не существующий среди значений первичного ключа.

Согласно бизнес-правила планеты Z — ее жители гермафродиты. Как будет выглядеть реляционная таблица учета ее жителей и их родства?

Модель данных "сущность-связь"

Для проектирования баз данных используются различные подходы. В достаточно несложных случаях строится ER-модель (entity-relationship) данных в виде диаграммы, на которой отображаются сущности (отношения) и связи между ними.

На диаграмме сущности показываются прямоугольниками, атрибуты сущности эллипсами, связи — прямоугольниками.

Пример: фильмы, режиссеры, актеры.

Связь между фильмом и актером — "многие-ко-многим". Следовательно будет еще одно связующее отношение, которое носит название слабой сущности (показывается двойной линией). Атрибуты могут быть: простые, составные (в виде присоединенных к нему эллипсов), производными (пунктиром), многозначными (двойной линией).

Лекция 4. НОРМАЛИЗАЦИЯ

При проектировании базы данных основной целью является создание наиболее точного представления данных, связей между ними и требуемых ограничений. И прежде всего определить отношения. Метод, который используется для решения этой задачи называется нормализацией.

Не любое отношение имеет право на существование. Существует ряд требований. Цель этих требований — уменьшение избыточности и и вероятности непреднамеренной потери данных. Требования эти могут быть Ре-ля¬ционная база данных должна быть нормализована. Т.е. отношения, образующие БД должны отвечать ряду требований. Процесс нормализации имеет своей целью устранение избыточности данных и предотвращение возможных нарушений целостности.

Сначала (Кодд, 1970) были предложены первые три нормальных формы: 1НФ, 2НФ, 3НФ. Затем было сформулированное более строгое определение третьей нормальной формы, которое получило название нормальная форма Бойса-Кодда. Затем появились 4НФ и 5НФ, на практике используемые крайне редко.

На практике процесс разработки БД сводится к достижению 3НФ. При достаточном опыте проектировщика БД нормализация происходит интуитивно. Либо ее можно использовать в качестве тестов, т.е. повергнуть уже разработанную БД проверке: удовлетворяет ли она требованиям нормализации.

Требуется обеспечить учет накладных на отпуск товара. Примерный вид накладной:

Накладная № 9999 от 29.02.04

Клиент: ООО "Золотой ключик"
Адрес: г. Калуга, ул. Труда-99 *Тел*: 99-99-99

Товар	*Количество*	*Цена*	*Ед. изм*	*Сумма*
Сахар	3	15.00	кг.	45
Молоко	2	16.50	л	31
Макароны	1	25.00	пач.	25

Ненормализованная таблица накладных будет иметь следующий вид:

N_nom	N_date	C_name	C_adr	C_tel	T_name	T_kol	T_cena	T_edizm	T_sum
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14	Крупа Сахар Макароны	12 10 9	13 16,5 12	кг кг кг	156 165 108
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57	Тушенка Подсол. масло Сахар	5 2 1	21 29 23	банка бут пачка	105 58 23
0003	11.01.04	ЧП "Васин"	пос. Росва	34-29-01	Печенье Горошек	3 2	30 16,5	кг банка	90 33

В данной таблице каждой строке соответствует накладная, у которой есть свой список товаров. Вот этот то список товаров и не является атомарным.

Первая нормальная форма

Первая нормальная форма (1НФ) требует, чтобы каждое поле таблицы БД было атомарным (неделимым).

Атомарность поля означает, что содержащиеся в нем значения не должны делиться на более мелкие.

Будет ли поле "Фамилия" неделимым? Ясно, что будет, куда же его еще делить. А поле "ФИО"? Тут все зависит от контекста — если вы не предполагаете использовать потом экзотические запросы типа "Выбрать всех Александровичей", то тоже можно считать его неделимым. А вот адрес лучше разделить на улицу, дом и квартиру, т.е. на 2 или 3 поля, тогда вам будет проще выбрать всех живущих на одной улице. Хотя, можете и его считать неделимым, если адрес у вас — второстепенная информация.

После приведения к 1НФ таблица будет выглядеть:

N_nom	N_date	C_name	C_adr	C_tel	T_name	T_kol	T_cena	T_edizm	T_sum
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14	Крупа	12	13	кг	156
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14	Сахар	10	16,5	кг	165
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14	Макароны	9	12	кг	108
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57	Тушенка	5	21	банка	105
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57	Подсол. масло	2	29	бут	58
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57	Сахар	1	23	пачка	23
0003	11.01.04	ЧП "Васин"	пос. Росва	34-29-01	Печенье	3	30	кг	90
0003	11.01.04	ЧП "Васин"	пос. Росва	34-29-01	Горошек	2	16,5	банка	33

Все поля данной таблицы неделимы.

Определим певичный ключ этого отношения. Номер накладной не может уникально определять запись, поскольку, он будет одинаков для всех записей, относящихся к одной и той же накладной (в случае если по данной накладной выписано более одного товара). Поэтому введем в первичный ключ поле "T_name".

При этом исходим из предположения, что по одной накладной может быть отпущено одно наименование конкретного товара, то есть не может иметь место ситуация, когда отпуск одного и того же товара оформляется в накладной двумя строками, что повлекло бы за собой две одинаковые записи в таблице «Отпуск товаров со склада»

N_nom	N_date	C_name	C_adr	C_tel	**T_name**	T_kol	T_cena	T_edizm	T_sum

При работе с таблицей в 1НФ существуют т.н. аномалии вставки, обновления и удаления.

Аномалия ввода — невозможность ввести данные в таблицу, вызванную отсутствием других данных. Н-р, невозможно добавить нового клиента, если он еще для него не выписана накладная. Или н-р, у данной накладной требуется хранить имя работника, который ее выписал. Если работник нанят на работу, но не успел еще выписать ни одной накладной. В том случае если не допускаются пустые значения мы не можем принять на работу нового человека.

Аномалия обновления — противоречивость данных, вызванная их избыточностью или частичным обновлением. Н-р, у ТОО «Кедр» сменилась форма собственности на "ЗАО". Значит, во всех строках таблицы нужно изменить поле C_ADR. Если этого не сделать, то получим из одного клиента — двух.

Аномалия удаления — непреднамеренная потеря данных, вызванная потерей других данных. Допустим покупатель не объявлялся в течение года, а по бизнес-правилам магазина информация о продажах хранится в течение 6 мес., а затем уничтожается. Т.о. информация о покупателе также окажется удаленной.

Вторая нормальная форма 2НФ

Существует такое понятие как функциональная зависимость (значение атрибута А в кортеже однозначно определяет значение атрибута В). ФЗ обозначается А -> В, -> читается как “функционально определяет”. Для одного значения В может существовать несколько значений А. При этом атрибуты А и В могут быть и составными.

Атрибут в левой части ФЗ (А) называется детерминантом, т.к. его зна-чение определяет значения других атрибутов ФЗ.

В нашем примере можно выделить следующие ФЗ:

N_nom –> N_date, C_name, C_adr, C_tel

C_name -> C_adr, C_tel

T_name –> T_kol, T_cena, T_edizm, T_sum

Вторая нормальная форма требует, чтобы никакие неключевые атри-буты не являлись функционально зависимыми лишь от части ключа. Те по-ля, которые зависят только от части первичного ключа, должны быть вы-делены в составе отдельных таблиц.

Таким образом, отношение у которого первичный ключ НЕ является составным атрибутом всегда находится во 2НФ.

Первое требование 2НФ выполнено, чего не скажешь о втором, гласящем, что значения всех полей записи должны однозначно зависеть от совокупного значения первичного ключа и не должна иметь место ситуация, когда некоторые поля зависят от части первичного ключа.

Поля «Единица измерения», «Цена за единицу измерения» зависят от значения поля «Товар», являющегося частью первичного ключа. Поэтому выделяем эти поля в самостоятельную таблицу «Товары».

T_name	T_cena	T_edizm
Крупа	13	кг
Сахар	16,5	кг
Макароны	12	кг
Тушенка	21	банка
Подсол. масло	29	бут
Сахар	23	пачка
Печенье	30	кг
Горошек	16,5	банка

Первая таблица имеет вид:

N_nom	N_date	C_name	C_adr	C_tel	T_name	T_kol	T_sum
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14	Крупа	12	156
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14	Сахар	10	165
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14	Макароны	9	108
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57	Тушенка	5	105
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57	Подсол. Масло	2	58
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57	Сахар	1	23
0003	11.01.04	ЧП "Васин"	пос. Росва	34-29-01	Печенье	3	90
0003	11.01.04	ЧП "Васин"	пос. Росва	34-29-01	Горошек	2	33

Следующая ФЗ: N_nom –> N_date, C_name, C_adr, C_tel также зависит от части первичного ключа и выделяются в отдельную таблицу.

N_nom	N_date	C_name	C_adr	C_tel
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57
0003	11.01.04	ЧП "Васин"	пос. Росва	34-29-01

Исходная таблица теперь имеет вид:

N_nom	T_name	T_kol	T_sum
0001	Крупа	12	156
0001	Сахар	10	165
0001	Макароны	9	108
0002	Тушенка	5	105
0002	Подсол. Масло	2	58
0002	Сахар	1	23
0003	Печенье	3	90
0003	Горошек	2	33

Т.о. формально процесс разбиения таблицы на две во 2НФ состоит из следующих шагов.

1. Создается новая таблица, атрибутами которой будут атрибуты ис-ходной таблицы, входящие в противоречащую правилу ФЗ.
2. Атрибут в правой части ФЗ исключается из исходной таблицы.
3. Детерминант этой ФЗ будет первичным ключом новой таблицы.

Третья нормальная форма 3НФ

Если в некотором отношении существуют ФЗ A□B и B□C, то говорят, что атрибут C транзитивно зависит от атрибута A.

Например, в отношении, полученном в результате приведения к 2НФ:

N_nom	N_date	C_name	C_adr	C_tel
0001	10.01.04	ООО "Звук"	г. Калуга	72-12-14
0002	11.01.04	ТОО "Кедр"	г. Медынь	57-50-57
0003	11.01.04	ЧП "Васин"	пос. Росва	34-29-01

имеем следующую транзитивную зависимость:

$$N\_nom \rightarrow C\_name, \qquad C\_name \rightarrow C\_adr, C\_tel$$

3НФ требует, чтобы отношение не имело транзитивно зависимых от первичного ключа неключевых атрибутов.

Поэтому выделяем покупателя в отдельную таблицу (детерминант ФЗ — ключ новой таблицы — остается в исходной):

C_name	C_adr	C_tel
ООО "Звук"	г. Калуга	72-12-14
ТОО "Кедр"	г. Медынь	57-50-57
ЧП "Васин"	пос. Росва	34-29-01

Исходная:

N_nom	N_date	C_name
0001	10.01.04	ООО "Звук"
0002	11.01.04	ТОО "Кедр"
0003	11.01.04	ЧП "Васин"

Для решения большинства задач приведения к 3НФ бывает достаточно. Нормальные формы высших порядков используются на практике крайне редко.

Полученные в результате приведения к 3НФ отношения:

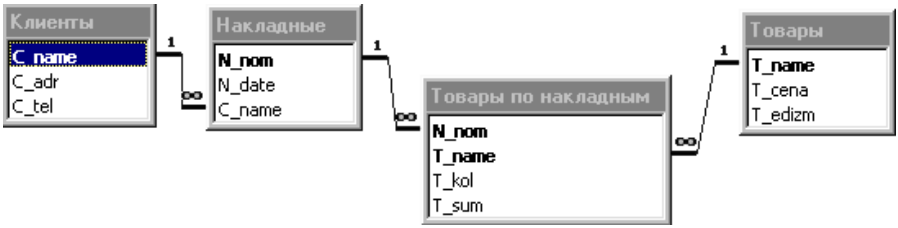


Рис. 4.1. Отношение, приведенное к 3НФ

Нормальная форма Бойса-Кодда

Отношение находится в НФБК, если каждый его детерминант является потенциальным ключом.

НФБК является более строгим выражением 3НФ. Нарушений требований НФБК происходит крайне редко и только когда в отношении имеется два и более составных потенциальных ключей, и когда эти потенциальные ключи перекрываются, т.е. имеют, по крайней мере, один общий атрибут.

В приведенном примере все отношения удовлетворяют НФБК, т.к. имеют только один потенциальный ключ — первичный.

Пример: Поставщик_Номер, Поставщик_Наименование, Деталь_Номер, Деталь_Количество.

Четвертая нормальная форма

Отношение находится в 4НФ, если оно находится в 3НФ и не имеет многозначных зависимостей.

Многозначная зависимость (multi-valued dependency) предполагает наличие для каждого значения атрибута A набора значений атрибута B и набора значений атрибута C. Причем входящие в эти наборы значения атрибутов B и C не зависят друг от друга.

Возможность существования в таблице МЗЗ возникает вследствие приведения исходных таблиц к 1НФ, для которой не допускается наличия набора значений для атрибута.

Например, таблица содержащая номер отдела, сотрудника отдела и задания выданные на отдел:

Номер отдела	Сотрудники	Задания
01	Иванов Петров Сидоров	Проект А Проект В
02	Теплов Круглов	Задание S Задание D

Поскольку такой таблицы быть не может (таблица д.б. в 1НФ), а атрибуты Сотрудники и Задания — многозначные, то для приведения к 1НФ необходимо создать кортежи сочетающие каждое из значений одного атрибута с каждым значением второго.

Номер отдела	Сотрудники	Задания
01	Иванов	Проект А
01	Петров	Проект А
01	Сидоров	Проект А
01	Иванов	Проект В
01	Петров	Проект В
01	Сидоров	Проект В
02	Теплов	Задание S
02	Круглов	Задание S
02	Теплов	Задание D
02	Круглов	Задание D

Поскольку в нашем примере Задания выдаются на Отдел, а не конкретному сотруднику, то данное отношение искажает реальную ситуацию.

В данном примере две многозначных зависимости: A ->> B (№ отдела ->> Сотрудник) и A ->> C (№ отдела ->> Задания)

Четвертая НФ требует устранить подобного рода избыточность посредством выделения в новое отношение одного или нескольких участвующих в МЗЗ атрибутов вместе с копией детерминанта:

Сотрудник	№ отдела
Иванов	01
Петров	01
Сидоров	01
Теплов	02
Круглов	02

Задание	№ отдела
Проект A	01
Проект B	01
Задание S	02
Задание D	02

Пятая нормальная форма

5НФ требует, чтобы отношения находились без зависимостей соединения. Т.е. чтобы при естественном соединении двух отношений, которые получились в результате приведения к 3НФ не возникало ложных кортежей.

Пример нарушения 5НФ к экзамену найти самостоятельно.

На практике не всегда возможно по¬лучить идеально нормализованную БД. Часто к этому и не стремятся по тем причинам, что, хотя нормализация БД ведет у снижению избыточности и уменьшению риска нарушения це-лостности, она также ведет к затруднению целостного восприятия БД из-за возрастающего количества таблиц и связей между ними. На практике чем больше таблиц участвует в получении данных, тем медленнее выполняется запрос, поэтому на практике части ищут компромисс между требованиями нормализации (т.е. логичности данных и экономии внешней памяти) и необходимостью обеспечения быстродействия системы.

Лекция 5. ОБЩИЙ ОБЗОР СУБД

Системы управления базами данных (СУБД)

База данных — совместно используемый набор логически связанных данных и описание этих данных.

Именно описание обеспечивает независимость между данными и программами, их использующими. Например, добавление нового поля не потребует от приложения никаких изменений. Однако, при удалении поля потребуют соответствующей переделки и приложения его использующие.

Описание данных, хранящееся совместно с самими данными называется словарем данных или системным каталогом, а элементы описания — метаданнные (т.е. данные о данных).

В системном каталоге содержатся следующие сведения:

- имена, типы и размеры элементов данных;
- имена связей;
- ограничения целостности данных;
- имена зарегистрированных пользователей, которым предоставлены некото¯рые права доступа к данным;
- используемые индексы и структуры хранения — например, инвертированные файлы или деревья B+.

СУБД — это программное обеспечение, с помощью которого пользователи могут определять 1), создавать и поддерживать 2) базу данных, а также осуществлять к ней контролируемый доступ 3).

1) Определение БД осуществляется при помощи языка определения данных DDL (Data Definition Language), который предоставляет пользователям средства указания типов данных, их структуры, а также средств задания ограничений на хранимые данные.

2) Создание, обновление, удаление, извлечение и т.д. информации из БД осуществляется при помощи языка управления данными DML (Data Ma-nipulation Language). Языки DML бывают процедурные (взаимодействуют с БД последовательно запись за записью, отвечают на вопрос "КАК получить желаемый результат?") и непроцедурные (оперируют сразу целыми наборами записей, отвечают на вопрос "ЧТО нужно получить в результате?"). Наиболее распространенным непроцедурным языком является SQL.

3) Контролируемый доступ обеспечивается при помощи:

– системы обеспечения безопасности, предотвращающей несанкционированный доступ к базе данных со стороны пользователей;
– системы поддержки целостности данных, обеспечивающей непротиворечивое состояние хранимых данных;
– системы управления параллельной работой приложений, контролирующей процессы их совместного доступа к базе данных;
– системы восстановления, позволяющей восстановить базу данных до предыдущего непротиворечивого состояния, нарушенного в результате сбоя аппаратного или программного обеспечения;
– доступного пользователям каталога, содержащего описание хранимой в базе данных информации.

Компоненты среды СУБД

Основные компоненты среды СУБД представлены на рис. 5.1.

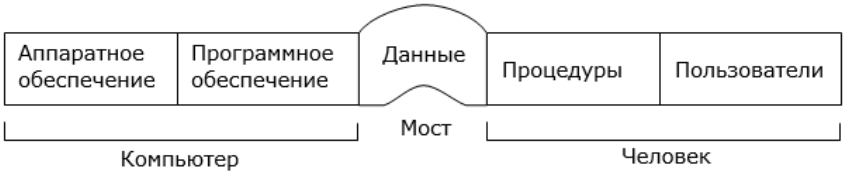


Рис. 5.1. Компоненты среды СУБД

Аппаратное обеспечение может варьироваться в очень широких пределах — от единственного персонального компьютера до сети из многих компьютеров.

Используемое аппаратное обеспечение зависит от требований данной организации и ис-пользуе¬мой СУБД. Одни СУБД предназначены для работы только с конкретными типами операционных систем или оборудования, другие могут работать с широким кругом аппаратного обеспечения и различными операционными системами. Для работы СУБД обычно требуется некоторый минимум оперативной и дисковой памяти, но та¬кой

минимальной конфигурации может оказаться совершенно недостаточно для дос¬тижения приемлемой производительности системы.

Программное обеспечение. Сюда входят программное обеспечение самой СУБД, прикладные программы, операционные системы. Приложения создаются на языках высокого уровня, которые используют операторы SQL.

Данные являются, вероятно, самым важным компонентом среды СУБД (с точки зрения конечных пользователей). Структура базы данных называется схемой (Schema). Схема состоит из таблиц (Table). Таблица состоит из полей (Fields) и записей (Record).

К процедурам относятся инструкции и правила, которые должны учитываться при проектировании и использовании базы данных. Пользователям и обслуживаю¬щему персоналу базы данных необходимо предоставить документацию, содержащую подробное описание процедур использования и сопровождения данной системы, включая инструкции о правилах выполнения приведенных ниже действий.

- Регистрация в СУБД.
- Использование отдельного инструмента СУБД или приложения.
- Запуск и останов СУБД.
- Создание резервных копий СУБД.
- Обработка сбоев аппаратного и программного обеспечения, включая процедуры идентификации вышедшего из строя компонента, исправления отказавшего компонента (например, посредством вызова специалиста по ремон¬ту аппаратного обеспечения), а также восстановления базы данных после устранения неисправности.
- Изменение структуры таблицы, реорганизация базы данных, размещенной на нескольких дисках, способы улучшения производительности и методы архивирования данных на вторичных устройствах хранения.

Среди пользователей можно выделить четыре основные группы: администраторы данных и баз данных, разработчики баз данных, прикладные программисты и конечные пользователи.

Администратор данных, или АД, отвечает управление данными, планирование БД, разработку и сопровождение стандартов, бизнес-правил и деловых процедур, а также за концептуальное и логическое проектирование базы данных.

Администратор базы данных, или АБД, отвечает за физическую реализацию базы данных, включая физическое проектирование и воплощение проекта, за обеспечение безопасности и целостности данных, за сопровождение операционной системы, а также за обеспечение максимальной производитель¬ности приложений и пользователей.

По сравнению с АД, обязанности АБД носят более технический характер, и для него необходимо знание конкретной СУБД и системного окружения.

В одних организациях между этими ролями не делается раз¬личий, а в других важность корпоративных ресурсов отражена именно в выделении отдельных групп персонала с указанным кругом обязанностей.

Разработчики баз данных. В проектировании больших баз данных участвуют два разных типа разработчи¬ков: разработчики логической базы данных и разработчики физической базы дан¬ных.

Разработчик логической базы данных занимается идентификацией данных (т.е. сущностей и их атрибутов), связей между данными и устанавливает ограничения, накладываемые на хранимые данные. Разработчик логической базы данных должен обладать всесторонним и полным пониманием структуры данных организации и ее бизнес-правил.

Для эффективной работы разработчик логической базы данных должен как мож¬но раньше вовлечь всех предполагаемых пользователей базы данных в процесс созда¬ния модели данных.

Разработка логической БД делится на два этапа:

– Концептуальное проектирование базы данных, которое совершенно не за¬висит от таких деталей ее воплощения, как конкретная целевая СУБД, приложения, языки программирования или любые другие физические характеристики.

– Логическое проектирование базы данных, которое проводится с учетом особенностей выбранной модели данных: реляционной, сетевой, иерархической или объектно-ориентированной.

Разработчик физической базы данных получает готовую логическую модель дан¬ных, занимается ее физической реализацией, в том числе:

– преобразованием логической модели данных в набор таблиц и ограничений целостности данных;
– выбором конкретных структур хранения и методов доступа к данным, обеспечивающих необходимый уровень производительности при работе с базой данных;
– проектированием любых требуемых мер защиты данных.

Многие этапы физического проектирования базы данных в значительной степени зависят от выбранной целевой СУБД, а потому может существовать несколько различных способов воплощения требуемой схемы. Следовательно, разработчик физической базы данных должен разбираться в функциональных возможностях целевой СУБД и понимать достоинства и недостатки каждого возможного варианта воплощения. Разработчик физической базы данных должен уметь выбрать наиболее подходящую стратегию хранения данных с учетом всех существующих особен¬ностей их использования. Если концептуальное и логическое проектирование базы данных отвечает на вопрос "что?", то физическое проектирование отвечает на во¬прос "как?". Для решения этих задач требуются разные навыки работы, которыми чаще всего обладают разные люди.

Сразу после создания базы данных следует приступить к разработке приложений, предоставляющих пользователям необходимые им функциональные возможности. Именно эту работу и выполняют прикладные программисты. Как правило, каждая программа содержит некоторые операторы, требующие от СУБД выполнения определенных действий с базой данных — например, таких как извлечение, вставка, обновление или удаление данных. Эти программы могут создаваться на различных языках.

Пользователи являются клиентами базы данных — она проектируется, создается и поддерживается для того, чтобы

обслуживать их информационные потребности. Пользователей можно классифицировать по способу использования ими системы.

Наивные пользователи обычно даже и не подозревают о наличии СУБД. Они обращаются к базе данных с помощью специальных приложений, позволяющих в максимальной степени упростить выполняемые ими опе¬раций. Такие пользователи инициируют выполнение операций базы дан¬ных, вводя простейшие команды или выбирая команды меню. Это зна¬чит, что таким пользователям не нужно ничего знать о базе данных или СУБД.

Например, чтобы узнать цену товара, кассир в супермаркете использует сканер для считывания нанесенного на него штрих-кода. В ре¬зультате этого простейшего действия специальная программа не только считывает штрих-код, но и выбирает на основе его значения цену товара из базы данных, а также уменьшает значение в другом поле базы дан¬ных, обозначающем остаток таких товаров на складе, после чего выбива¬ет цену и общую стоимость на кассовом аппарате.

Опытные пользователи. С другой стороны спектра находятся опытные конечные пользователи, которые знакомы со структурой базы данных и возможностями СУБД. Для выполнения требуемых операций они могут использовать такой язык запросов высокого уровня, как SQL. А неко¬торые опытные пользователи могут даже создавать собственные при¬кладные программы.

Функции СУБД

1. Хранение, извлечение и обновление данных
2. Системный каталог
3. Средства для параллельной работы
4. Поддержка транзакций
5. Службы восстановления
6. Контроль доступа к данным
7. Поддержка обмена данными
8. Службы поддержки целостности данных
9. Вспомогательные службы

Хранение, извлечение и обновление данных. СУБД должна предоставлять пользователям возможность сохранять, извлекать и обновлять данные в базе данных. Кроме того, СУБД должна обеспечить быстрый поиск необходимых данных.

Реализация этой функции должна позволить скры¬ть от конечного пользователя внутренние детали физической реализации системы (например, файловую организацию или используемые структуры хранения).

СУБД должна иметь доступный конечным пользователям интегрированный системный каталог с данными о схемах, пользователях, приложениях и т.д. Предполагается, что каталог доступен как пользователям, так и функциям СУБД. Системный каталог, или словарь данных, является хранилищем информа¬ции, описывающей данные в базе данных (по сути, это "данные о данных", или метаданные). В зависимости от типа используемой СУБД количество информации и способ ее применения могут варьироваться.

Средства для параллельной работы. Одна из основных целей СУБД заключается в том, чтобы множество пользователей могло осуществлять параллельный доступ к совместно обрабатываемым данным.

СУБД должна иметь механизм, который гарантирует корректное обновление базы данных при параллельном выполнении операций обновления многими пользователями.

Параллельный доступ сравнительно просто организовать, если все пользователи выполняют только чтение данных, поскольку в этом случае они не могут помешать друг другу. Однако, когда два или больше пользователей одновременно получают доступ к базе данных, конфликт с нежелательными последствиями легко может возникнуть, например, если хотя бы один из них попытается обновить данные.

СУБД должна гарантировать, что при одновременном доступе к базе данных мно¬гих пользователей не произойдет конфликтов одновременного изменения одних и тех же данных.

Поддержка транзакций. Транзакция представляет собой набор действий, выполняемых отдельным пользо¬вателем для доступа или изменения данных. Примерами простых транзакций может служить

добавление в базу данных сведений о новом сотруднике, обновление сведений о зар¬плате некоторого сотрудника или удаление сведений о некотором объекте. Классическим примером более сложной транзакции может быть бухгалтерская проводка, когда деньги переходят с одного счета на другой. В этом случае в базу данных потребуется внести сразу несколько изменений (транзакция состоит из нескольких действий): записать информацию по кредиту одного счета и по дебету другого. Если во время выполнения транзакции произойдет сбой, например, из-за выхода из строя компьютера, база данных попадает в противоречивое состояние, поскольку деньги списаны с одного счета и не зачислены на другой.

СУБД должна иметь механизм, который гарантирует выполнение либо всех операций данной транзакции, либо ни одной из них.

СУБД должна предоставлять средства восстановления базы данных на случай ка¬кого-либо ее повреждения.

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

Примерами программных сбоев могут быть: аварийное завершение работы СУБД (по причине ошибки в программе или в результате некоторого аппаратного сбоя) или аварийное завершение пользовательской программы, в результате чего некоторая транзакция остается незавершенной. Первую ситуацию можно рассматривать как особый вид мягкого аппаратного сбоя; при возникновении последней требуется ликвидировать последствия только одной транзакции.

Понятно, что в любом случае для восстановления БД нужно располагать некоторой дополнительной информацией. Другими словами, поддержание надежности хранения данных в БД требует избыточности хранения данных, причем та часть данных, которая

используется для восстановления, должна храниться особо надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений БД.

Журнал — это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью (иногда поддерживаются две копии журнала, располагаемые на разных физических дисках), в которую поступают записи обо всех изменениях основной части БД.

В разных СУБД изменения БД журнализируются на разных уровнях: иногда запись в журнале соответствует некоторой логической операции изменения БД (например, операции удаления строки из таблицы реляционной БД), иногда - минимальной внутренней операции модификации страницы внешней памяти; в некоторых системах одновременно используются оба подхода.

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

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия — это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя.

Контроль доступа к данным. СУБД должна иметь механизм, гарантирующий возможность доступа к базе дан¬ных только санкционированных пользователей.

Термин безопас¬ность относится к защите базы данных от преднамеренного или случайного несанк¬ционированного доступа. Предполагается, что СУБД обеспечивает механизмы подоб¬ной защиты данных.

Поддержка обмена данными. СУБД должна обладать способностью к интеграции с коммуникационным про¬граммным обеспечением. Доступ к БД должен осуществляться как посредством локальных, так и удаленных каналов связи.

Службы поддержки целостности данных. СУБД должна обладать инструментами контроля за тем, чтобы данные и их изме¬нения соответствовали заданным правилам.

Целостность базы данных означает корректность и непротиворечивость хранимых данных. Она может рассматриваться как еще один тип защиты базы данных. Поми¬мо того, что данный вопрос связан с обеспечением безопасности, он имеет более широкий смысл, поскольку целостность связана с качеством самих данных. Целост¬ность обычно выражается в виде ограничений или правил сохранения непротиворе¬чивости данных, которые не должны нарушаться в базе. Например, можно указать, что сотрудник не может отвечать одновременно более чем за десять объектов недвижимости. В этом случае при попытке закрепить очередной объект недвижимости за некоторым сотрудником, СУБД должна проверить, не превышен ли установленный лимит и в случае обнаружения подобного превышения запретить закрепление нового объекта за данным сотрудником.

СУБД должна предоставлять некоторый набор различных вспомогательных служб.

Вспомогательные утилиты обычно предназначены для оказания помощи АБД в эффективном администрировании базы данных. Одни утилиты работают на внешнем уровне, а потому они, в принципе, могут быть созданы самим АБД, тогда как другие функционируют на внутреннем уровне системы и потому должны быть предоставлены самим разработчиком СУБД. Ниже приводятся некоторые примеры подобных утилит.

– Утилиты импортирования, предназначенные для загрузки базы данных из плоских файлов, а также утилиты экспортирования, которые служат для выгрузки базы данных в плоские файлы.
– Средства мониторинга, предназначенные для отслеживания характеристик функционирования и использования базы данных.
– Программы статистического анализа, позволяющие оценить производительность или степень использования базы данных.
– Инструменты реорганизации индексов, предназначенные для перестройки индексов и обработки случаев их переполнения.
– Инструменты перераспределения памяти для физического устранения удаленных записей с запоминающих устройств, объединения освобожденного пространства и перераспределения памяти в случае необходимости.

Языки баз данных

Внутренний язык СУБД для работы с данными состоит из двух частей: языка определения данных (Data Definition Language — DDL) и языка управления данными (Data Manipulation Language — DML). Язык DDL используется для определения схемы базы данных, а язык DML — для чтения и обновления данных, хранимых в базе. Эти языки называются подъязыками данных, поскольку в них отсутствуют конструкции для выполнения всех вычислительных операций, обычно используемых в языках програм-ми¬рования высокого уровня. Во многих СУБД предусмотрена возможность внедрения операторов подъязыка данных в программы, написанные на таких языках программи¬рования высокого уровня. В этом случае язык высокого уровня принято называть базовым языком (host language). Перед компиля¬цией файла программы на базовом языке, содержащей внедренные операторы подъязыка данных, потребуется предварительно удалить эти операторы, заменив их вызова¬ми соответствующих функций СУБД. Затем этот предварительно обработанный файл обычным образом компилируется с помещением результатов в объектный модуль, ко¬торый компонуется с библиотекой, содержащей вызываемые в программе функции СУБД. После этого полученный программный текст будет готов к выполнению. Помимо механизма внедрения, для большинства

подъязыков данных также предоставляют¬ся средства интерактивного выполнения их операторов, вводимых пользователем непо¬средственно со своего терминала.

Язык определения данных — DDL

Описательный язык, который позволяет АБД или пользователю описать и поименовать сущности, необходимые для работы некоторого приложения, а также связи, имеющиеся между различными сущностями.

Схема базы данных состоит из набора определений, выраженных на специальном языке определения данных — DDL. Язык DDL используется как для определения новой схемы, так и для модификации уже существующей. Этот язык нельзя исполь¬зовать для управления данными.

Результатом компиляции DDL-операторов является набор таблиц, хранимый в системном каталоге. В системном каталоге интег¬рированы мета-данные — т.е. данные, которые описывают объекты базы данных, а также позволяют упростить способ доступа к ним и управления ими. Метаданные включают определения записей, элементов данных, а также другие объекты, представляющие интерес для пользователей или необходимые для работы СУБД. Перед доступом к реальным данным СУБД обычно обращается к системному каталогу.

Язык управления данными — DML

Язык, содержащий набор операторов для поддержки основных операций манипулирования содержащимися в базе данными.

К операциям управления данными относятся:

- вставка в базу данных новых сведений;
- модификация сведений, хранимых в базе данных;
- извлечение сведений, содержащихся в базе данных;
- удаление сведений из базы данных.

Таким образом, одна из основных функций СУБД заключается в поддержке языка манипулирования данными, с помощью которого пользователь может создавать выражения для выполнения перечисленных выше операций с данными. Понятие ма¬нипулирования данными применимо как к внешнему и

концептуальному уровням, так и к внутреннему уровню. Однако на внутреннем уровне для этого необходимо оп¬ределить очень сложные процедуры низкого уровня, позволяющие выполнять доступ к данным весьма эффективно. На более высоких уровнях, наоборот, акцент перено¬сится в сторону большей простоты использования и основные усилия направляются на обеспечение эффективного взаимодействия пользователя с системой.

Языки DML отличаются базовыми конструкциями извлечения данных. Следует раз¬личать два типа языков DML: процедурный и непроцедурный. Основное отличие между ними заключается в том, что процедурные языки указывают то, как можно получить ре¬зультат оператора языка DML, тогда как непроцедурные языки описывают то, какой ре¬зультат будет получен. Как правило, в процедурных языках записи рассматриваются по отдельности, тогда как непроцедурные языки оперируют с целыми наборами записей.

Процедурные языки DML сообщают системе о том, какие данные необходимы, и точно указать, как их можно извлечь.

С помощью них пользователь, а точнее — программист, указывает на то, какие данные ему необходимы и как их можно получить. Это зна¬чит, что пользователь должен определить все операции доступа к данным (осуществляемые посредством вызова соответствующих процедур), которые должны быть выполнены для получения требуемой информации. Обычно такой процедурный язык DML позволяет извлечь запись, обработать ее и, в зависимости от полученных результатов, извлечь другую запись, которая должна быть подвергнута аналогичной обработке, и т.д. Подобный процесс извлечения данных продолжается до тех пор, пока не будут извлечены все запрашиваемые данные. Языки DML сетевых и иерархических СУБД обычно являются процедурными.

Непроцедурные языки DML позволяют указать лишь то, какие данные требуются, но не то, как их следует извлекать.

Непроцедурные языки DML позволяют определить весь набор требуемых данных с помощью одного оператора извлечения или обновления. С помощью непроцедурных языков DML пользователь указывает, какие данные ему нужны, без определения способа их

получения. СУБД транслирует выражение на языке DML в процедуру (или набор процедур), которая обеспечивает манипулирование затребованным набором записей. Данный подход освобождает пользователя от необходимости знать детали внутренней реализации структур данных и особенности алгоритмов, используемых для извлечения и возможного преобразования данных В результате работа пользователя получает определенную степень независимости от данных. Непроцедурные языки часто также называют декларативными языками. Реляционные СУБД в той или иной форме обычно включают поддержку непроцедурных языков манипулирования данными — чаще всего это бывает язык структурированных запросов SQL (Structured Query Language) или язык запросов по образцу QBE (Query-by Example). Непроцедурные языки обычно проще понять и использовать, чем процедурные языки DML, поскольку пользователем выполняется меньшая часть работы, а СУБД — большая.

Языки 4GL

Аббревиатура "4GL" представляет собой сокращенный английский вариант написания термина язык четвертого поколения (Fourth Generation Language). Не существует четкого определения этого понятия, хотя, по сути, речь идет о некотором стенографическом варианте языка программирования. Если для организации некоторой операции с данными на языке третьего поколения (3GL) типа COBOL потребуется записать сотни строк кода, то для реализации этой же операции на языке четвертого поколения будет достаточно 10-20 строк.

В то время как языки третьего поколения являются процедурными, языки 4GL Вступают, как непроцедурные, поскольку пользователь опреде-ляет, что должно быть сделано, но не говорит, как именно желаемый ре-зультат должен быть достигнУТ. Предполагается, что реализация языков четвертого поколения будет в значительной мере основана на использовании компонентов высокого уровня, которые часто называют "инструментами четвертого поколения". Пользователю не потребуется определять все этапы выполнения программы, необходимые для решения поставленной задачи, а достаточно будет лишь определить нужные параметры, на основании которых упомянутые выше инструменты автоматически осуществят

генерацию прикладного приложения. Ожидается, что языки четвертого поколения позволят повысь производительность работы на порядок, но за счет ограничения типов задач, которые можно будет решать с их помощью. Выделяют следующие типы языков четвертого поколения:

- языки представления информации, например, языки запросов или генера¬торы отчетов;
- специализированные языки, например, языки электронных таблиц и баз данных;
- генераторы приложений, которые при создании приложений обес-печи¬вают определение, вставку, обновление или извлечение сведений из базы данных;
- языки очень высокого уровня, предназначенные для генерации кода приложений.

Преимущества и недостатки СУБД

СУБД обладают как многообещающими потенциальными преимуществами, так и недостатками.

Преимущества:

- Контроль за избыточностью данных.
- Непротиворечивость данных.
- Совместное использование данных.
- Поддержка целостности данных.
- Повышенная безопасность.
- Применение стандартов.
- Повышение эффективности с ростом масштабов системы.
- Возможность нахождения компромисса при противоречивых требованиях.
- Повышение доступности данных и их готовности к работе.
- Улучшение показателей производительности.
- Упрощение сопровождения системы за счет независимости от данных.
- Улучшенное управление параллельностью.
- Развитые службы резервного копирования и восстановления.

Контроль за избыточностью данных. Файловые системы неэкономно рас¬ходуют внешнюю память, сохраняя одни и те же данные в нескольких файлах. При использовании базы данных, наоборот, предпринимается попытка исключить избыточность данных

за счет интеграции файлов, чтобы избежать хранения нескольких копий одного и того же эле¬мента информации. Однако полностью избыточность информации в базах данных не исключается, а лишь контролируется ее степень. В одних случаях ключевые элементы дан¬ных необходимо дублировать для моделирования связей, а в других случаях некоторые данные потребуется дублировать из соображений повышения производительности систе¬мы.

Непротиворечивость данных. Устранение избыточности данных или контроль над ней позволяет сократить риск возникновения противоречивых состояний. Если элемент данных хранится в базе толь¬ко в одном экземпляре, то для изменения его значения потребуется выполнить только одну операцию обновления, причем новое значение станет доступным сразу всем пользователям базы данных. А если этот элемент данных с ведома системы хранится в базе данных в нескольких экземплярах, то такая система сможет следить за тем, чтобы копии не противоречили друг другу.

К сожалению, во многих современных СУБД та¬кой тип непротиворечивости данных автоматически не поддерживается.

Совместное использование данных в файловых системах доступ к данным со стороны многих пользователей (если это вообще возможно) контролируется операционной системой или программистом. В современных СУБД имеются средства для совместной работы многочисленных пользователей.

Можно создавать новые приложения на основе уже существующей информации и добавлять в БД только те данные, которые в настоящий момент еще не хранятся в ней, а не определять заново требования ко всем данным, необходимым новому приложе¬нию. Новые приложения могут также использовать такие предоставляемые типич¬ными СУБД функциональные возможности, как определение структур данных и управление доступом к данными, организация параллельной обработки и обеспече¬ние средств копирования/восстановления, исключив необходимость реализации этих функции со своей стороны.

Поддержка целостности данных. Целостность базы данных означает корректность и непротиворечивость хранимых в ней данных. Целостность обычно описывается с помощью ограничений, т.е. правил

под¬держки непротиворечивости, которые не должны нарушаться в базе данных. Огра-ниче¬ния можно применять к элементам данных внутри одной записи или к связям между за¬писями.

Повышенная безопасность. Безопасность базы данных заключается в защите базы данных от несанкциони¬рованного доступа со стороны польз-вателей. Без привлечения соответствующих мер безопасности интегрированные данные становятся более уязвимыми, чем дан¬ные в файловой системе. Однако интеграция позволяет АБД определить требуемую систему безопасности базы данных, а СУБД привести ее в действие. Система обеспечения безопасности может быть выражена в форме учетных имен и паролей для идентификации пользователей, которые зарегистрированы в этой базе данных. Доступ к данным со стороны зарегистрированного пользователя может быть ограничен только некоторыми операциями (извлечением, вставкой, обновлением и уда¬лением). Например, АБД может быть предоставлено право доступа ко всем данным в базе данных, менеджеру отделения компании — ко всем данным, которые отно¬сятся к его отделению, а инспектору отдела реализации — лишь ко всем данным о недвижимости, в результате чего он не будет иметь доступа к конфиденциальным данным, например, о зарплате сотрудников.

Применение стандартов. Интеграция позволяет АБД определять и применять необходимые стандарты. На¬пример, стандарты отдела и организации, государственные и международные стандар¬ты могут регламентировать формат данных при обмене ими между системами, соглашения об именах, форму представления документации, процедуры обновления и пра¬вила доступа.

Повышение эффективности с ростом масштабов системы. Комбинируя все рабочие данные организации в одной базе данных и создавая набор приложений, которые работают с одним источником данных, можно добиться существен¬ной экономии средств. В этом случае бюджет, который обычно выделялся каждому отде¬лу для разработки и поддержки их собственных файловых систем, можно объединить с бюджетами других отделов (с более низкой общей стоимостью), что позволит добиться по¬вышения эффективности при росте масштабов производства. Теперь объединенный бюд¬жет можно

будет использовать для приобретения оборудования той конфигурации, которая в большей степени отвечает потребностям организации. Например, она может состо¬ять из одного мощного компьютера или сети из небольших компьютеров.

Возможность нахождения компромисса для противоречивых требований. Потребности одних пользователей или отделов могут противоречить потребностям других пользователей. Поскольку база данных контролируется АБД, он может при¬нимать решения о проектировании и способе использования базы данных, при кото¬рых имеющиеся ресурсы всей организации в целом будут использоваться наилучшим образом. Эти решения обеспечивают оптимальную производительность для самых важных приложений, причем чаще всего за счет менее критичных.

Повышение доступности данных и их готовности к работе. Данные, которые пересекают границы отделов, в результате интеграции становятся непосредственно доступными конечным пользователям. Потенциально это повышает функциональность системы, что, например, может быть использовано для более каче¬ственного обслуживания конечных пользователей или клиентов организации. Во мно¬гих СУБД предусмотрены языки запросов или инструменты для создания отчетов, ко¬торые позволяют пользователям задавать непредусмотренные заранее вопросы и почти немедленно получать требуемую информацию на своих терминалах, не прибегая к помощи программиста, который для извлечения этой информации из базы данных должен был бы создать специальное программное обеспечение.

Улучшение показателей производительности. Как уже упоминалось выше, в СУБД предусмотрено много стандартных функций, вторые программист обычно должен самостоятельно реализовать в приложениях для файловых систем. На базовом уровне СУБД обеспечивает все низкоуровневые процеду¬ры работы с файлами, которую обычно выполняют приложения. Наличие этих процедур позволяет программисту сконцентрироваться на разработке более специальных, не¬обходимых пользователям функций, не заботясь о подробностях их воплощения на бо¬лее низком уровне. Во многих

СУБД также предусмотрена среда разработки четвертого поколения с инструментами, упрощающими создание приложении баз данных. Результатом является повышение производительности работы программистов и сокращение времени разработки новых приложений (с соответствующей экономией средств).

Упрощение сопровождения системы за счет независимости от данных. В файловых системах описания данных и логика доступа к данным встроены в каждое приложение, что делает программы зависимыми от данных. Для изменения структуры данных — например, для увеличения длины поля с адресом с 40 символов до 41 символа — или для изменения способа хранения данных на диске может потребоваться существенно преобразовать все программы, на которые эти изменения способны оказать влияние. В СУБД подход иной: описания данных отделены от при¬ложений, а потому приложения защищены от изменений в описаниях данных. Эта особенность называется независимостью от данных. Наличие независимости программ от данных значительно упрощает обслуживание и сопровождение приложений, работающих с базой данных.

Улучшенное управление параллельностью. В некоторых файловых системах при одновременном доступе к одному и тому же файлу двух пользователей может возникнуть конфликт двух запросов, результатом которого будет потеря информации или утрата ее целостности. В свою очередь, во многих СУБД предусмотрена возможность параллельного доступа к базе данных и гарантируется отсутствие подобных проблем.

Развитые службы резервного копирования и восстановления. Ответственность за обеспечение защиты данных от сбоев аппаратного и программного обеспечения в файловых системах возлагается на пользователя. Так, может потребо¬ваться каждую ночь выполнять резервное копирование данных. При этом в случае сбоя может быть восстановлена резервная копия, но результаты работы, выполненной после резервного копирования, будут утрачены, и данную работу потребуется выполнить за¬ново. В современных СУБД предусмотрены средства сокращения объема потерь информации от возникновения различных сбоев.

Недостатки:

- Сложность.
- Размер.
- Стоимость СУБД.
- Затраты на преобразование.
- Производительность.
- Более серьезные последствия при выходе системы из строя.

Сложность. Обеспечение функциональности, которой должна обладать каждая хорошая СУБД, сопровождается значительным усложнением программного обеспечения СУБД.

Чтобы воспользоваться всеми преимуществами СУБД, проектировщики и разработчики баз данных, администраторы данных и администраторы баз данных, а также конечные пользователи должны хорошо понимать функциональные возможности СУБД. Непонимание принципов работы системы может привести к неудачным результатам проектирования, что будет иметь самые серьезные последствия для всей организации.

Размер. Сложность и широта функциональных возможностей приводит к тому, что СУБД стано¬вится чрезвычайно сложным программным продуктом, который может занимать много места на диске и требовать большого объема оперативной памяти для эффективной работы.

Стоимость СУБД. В зависимости от имеющейся вычислительной среды и требуемых функциональ¬ных возможностей, стоимость СУБД может варьировать в очень широких пределах, как правило — чем больше пользователей системы, тем выше стоимость. Кроме того, следует учесть ежегодные расходы на сопровождение системы, которые составляют некоторый процент от ее об¬щей стоимости.

Затраты на преобразование. В некоторых ситуациях стоимость СУБД и дополнительного аппаратного обеспечения может оказаться несущественной по сравнению со стоимостью преобразования существую¬щих приложений для работы с новой СУБД и новым аппаратным обеспечением. Эти затраты также включают стоимость подготовки персонала для работы с новой системой, а также оплату

услуг специалистов, которые будут оказывать помощь в преобразовании и запуске новой системы.

Все это является одной из основных причин, по которой некоторые органи¬зации остаются сторонниками прежних систем и не хотят переходить к более современным технологиям управления базами данных. Термин традиционная система иногда используется для обозначения устаревших и, как правило, не самых лучших систем.

Производительность. Обычно файловая система создается для некоторых специализированных приложе¬ний, например, для оформления счетов, а потому ее производительность может быть весьма высока. Однако СУБД предназначены для решения более общих задач и обслуживания сразу нескольких приложений, а не какого-то одного из них. В результате многие приложения в новой среде будут работать не так быстро, как прежде.

Более серьезные последствия при выходе системы из строя. Централизация ресурсов повышает уязвимость системы. Поскольку работа всех пользователей и приложений зависит от готовности к работе СУБД, выход из строя одного из ее компонентов может привести к полному прекращению всей работы организации.

Лекция 6. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

Основная цель СУБД заключается в том, чтобы предложить пользователю абстрактное представление данных, скрыв конкретные особенности хранения и управления ими. Следовательно, отправной точкой при проектировании базы данных должно быть абстрактное и общее описание информационных потребностей, которые должны найти свое отражение в создаваемой базе данных: сущностей, атрибутов и связей.

Поскольку база данных является общим ресурсом, то каждому пользователю может потребоваться свое, отличное от других представление о характери¬стиках информации, сохраняемой в базе данных. Для удовлетворения этих потребно¬стей архитектура большинства современных коммерческих СУБД в той или иной степени строится на базе т. н. архитектуры ANSISPARC.

Трехуровневая архитектура, состоит из трех уровней абстракций: внешний, концептуальный и внутренний уров¬ни. Цель трехуровневой архитектуры заключается в отделении пользовательского представления базы данных от ее физического представления. Почему желательно выполнять такое разделение:

- Каждый пользователь должен иметь возможность обращаться к одним и тем же данным, используя свое собственное представление о них. Каждый пользователь должен иметь возможность изменять свое представление о данных, причем это изменение не должно оказывать влияния на других пользователей.
- Пользователи не должны непосредственно иметь дело с такими подробно¬стями физического хранения данных в базе, как индексирование, хеши¬рование и т.п. Иначе говоря, взаимодействие пользователя с базой не должно зависеть от особенностей хранения в ней данных.
- Администратор базы данных (АБД) должен иметь возможность изменять структуру хранения данных в базе, не оказывая влияния на пользовательские представления.

– Внутренняя структура базы данных не должна зависеть от таких изменений физических аспектов хранения информации. Нр, переключение на новое устройство хранения.
– АБД должен иметь возможность изменять концептуальную или глобальную структуру базы данных без какоголибо влияния на всех пользователей.

Уровень, на котором воспринимают данные пользователи, называется внешним уровнем (external level), тогда как СУБД и операционная система воспринимают данные на внутрен¬нем уровне (internal level). Именно на внутреннем уровне данные реально сохраняются с использованием структур и файловой организации. Концептуальный уровень (conceptual level) представления данных предназначен для отображения внешнего уровня на внутренний и обеспечения необходимой независимости друг от друга.

Например, поставлена задача проктирования БД по студентам для нашего ВУЗа. Первое, что для этого нужно — собрать информацию обо всех потенциальных пользователях (подразделениях ВУЗа) и о том, чтобы они хотели получить от будущей БД. Для простоты ограничимся двумя подразделениями: бухгалтерией и деканатом (рис. 6.1).

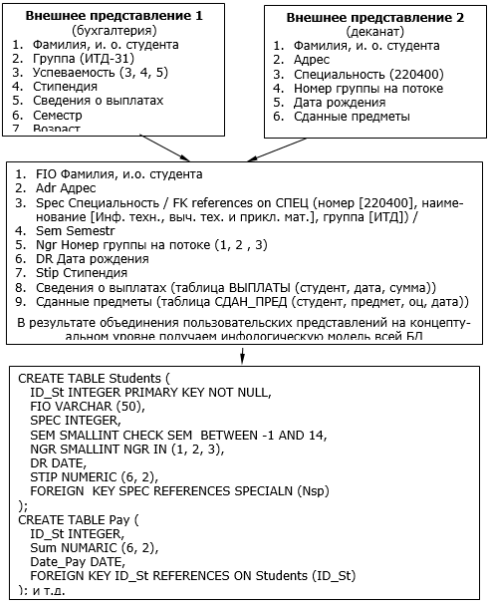


Рис. 6.1. Структура базы данных

Внешний уровень

Внешний уровень — это представление базы данных с точки зрения пользователей. Он описывает ту часть базы данных, которая относится к каждому пользователю.

Внешний уровень состоит из нескольких различных внешних представлений ба¬зы данных. Каждый пользователь имеет дело с представлением "реального мира", выраженным в наиболее удобной для него форме. Внешнее представление содержит только те сущности, атрибуты и связи "реального мира", которые интересны пользователю. Другие сущности, атрибуты или связи, которые ему неинтересны, также могут быть представлены в базе данных, но пользователь может даже не по¬дозревать об их существовании.

Помимо этого, различные представления могут поразному отображать одни и те же данные. Например, один пользователь может просматривать даты в формате (день, месяц, год), а другой — в формате (год, месяц, день). Некоторые представления могут включать производные или вычисляемые данные, которые не хранятся в базе данных как таковые, а создаются по мере надобности. Например, можно было бы орга¬низовать просмотр данных о возрасте преподавателей или студентов. Однако вряд ли стоит хранить эти сведения в базе данных, поскольку в таком случае их пришлось бы ежедневно обновлять. Вместо этого в базе данных хранятся даты рождения сотрудников, а возраст вычисляется. Представления могут также включать комбинированные или производные данные из нескольких объектов.

Концептуальный уровень

Концептуальный уровень — это обобщающее представление базы данных. Этот уровень описывает то, какие данные хранятся в базе данных, а также связи, существующие между ними.

Он является промежуточным уровнем в трехуровневой архитектуре. Этот уровень содержит логическую структуру всей базы данных (с точки зрения АБД). Фактически, это полное представление требований к данным со сторо¬ны организации, которое не зависит от любых

соображений относительно способа их хранения. На концептуальном уровне представлены следующие компоненты:

- все сущности, их атрибуты и связи;
- накладываемые на данные ограничения;
- семантическая информация о данных;
- информация о мерах обеспечения безопасности и поддержки целостности данных.

Концептуальный уровень поддерживает каждое внешнее представление, в том смысле, что любые доступные пользователю данные должны содержаться (или могут быть вычисле¬ны) на этом уровне. Однако этот уровень не содержит никаких сведений о методах хранения данных. Например, описание сущности должно содержать сведения о типах данных атрибутов (целочисленный, действительный или символьный) и их длине (количестве значащих цифр или максимальном количестве символов), но не должно включать сведений об организации хранения данных, например, об объеме занятого пространства в байтах.

Внутренний уровень

Внутренний уровень — физическое представление базы данных в компьютере. Этот уровень описывает, как информация хранится в базе данных.

Внутренний уровень описывает физическую реализацию базы данных и предна¬значен для достижения оптимальной производительности и обеспечения экономно¬го использования дискового пространства. Он содержит описание структур данных и организации отдельных файлов, используемых для хранения данных в ЗУ. На этом уровне осуществляется взаимодействие СУБД с ме¬тодами доступа операционной системы (вспомогательными функциями хранения и извлечения записей данных) с целью размещения данных на запоминающих уст¬ройствах, создания индексов, извлечения данных и т.д. На внутреннем уровне хранится следующая информация:

- распределение дискового пространства для хранения данных и индексов;

– описание подробностей сохранения записей (с указанием реальных размеров сохраняемых элементов данных);
– сведения о размещении записей;
– сведения о сжатии данных и выбранных методах их шифрования.

Физический уровень

Ниже внутреннего уровня находится физический уровень (physical level), кото¬рый контролируется операционной системой, но под руководством СУБД. Однако функции СУБД и операционной системы на физическом уровне не вполне четко разделены и могут варьироваться от системы к системе. В одних СУБД используются многие предусмотренные в данной операционной системе методы доступа, тогда как в других применяются только самые основные и реализована собственная файловая организация.

Основным назначением трехуровневой архитектуры является обеспечение независимости от данных, которая означает, что изменения на нижних уровнях никак не влияют на верхние уровни. Различают два типа независимости от данных: логическую и физическую.

Логическая независимость от данных означает полную защищенность внешних схем от изменений, вносимых в концептуальную схему.

Такие изменения концептуальной схемы, как добавление или удаление новых сущностей, атрибутов или связей, должны осуществляться без необходимости внесения изменений в уже существующие внешние схемы или переписывания прикладных программ. Ясно, что пользователи, для которых эти изменения предназначались, должны знать о них, но очень важно, чтобы другие пользователи даже не подозревали об этом.

Физическая независимость от данных означает защищенность концептуальной схемы от изменений, вносимых во внутреннюю схему.

Такие изменения внутренней схемы, как использование различных файловых систем или структур хранения, разных устройств хранения, модификация индексов или хеширование, должны осуществляться без необходимости внесения изменений в концептуальную или внешнюю

схемы. Пользователи заметить изменения только в общей производительности системы.

Общий обзор средств Delphi

В состав Delphi 5 Enterprise входят следующие средства для разработки и эксплуатации приложений, использующих базы данных.

BDE (Borland Database Engine) — машина баз данных Borland

Представляет собой набор DLLбиблиотек, обеспечивающих низкоуровневый доступ к локальным и клиентсерверным БД. Должна устанавливаться на каждом компьютере, который использует приложения для работы с БД, написанные на Delphi (за исключением облегченных клиентов в трехзвенной архитектуре).

SQL Links. Драйверы для работы с удаленными серверами данных, такими как Sybase, MS SQL Server, Oracle. Для работы с «родным» SQLсервером InterBase устанавливать SQL Links нет необходимости. Доступ к таблицам локальных СУБД типа Paradox и dBase также осуществляется BDE напрямую, без использования SQL Links.

BDE Administrator. Утилита для установки псевдонимов (имен) баз данных, параметров БД и драйверов баз данных на конкретном компьютере. При работе с БД из приложения, созданного с помощью Delphi, доступ к базе данных производится по ее псевдониму. Параметры определяемой псевдонимом БД действуют только для этой БД; параметры, установленные для драйвера БД, действуют для всех баз данных, использующих драйвер. Кроме того, в утилите BDE Administrator можно произвести установку таких общих для всех БД параметров, как формат даты и времени, форматы представления числовых значений, используемый языковый драйвер и т. д.

Database Desktop (DBD). Средство для создания, изменения и просмотра БД. Эта утилита прежде всего ориентирована на работу с таблицами локальных СУБД. В ряде случаев может использоваться и для работы с таблицами распределенных СУБД. Например, с помощью DBD можно с некоторыми ограничениями создавать и просматривать таблицы баз данных, работающих под управлением промышленных серверов InterBase, MS SQL Server, Oracle и некоторых других. DBD

предоставляет программисту возможность сформировать запрос к БД методом QBE (Query By Example запрос по образцу).

SQL Explorer. Универсальная утилита, совмещающая многие функции BDE Administrator и DBD. С ее помощью можно создавать и просматривать псевдонимы БД, просматривать структуры и содержимое таблиц БД, формировать запросы к БД на языке SQL, создавать словари данных (шаблоны полей таблиц).

SQL Monitor. Средство для трассировки выполнения SQLзапросов.

VisiBroker. Комплекс программных средств фирмы VisiGen (ныне входит в состав Inprise) для поддержки технологии CORBA.

Datapump. Средство для перемещения данных между БД различных типов (например, при переходе от локальных систем к распределенным).

Data Dictionary. Словарь данных. Предназначен для хранения атрибутов полей таблиц БД отдельно от самих БД. Информация о полях может использоваться различными приложениями.

Data Module. Невизуальные компоненты DataModule применяются для централизованного хранения наборов данных в приложении, работающем с БД. Одним из главных удобств использования Data Module является возможность связывания с каждым набором данных правил по управлению данными — бизнесправил. Бизнесправила определяют реакцию системы на добавление, изменение, удаление данных и реализуют блокировку действий, которые могут разрушить ссылочную или смысловую целостность БД.

Невизуальные компоненты для работы с БД служат для соединения приложения с таблицами БД в локальных и распределенных системах. Они расположены на страницах Data Access и Midas палитры компонентов (на странице Data Access собраны компоненты как для локальных, так и для распределенных систем, а на странице Midas только для распределенных систем). С помощью невизуальных компонентов осуществляется подключение к базам данных, формирование запросов к ним, манипулирование таблицами, создание клиентов и серверов в трехзвенной архитектуре.

Визуальные компоненты для работы с БД. Визуальные компоненты предназначены для визуализации записей наборов данных или их

отдельных полей. Эти компоненты расположены на странице Data Controls палитры компонентов. Они служат основным инструментом разработки пользовательского интерфейса доступа к данным.

Компоненты для построения отчетов. На странице QReport палитры компонентов размещены компоненты для построения отчетов. Отчеты играют важную роль: на основании запросов к БД они создают для пользователя нужные ему документы.

Особенности программ для работы с БД

Характерной особенностью созданных с помощью Delphi программ для работы с БД является непременное использование в них BDE, которая осуществляет роль связующего моста между программой и БД. BDE берет на себя всю низкоуровневую работу по обеспечению клиентской программы нужными ей данными, поэтому в самом общем случае взаимодействие программы с данными происходит так, как показано на рис 6.2.

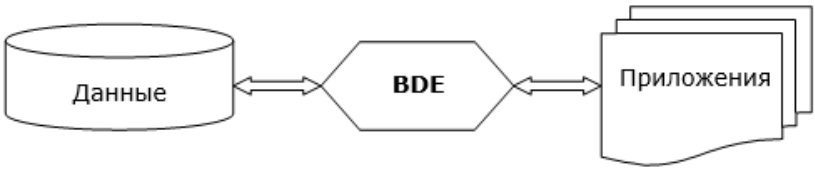


Рис. 6.2. Схема взаимодействия программы с данными

BDE не является частью программы. В зависимости от типа СУБД она может размещаться на машине клиента или сервера. Лицензионное соглашение разрешает вам устанавливать это программное средство бесплатно вместе с распространяемыми программами. Поскольку сама BDE содержится в нескольких каталогах и должна регистрироваться в реестре Windows 32, создание дистрибутивных носителей распространяемых программ становится не совсем простым делом. Существенно упростить эту процедуру помогает утилита InstallShield Expreess for Delphi, поставляемая на одном компактдиске с Delphi.

Программа может использовать низкоуровневый интерфейс функций API BDE для непосредственного обращения к данным, однако обычно между ней и BDE располагается слой компонентов,

существенно упрощающих разработку программ. Как уже говорилось, невизуальные компоненты осуществляют непосредственную работу с BDE, и три из них (TTable, TQuery и TStoredProc) служат наборами данных, в то время как визуальные компоненты отображают поставляемые им данные и служат для создания удобного интерфейса пользователя. Между наборами данных и визуальными компонентами обязательно располагаются компоненты TDataSource, играющие роль клапанов, открывающих или закрывающих потоки данных, которыми обмениваются источники с визуальными компонентами.

В версии Delphi 5 появились средства, позволяющие работать с данными без BDE, — это ActiveX Data Objects (ADO) и InterBase Express (IPX).

Лекция 7. РЕЛЯЦИОННАЯ АЛГЕБРА И РЕЛЯЦИОННОЕ ИСЧИСЛЕНИЕ

РА использует процедурный язык обработки таблиц (язык, обеспечивающий пошаговое решение задач), а РИ — непроцедурный язык создания запросов (язык, позволяющий формулировать, что нужно сделать, а не как этого добиться)

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

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

Механизмы РА и РИ эквивалентны, т.е. для любого допустимого выражения РА можно построить эквивалентную (т.е. производящую такой же результат) формулу РИ и наоборот.

Почему же в реляционной модели данных присутствуют оба эти механизма?

В реляционной модели присутствуют оба эти механизма потому, что они различаются уровнем процедурности.

Реляционная алгебра

Основная идея РА состоит в том, что коль скоро отношения являются множествами, то средства манипулирования отношениями могут базироваться на традиционных теоретикомножественных

операциях, дополненных некоторыми специальными операциями, специфичными для баз данных.

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

В состав теоретикомножественных операций входят операции:

- объединения отношений;
- пересечения отношений;
- взятия разности отношений;
- прямого произведения отношений.

Специальные реляционные операции включают:

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

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

Теоретико-множественные операций

При выполнении операции объединения двух отношений производится отношение, включающее все кортежи, входящие хотя бы в одно из отношенийоперандов.

Операция пересечения двух отношений производит отношение, включающее все кортежи, входящие в оба отношенияоперанда.

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. Примеры приведены на рис. 7.1.

Рис. 7.1. Теоретико-множественные операции

Начнем с операции объединения (все, что касается объединения, переносится на операции пересечения и взятия разности). Смысл операции объединения в реляционной алгебре в целом остается теоретикомножественным. Но если в теории множеств операция объединения осмысленна для любых двух множествоперандов, то в случае реляционной алгебры результатом операции объединения должно являться отношение.

Если допустить в реляционной алгебре возможность теоретикомножественного объединения произвольных двух отношений (с разными схемами), то, конечно, результатом операции будет множество, но множество разнотипных кортежей, т.е. не отношение. Если исходить из требования замкнутости реляционной алгебры относительно понятия отношения, то такая операция объединения является бессмысленной.

Все эти соображения приводят к появлению понятия совместимости отношений по объединению: два отношения совместимы по объединению в том и только в том случае, когда обладают

одинаковыми заголовками. Более точно, это означает, что в заголовках обоих отношений содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене.

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

Если два отношения совместимы по объединению, кроме имен атрибутов, то до выполнения операции типа объединения эти отношения можно сделать полностью совместимыми по объединению путем применения операции переименования.

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

При выполнении прямого произведения двух отношений производится отношение, кортежи которого являются конкатенацией (сцеплением) кортежей первого и второго операндов.

В теории множеств прямое произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств.

{A, B, C} {X, Y, Z}

{(A,X); (A,Y), (A,Z), (B,X); (B,Y), (B,Z), (C,X); (C,Y), (C,Z), }

Поскольку отношения являются множествами, то и для любых двух отношений возможно получение прямого произведения. Но результат не будет отношением! Элементами результата будут являться не кортежи, а пары кортежей.

Поэтому в реляционной алгебре используется специализированная форма операции взятия прямого произведения — расширенное прямое произведение отношений. При взятии расширенного прямого

произведения двух отношений элементом результирующего отношения является кортеж, являющийся конкатенацией (или слиянием) одного кортежа первого отношения и одного кортежа второго отношения (рис. 7.2).

Исходные отношения

R1

A	B
341	12
275	34
383	67

R2

C	D
1	6
2	5

Прямое расширенное произведение *R1×R2*

A	B	C	D
341	12	1	6
341	12	2	5
275	34	1	6
275	34	2	5
383	67	1	6
383	67	2	5

Рис. 7.2. Прямое расширенное произведение

Но теперь возникает второй вопрос — как получить корректно сформированный заголовок отношениярезультата?

Очевидно, что проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами.

Эти соображения приводят к появлению понятия совместимости по взятию расширенного прямого произведения:

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

простым способом переименования является уточнение имени атрибута именем отношения — R.A

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

По поводу теоретикомножественных операций реляционной алгебры следует еще заметить, что все четыре операции являются ассоциативными. Т. е., если обозначить через OP любую из четырех операций, то

$$(A\ OP\ B)\ OP\ C = A\ OP\ (B\ OP\ C),$$

и, следовательно, без введения двусмысленности можно писать A OP B OP C (A, B и C — отношения, обладающие свойствами, требуемыми для корректного выполнения соответствующей операции).

Все операции, кроме взятия разности, являются коммутативными, т.е.

$$A\ OP\ B = B\ OP\ A.$$

Специальные реляционные операции

Отбор. Другое название операции — выборка или ограничение отношения по некоторому условию.

Обозначается σ_C (R), где C — некоторое условие отбора, R — отношение, к которому применяется операция.

Результат операции — отношение, которое включает в себя подмножество кортежей исходного отношения R, удовлетворяющих условию C.

№	ФИО	Д/р	Пол
001	Иванов	06.04.90	M
002	Семенова	10.03.89	F
003	Петров	27.10.70	M
004	Васина	30.11.84	F
005	Маслова	01.01.78	F
006	Сидоров	23.02.85	M
007	Анисимова	15.05.92	F

$\sigma_{(Д/р>=01.01.1984\ AND\ Пол='F')}(R)$

№	ФИО	Д/р	Пол
002	Семенова	10.03.89	F
004	Васина	30.11.82	F
007	Анисимова	15.05.92	F

Проекция — выборка из отношения подмножества атрибутов.

Результат операции — отношение, содержащее только указанные атрибуты. Обозначается πA1, A2, ..., An A1, A2, ..., AN (R), где A1, A2, ..., An — набор желаемых атрибутов, R — исходное отношение (рис. 7.3).

$\pi_{ФИО,\ Д/р}\ (R)$

ФИО	Д/р
Иванов	06.04.90
Семенова	10.03.89
Петров	27.10.70
Васина	30.11.84
Маслова	01.01.78
Сидоров	23.02.85
Анисимова	15.05.92

$\pi_{ФИО}\ (\sigma_{(Д/р>=01.01.1984\ AND\ Пол='F')}(R))$

ФИО
Семенова
Васина
Анисимова

Рис. 7.3. Операция проекция

При естественном соединении двух отношений образуется результирующее отношение, кортежи которого являются конкатенацией тех кортежей первого и второго отношений, которые совпадают по некоторым атрибутам.

Обозначается как R1⊳⊲R2. Пусть A1, A2, ... An — атрибуты, входящие и в R1 и R2, тогда кортежи из R1 И R2 соединяются в пару, если они совпадают по каждому из атрибутов A1, A2, ... An (рис 7.4).

Исходные отношения

R1

№	ФИО	Д/р
001	Иванов	06.04.90
002	Петров	10.03.89
003	Сидорова	27.10.70
004	Маслов	06.07.90
005	Иванов	06.04.90

R2

ФИО	Д/р	Адрес
Маслов	10.12.89	Москва
Иванов	06.04.90	СПб
Сидорова	27.10.70	Калуга
Фролов	05.07.89	Калуга

Естественное соединение *R1*⊳⊲*R2*.

№	ФИО	Д/р	Адрес
001	Иванов	06.04.90	СПб
003	Сидорова	27.10.70	Калуга
005	Иванов	06.04.90	СПб

Рис. 7.4. Естественное соединение

Результирующий набор атрибутов получается объединением атрибутов R1 и R2.

Если кортеж не соединяется ни с одним из кортежей другого отношения, то он называется висящим.

Основной смысл операции естественного соединения — возможность восстановления сложной сущности, декомпозированной по причине требования первой нормальной формы. Операция естественного соединения не включается прямо в состав набора операций реляционной алгебры, но она имеет очень важное практическое значение.

При соединении двух отношений по некоторому условию образуется результирующее отношение, кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию. Обозначается $R1 \triangleright\triangleleft_C R2$

Результат получается следующим образом: берется прямое произведение двух отношений R1 и R2, затем к полученному отношению применяется операция отбора, т.е.

$$R1 \triangleright\triangleleft_C R2 = \sigma_C(R1 \times R2)$$

При этом в случае наличия одноименных атрибутов в R1 и R2 также применяется операция переименования при помощи префикса-имени отношения (рис. 7.5).

Исходные отношения

R1

A	B
341	12
275	15
383	67

R2

B	D
17	6
12	5

Соединение по условию $R1 \triangleright\!\triangleleft_{A>300} R2$

A	R1.B	R2.B	D
341	12	17	6
341	12	12	5
275	15	17	6
275	15	12	5
383	67	17	6
383	67	12	5

Соединение по условию $R1 \triangleright\!\triangleleft_{R1.B>R2.B} R2$

A	R1.B	R2.B	D
341	12	17	6
341	12	12	5
275	15	17	6
275	15	12	5
383	67	17	6
383	67	12	5

Рис. 7.5. Соеднине по условию

В общем случае применение соединения по условию существенно уменьшит мощность результата промежуточного прямого произведения отношенийоперандов только в том случае, когда условие проверяется над именами атрибутов разных отношенийоперандов (а не атрибута одного из отношений и константы). Поэтому на практике обычно считают реальными операциями соединения именно те операции, которые основываются на условии соединения по атрибутам.

Имеется важный частный случай соединения по условию — эквисоединение. Операция соединения называется операцией

эквисоединения, если условие соединения имеет вид (a = b), где a и b — атрибуты разных отношений соединения. Этот случай важен потому, что (a) он часто встречается на практике, и (b) для него существуют эффективные алгоритмы реализации.

Соединение по условию $R1 \triangleright\!\triangleleft_{R1.B=R2.B} R2$

A	R1.B	R2.B	D
341	12	17	6
341	12	12	5
275	15	17	6
275	15	12	5
383	67	17	6
383	67	12	5

Если взять проекцию полученного отношения по атрибутам A, B (любой) и D, то получим результат естественного соединения $R1 \triangleright\!\triangleleft R2$, т.е.

$$R1 \triangleright\!\triangleleft R2 = \pi_L (\sigma_C(R1 \times R2)),$$

где L – результат объединения атрибутов отношений R1 и R2, C — условие равенства атрибутов составляющих пересечение атрибутов отношений R1 и R2.

У операции реляционного деления два операнда — бинарное и унарное отношения. Результирующее отношение состоит из одноатрибутных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) совпадает со множеством значений второго операнда.

Эта операция наименее очевидна из всех операций реляционной алгебры и поэтому нуждается в более подробном объяснении. Пусть заданы два отношения — A с заголовком {a1, a2, ..., an, b1, b2, ..., bm} и B с заголовком {b1, b2, ..., bm}. Будем считать, что атрибут bi отношения A и атрибут bi отношения B не только обладают одним и тем же именем, но и определены на одном и том же домене. Назовем множество атрибутов {aj} составным атрибутом a, а множество атрибутов {bj} — составным атрибутом b. После этого будем говорить

о реляционном делении бинарного отношения A(a, b) на унарное отношение B(b).

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

Например:

Исходные отношения

ИЗУЧЕНИЕ ПРЕДМЕТА

ФИО	ПРЕДМЕТ
Иванов	Физика
Петров	Физика
Сидоров	Математика
Иванов	Химия
Семенов	Химия
Петров	Математика
Маслов	Физика
Петров	Химия
Иванов	Математика

ПРЕДМЕТЫ

ПРЕДМЕТ
Физика
Математика
Химия

Результат операции деления состоит из строк одной таблицы, соответствующих каждой строке другой таблицы.

ФИО
Иванов

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.

Реляционное исчисление

Механизмы РА и РИ эквивалентны, т.е. для любого допустимого выражения РА можно построить эквивалентную (т.е. производящую такой же результат) формулу РИ и наоборот.

Базисными понятиями реляционного исчисления являются понятие переменной с определенной для нее областью допусти¬мых значений и

понятие правильно построенной формулы, опи¬рающейся на переменные и специальные функции.

В следующем примере имеется два отношения: СТУДЕНТЫ и ГРУППЫ:

Исходные отношения

STUDENTS

ID_S	FIO	ID_GR	...
1	Иванов	ИТД-51	
2	Петров	ЭВМ-51	
3	Сидоров	ЭВМ-52	
4	Смирнова	ИТД-51	
5	Волков	ЭВМ-51	
6	Зайцева	ИТД-52	
7	Медведев	ИТД-51	
8	Плюшкин	ИТД-52	
9	Блинова	ЭВМ-51	
	...	...	

GROUP

ID_GR	KOL	ID_STAR
ИТД-51	25	4
ИТД-52	26	8
ЭВМ-51	24	5
ЭВМ-52	26	3

Запрос: кто из студентов учится в группе ИТД-51.

В РА этот запрос выражается при помощи двух операций: проекции и отбора:

$$\pi_{FIO}(\sigma_{ID\_GR='ИТД-51'}(STUDENTS))$$

Формула РИ, производящая тот же результат:

{s.FIO : s IN STUDENTS AND s.ID_GR=’ИТД51’}

Фиг. скобки означают, что ответом на запрос будет множество значений данных.

s — это переменная, обозначающая произвольную строку.

s.FIO — целевой список, определяющий атрибуты таблицы решения.

Выражение s IN STUDENTS говорит о том, что строка берется из таблицы S (СТУДЕНТЫ).

s.ID_GR=’ИТД51’. Это условие говорит о том, что атрибут ID_GR строки s должен быть равен ’ИТД51’.

s IN STUDENTS AND s.ID_GR=’ИТД51’ — это определяющее выражение, ограничивающее вхождение элементов в таблицу решения.

В целом запрос выполняется примерно так: система просматривает строки таблицы STUDENTS одну за другой. Переменной s присваивается значение очередной строки. Проверяется истинность

определяющего выражения. Если оно истинно, строка помещается в результирующую таблицу, если ложно, то нет.

Целевой список может состоять из нескольких атрибутов.

{s.FIO, s.DR : s IN STUDENTS AND s.ID_GR=’ИТД51’}

Определяющие выражения могут строится при помощи квантора существования и квантора всеобщности.

Квантор существования означает, что существует хотя бы один экземпляр чеголибо. В РИ он используется для задания условия того, что определенный тип строк в таблице существует.

Предположим, что необходимо узнать име¬на студентов, являющихся ста¬ростами групп с количеством студентов больше 25.

Если бы для формулировки такого запроса использовалась реляционная алгебра, то мы получили бы алгебраическое выражение:

$$\pi(\sigma(S \rhd\lhd_{STUDENTS.ID\_S=GROUP.ID\_STAR} G)_{KOL>25})_{FIO}$$

которое читалось бы, например, следующим образом:

– выполнить соединение отношений СТУДЕНТЫ и ГРУППЫ по условию STUDENTS.ID_S=GROUP.ID_STAR. Полученное отношение:

ID_S	FIO	STUDENTS .ID_GR	GROUP.ID_ GR	KOL	ID_STAR	...
4	Смирнова	ИТД-51	ИТД-51	25	4	
8	Плюшкин	ИТД-52	ИТД-52	26	8	
5	Волков	ЭВМ-51	ЭВМ-51	24	5	
3	Сидоров	ЭВМ-52	ЭВМ-52	26	3	
	...	...				

– ограничить полученное отношение по условию KOL>25;

ID_S	FIO	STUDENTS. ID_GR	GROUP.ID_ GR	KOL	ID_STAR	...
8	Плюшкин	ИТД-52	ИТД-52	26	8	
3	Сидоров	ЭВМ-52	ЭВМ-52	26	3	
	...	...				

– спроецировать результат предыдущей операции на атрибут FIO:

FIO
Плюшкин
Сидоров
...

Здесь пошагово сформулирована последовательность выпол¬нения запроса к БД, каждый из которых соответствует одной ре¬ляционной операции. Если же сформулировать тот же запрос с использованием реляционного исчисления, то мы получили бы формулу, которую можно было бы прочитать, например, сле¬дующим образом:

Выдать FIO для таких студентов, чтобы существовала группа с таким же значением ID_STAR, что и ID_S и значением KOL больше 25.

{s.FIO: s IN STUDENTS and EXISTS g IN
GROUP (s.ID_S=g.ID_STAR and g.KOL>25)}

Во второй формулировке мы указали лишь характеристики результирующего отношения, но ничего не сказали о способе его формирования. В этом случае СУБД должна сама решить, что за операции и в каком порядке нужно выполнить над отношениями СТУДЕНТЫ и ГРУППЫ. Оба рассмотренных в примере способа на самом деле эквивалентны, и существуют не очень сложные правила преобразования одного в другой.

Квантор всеобщности означает, что некоторое условие истинно для каждой строки.

{s1.FIO: s1 IN STUDENTS and FORALL s2 IN STUDENTS
(s1.STIP>=s2.STIP)}

В зависимости от того, что является областью определения переменной, различаются исчисление кортежей и исчисление доменов. В исчислении кортежей областями определения пере¬менных являются отношения БД, то есть допустимым значением каждой переменной является кортеж некоторого отношения. В исчислении доменов областями определения переменных явля¬ются домены, на которых определены атрибуты отношений БД, то есть допустимым значением каждой переменной является зна¬чение некоторого домена.

В исчислении доменов областью определения переменных яв¬ляются не отношения, а домены. Применительно к БД СТУДЕНТЫ-ГРУППЫ можно говорить, например, о доменных переменных ИМЯ (значения домена — допустимые имена) или НОМСТУД (значения домена — допустимые номера студентов)

Основным отличием исчисления доменов от исчисления кортежей является наличие дополнительного набора предикатов (см. ниже), позволяющих выражать так называемые усло¬вия членства. Если R — это nарное отношение с атрибутами a1, a2,... , an, то условие членства имеет вид:

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

где vij — это либо литерально задаваемая константа, либо имя доменной переменной. Условие членства принимает значение ИС¬ТИНА только в том случае, если в отношении R существует кор¬теж, содержащий соответствующие значения указанных атрибутов. Если vij — константа, то на атрибут aij,, задается жесткое усло¬вие, не зависящее от текущих значений доменных переменных, если же vij — имя доменной переменной, то условие членства мо¬жет принимать различные значения при разных значениях этой переменной.

Несколько слов о предикатах. Предикатом принято называть некую логическую функцию, которая для некоторого аргумента возвращает значение ИСТИНА или ЛОЖЬ. Отношение может быть рассмотрено как предикат с аргументами, являющимися атрибутами рассматриваемого отношения. Если заданный кон¬кретный набор кортежей присутствует в отношении, то предикат выдаст истинный результат, в противном случае — ложный.

Во всех остальных отношениях формулы и выражения исчисления доменов выглядят похожими на формулы и выражения исчисления кортежей. Реляционное исчисление доменов положено в основу большинства языков запросов, основанных на использовании форм.

Лекция 8. ЯЗЫК SQL

Первое, с чего структурно надо бы начать изучение SQL — это с определения данных (таблиц, индексов, ограничений и т.д.), но часть DDL более требовательна к синтаксису и, следовательно, должна изучатся на примере конкретной СУБД и даже конкретной версии СУБД. В то время как часть DML не так сильно отличается в различных реализациях языка SQL.

В качестве примера будем разбирать предметную область, условно называемую КИНО. Структура таблиц и БД в целом может изменятся в целях наглядности выполнения запросов.

Movie {Фильм (идентификатор, название, идентификатор продюсера (FK), год выпуска, продолжительность фильма в минутах, жанр (комедия, мелодрама, боевик))}

ID_M	Title	ID_Pr	Year	Len	Kind
1	Джентльмены удачи	4	1980	98	комедия
2	Операция "Ы"	2	1979	91	комедия
3	Гараж	3	1978	103	комедия
4	Война и мир	9	1973		боевик
5	Бандитский Петербург	3	1999	255	боевик
6	Москва слезам не верит	7	1984	180	мелодрама
7	Ширли-мырли	7	1997	241	комедия
8	Иван Васильевич меняет профессию	2	1970	104	комедия
9	Гусарская баллада	3	1970	135	
10	Гардемарины вперед	6	1991	255	
11	Они сражались за Родину	8	1970	201	боевик
12	Служебный роман	3	1981	195	мелодрама
13	Карнавал	7	1983	204	комедия
14	Бриллиантовая рука	2	1976	209	комедия
15	Соломенная шляпка	3	1970		
16	Девчата	4	1975	102	комедия
17	Зимняя вишня	5	1989	255	мелодрама
18	Обыкновенное чудо	5	1986	201	

Prod {Продюсер (идентификатор, ФИО, адрес, дата рождения, доход)}

ID_Pr	FIO	Adres	Bd	Money
1	Л. Гайдай	г. Москва, п-т Мира-2-234	04.09.1963	95 920
2	Г. Данелия	г. С.-Петербург, ул. Жукова-4-2	07.02.1954	71 540
3	Э. Рязанов	г. Киев, ул. Московская-7	15.04.1969	85 790
4	С. Михалков	п. Переделкино, ул. Труда-23	16.02.1970	186 830
5	М. Захаров	г. С.-Петербург, ул. Маркса-15	16.04.1932	92 050
6	С. Дружинина	г. Екатеринбург, ул. Разина-4-1	04.02.1973	83 960
7	О. Меньшов	г. Москва, п-к Якимовский-43-2	24.07.1946	141 150
8	В. Шукшин	п. Одинцово, ул. Луговая-5-78	05.02.1943	44 210
9	С. Бондарчук	п. Киевский, ул. Северная-12-1	22.05.1958	96 700

Star {Актер (идентификатор, имя, адрес, дата рождения, доход)}

ID_St	FIO	Adres	Bd	Money
1	Е. Леонов	г. Москва, ул. Арбат-34-2	26.02.1961	80 860
2	Ю. Никулин	г. Москва, п-т Мира-72-1	09.02.1968	79 240
3	И. Муравьева	г. Москва, ул. Ташкентская-3-2	12.01.1963	55 280
4	Л. Голубкина	г. Москва, ул. Песчаная-53-122	26.11.1975	75 110
5	Е. Моргунов	г. С.-Петербург, ул. Киевская-5-4	30.06.1975	49 980
6	А. Миронов	г. Москва, ул. Песчаная-53-122	23.11.1969	67 240
7	О. Янковский	г. Москва, б-р Гоголя-122-132	11.11.1952	76 240
8	О. Дроздова	г. С.-Петербург, ул. Маркса-1-1	27.07.1971	69 270

9	О. Меньшов	г. Москва, п-к Якимовский-43-2	10.05.1961	97 520
10	Л. Ахеджакова	г. Москва, ул. Тверская-22-12	19.07.1978	62 000
11	Г. Вицин	г. Москва, ул. Абельмана-23-23	10.11.1960	59 480
12	В. Алентова	г. Москва, п-к Якимовский-43-2	05.08.1938	83 430
13	С. Михалков	п. Переделкино, ул. Труда-23	29.05.1968	99 960
14	В. Соломин	г. С.-Петербург, п-т Литейный-2-9	02.07.1973	78 740
15	В. Тихонов	г. Москва, п-т Ленинский-38-282	15.05.1945	98 980
16	В. Ливанов	г. Екатеринбург, ул. Баумана-2-29	27.04.1960	50 320
17	Ю. Яковлев	г. Москва, ул. Таганская-38-28	09.05.1961	68 780
18	Н. Крачковская	г. Москва, п-т Мира-282-298	15.09.1947	87 710
20	А. Папанов	г. Москва, ул. Киевская-83-28	11.10.1977	63 740
21	Е. Сафронова	г. С.-Петербург, ул. Пушкина-2-11	20.11.1936	97 680
22	Н. Русланова	г. Москва, пл. Маяковского-1-5	27.08.1965	90 470
23	Л. Удовиченко	г. С.-Петербург, ул. Гоголя-9-1	18.11.1953	74 150
24	И. Розанова	г. Москва, ул. М. Жукова-89-99	21.01.1937	77 740
25	М. Боярский	г. С.-Петербург, п-к Малый-1-9	09.01.1975	68 240
26	Н. Гундарева	г. Москва, проезд Бабеля-86-46	20.06.1969	86 520
27	Д. Певцов	г. С.-Петербург, ул. Маркса-1-1	09.04.1957	98 270
28	В. Шукшин	п. Одинцово, ул. Луговая-5-78	13.11.1972	99 960
29	А. Мягков	г. Москва, ул. Садовое кольцо-7-3	26.05.1952	59 740
30	А. Фрейндлих	г. С.-Петербург, ул. Перова-8-2	21.08.1974	53 400

StarIn {Таблица для реализации связи «многие ко многим», между таблицами ФИЛЬМ и АКТЕР}

ID_St	ID_M
1	1
1	18
2	2
2	11
2	14
3	13
4	9
5	2
6	14
6	15
6	18
7	18

ID_St	ID_M
8	5
9	7
10	3
10	12
11	1
11	2
12	6
12	7
14	17
15	4
15	11
17	8

ID_St	ID_M
17	9
17	13
18	8
20	14
21	17
25	10
26	10
27	5
28	11
29	3
29	12
30	12

Общая семантика оператора SELECT

Оператор, являющийся основным в языке SQL — SELECT. Математической основой для него служат такие операции РА как проекция, выборка, произведение, отбор, соединение.

Общая семантика оператора SELECT:

SELECT [DISTINCT | ALL] { * | [col_expr [AS <new_name>]] [, ...] }

FROM <имя_таблицы> [alias] [, ...]

[WHERE <условие>]

[GROUP BY <список_столбцов> [HAVING <условие>]]

[ORDER BY <список_столбцов>]

Прописными буквами (SELECT) написаны зарезервированные слова. Строчные — идентификаторы пользователя. Квадратные скобки означают необязательность. Фигурные — обязательность. Знак ‘|’ – ‘или’. Многоточие говорит о возможности повторения констукции.

col_expr имя столбца или выражение из нескольких имен, которое после выполнения запроса станет столбцом результирующей таблицы и этот новый столбец может получить имя new_name.

table_name имя существующей в БД таблицы, которая в пределах данного запроса может получить псевдоним alias.

FROM – определяются имена используемой таблицы или построение нескольких таблиц, в этом случае берется их произведение.

WHERE – отбор строк полученной таблицы по условию

GROUP BY – группировка строк, имеющих одинаковые значения кл атрибута

HAVING – в соответствии с условием фильтруются группы записей.

ORDER BY – определяется порядок следования строк результата.

Порядок предложений при написании оператора SELECT не может быть изменен.

Пример простейшего запроса:

```
SELECT Title
FROM Movie;
```

Результатом данного запроса будет таблица, содержащая все строки таблицы Movie и только один столбец из нее — Title. Нетрудно убедится, что этот запрос соответствует операции проекции РА: πTitle(Movie).

Чтобы выбрать несколько столбцов из таблицы нужно в предложении SELECT указать их через запятую.

```
SELECT Title, Len
FROM Movie;
```

Если требуется перечисление всех атрибутов в том же порядке, то достаточно указать *:

```
SELECT *
FROM Movie;
```

Кроме того, можно в качестве столбцов будущей таблицы указывать выражения и давать им новые имена при помощи слова AS:

```
SELECT Title, Len/60 AS Len_In_Hours
FROM Movie;
```

Такие поля называют вычисляемыми или производными. В том случае, если не указано имя для вычисляемого столбца, он получает системное наименование, нр, Expr1001, Col1.

Ключевое слово DISTINCT служит для исключения из результатов повторяющихся значений. Нр, SELECT Kind FROM Movie; даст просто

проекцию по данному столбцу, а данный запрос вернет таблицу из четырех строк — различных жанров, в том числе и неопределенного.

Выбор необходимых строк из таблицы осуществляется при помощи предложения WHERE, за которым следует условие отбора.

Для поиска нечетких значений (как правило для строк) можно пользоваться шаблонами: LIKE (NOT LIKE).

При этом используются следующие символы:

% (*) — означает последовательность из нуля или более символов;

_ (?) — подчеркивание означает любой одиночный символ.

Нр, выбрать всех актеров проживающих в Москве:

SELECT Fio, Adres

FROM Star

WHERE Adres LIKE ‘г. Москва%’;

Adres LIKE ‘г. Москва%’ – означает, что все символы после ‘г. Москва’ не рассматриваются при решении условия.

Adres NOT LIKE ‘г. Москва%’ – означает, условию удовлетворяют строки, не начинающиеся с ‘г. Москва’.

Adres LIKE ‘М__’ – означает, что строка, удовлетворяющая условию, должна состоять строго из 3х символов и начинаться с ‘М’.

Adres LIKE ‘%бург%’ – означает, что искомая строка должна содержать в себе символы ‘бург’.

Еще одной формой проверки является проверка на определитель NULL — IS NULL, т.е. если значение какоголибо поля строки таблицы не определено.

Отрицательная форма — IS NOT NULL.

Нр, выбрать все фильмы неопределенного жанра.

SELECT Title, Len FROM Movie

WHERE Kind IS NULL;

САМОСТОЯТЕЛЬНО:

Сортировка результатов

Исходные данные:

A	B
2	N
5	T
5	E
1	V
2	E
2	V

В результирующей таблице тот же порядок следования строк, что и в исходной. Изменить его можно, указав столбец, по которому необходимо упорядочить строки результата, при помощи фразы ORDER BY <имя_столбца1 [ASC | DESC] [, имя_столбца2 [ASC | DESC]...]>

ASC | DESC определяет направление упорядочивания. ASC — по возрастанию (по умолчанию), DESC — по убыванию (рис. 8.1).

SELECT *
FROM Table
ORDER BY A

A	B
1	V
2	N
2	V
2	E
5	T
5	E

SELECT *
FROM Table
ORDER BY B

A	B
5	E
2	E
2	N
5	T
1	V
2	V

SELECT *
FROM Table
ORDER BY A, B

A	B
1	V
2	E
2	N
2	V
5	E
5	T

SELECT *
FROM Table
ORDER BY 2, 1 DESC

A	B
5	E
2	E
5	N
2	T
2	V
1	V

Рис. 8.1. Примеры сортировки данных

Можно указывать не названия столбцов, а их порядковый номер в предложении SELECT.

Использование обобщающих функций SQL

Обобщающие (агрегатные, статистические) функции в качестве единственного параметра должны содержать имя столбца таблицы и возвращают единственное значение. Таких функций в языке SQL всего пять:

Count – возвращает количество непустых значений в указанном столбце. Существует вариант Count(*), возвращающий общее количество строк в результирующей таблице

Sum – возвращает сумму значений в указанном столбце

Avg – среднее значение по указанному столбцу

Min – минимальное значение в указанном столбце

Max– максмаксимальное значение в указанном столбце

Для следующей таблицы приведем примеры использования этих функций. Напомним, что результатом выполнения оператора SELECT является таблица, даже функция возвращает единственное значение, это означает, что результирующая таблица будет состоять из одной строки и одного столбца (рис. 8.2.).

A
1
5
9

SELECT Count(A) FROM Table	SELECT Count(*) FROM Table	SELECT Sum(A) FROM Table	SELECT Avg(A) FROM Table
Expr1001	Expr1001	Expr1001	Expr1001
3	5	15	5

Рис. 8.2. Агрегирующие функции

Если в предложении SELECT используются обобщающие функции, то в нем нельзя использовать имена столбцов (за исключением группировки):

SELECT Title, Sum(Len)

FROM Movie;

Для обобщающих функций, перед именем столбца можно использовать ключевое слово DISTINCT с тем, чтобы предварительно исключить из подсчетов дублирующиеся значения. Очевидно, что это действие не имеет смысла для функций MIN и MAX. Нр:

Группировка результатов (фраза GROUP BY)

Группировка предполагает объединение строк таблицы в одну по какомулибо атрибуту. При этом неопределенные значения рассматриваются как равные. Например,

```
SELECT Kind
FROM MOVIE
GROUP BY Kind;
```

В результате получим таблицу, содержащую все возможные жанры, в том числе и не указанные.

При использовании в операторе SELECT фразы GROUP BY каждый элемент списка в предложении SELECT должен иметь единственное значение для всей группы. Кроме того, предложение SELECT может включать только: имена столбцов, обобщающие функции, константы, выражения, содержащие комбинации перечисленных выше элементов.

Группировка очень часто используется для подведения промежуточных итогов при помощи обобщающих функций.

Нр: определить количество фильмов каждого жанра:

```
SELECT Kind, Count (*)
FROM MOVIE
GROUP BY Kind;
```

Этот запрос выполняется следующим образом: сначала выполняется группировка по жанрам (см. первый запрос), для каждой из полученных четырех групп подчитывается количество строк и сумма значений поля Len в пределах каждой группы.

Немного усложним запрос: требуется подсчитать количество фильмов кажого жанра и среднюю длину по каждому жанру:

```
SELECT Kind, Count (Kind), Avg (Len)
FROM MOVIE
GROUP BY Kind;
```

Еще раз обратите внимание на функцию COUNT. В первом примере использовалась COUNT(*), что позволило подсчитать количество фильмов в группе с неопределенным жанром. Во втором запросе COUNT (Kind) в группе, где жанр равен NULL, возвращает 0.

Если в запросе используется предложение WHERE, то отбор строк выполняется в первую очередь, а затем уже группировка результатов. Нр, предыдущий запрос, но только для фильмов, снятых в 70х и 80х годах:

SELECT Kind, Count (*), Avg (Len)
FROM MOVIE
WHERE Year BETWEEN 1970 AND 1989
GROUP BY Kind;

Очевидно, атрибут по которому проводится группировка должен иметь дублирующиеся значения, хотя это не является обязательным требованием. Например, в следующем запросе мы получим просто список всех фильмов:

SELECT Title
FROM Movie
GROUP BY Title

Если есть необходимость отбора строк уже после группировки, т.е. наложить условие уже на сгруппированные строки, для этого используется предложение HAVING.

Нр выбрать только те жанры, количество фильмов которых больше 3.

SELECT Kind, Count (*), Avg (Len)
FROM MOVIE
GROUP BY Kind
HAVING Count(*)>3;

Стандарт SQL требует, чтобы имена столбцов, используемые во фразе HAVING, обязательно присутствовали в списке фразы GROUP BY или применялись в обобщающих функциях. См. ниже пример 3.

Подзапросы

П/з используются во фразах WHERE и HAVING оператора SELECT и представляют из себя другой запрос (выраженный оператором SELECT). Первый (внешний) оператор SELECT использует результаты выполнения второго (внутреннего).

П/з представляет собой инструмент создания временной таблицы, содержимое которой извлекается и обрабатывается внешним оператором. Текст подзапроса должен быть заключен в скобки.

Существует три типа п/з:

Скалярный подзапрос возвращает единственное значение, но это единственное значение представляется таблицей из одной строки и одного столбца.

Нр, выбрать фильмы Э. Рязанова. В таблице MOVIE информация о режиссере содержится в виде идентификатора. Потому вначале (внутренний подзапрос) определяем ID_Pr по его фамилии из таблицы PROD.

```
SELECT title, year
FROM movie
WHERE id_pr = (SELECT id_pr
        FROM prod
        WHERE FIO = ‘Э. Рязанов’)
```

Сравнивать ID_Pr со всем подзапросом, т.е. с таблицей уместно только в случае если подзапрос скалярный. В случае, нр такого варианта написания подзапроса — SELECT id_pr, FIO или если режиссеров с фамилией 'Э. Рязанов' — более одного, компиляция такого запроса приведет к ошибке.

Операции сравнения можно использовать только со скалярными подзапросами.

Можно использовать п/з с обобщающими функциями. Нр, выбрать всех актеров, чей доход выше среднего:

```
SELECT fio, money
FROM star
WHERE money > (SELECT AVG(money)
        FROM star)
```

Строковый подзапрос возвращает значения нескольких столбцов таблицы в виде единственной строки.

Табличный подзапрос возвращает значения одного или более столбцов таблицы, размещенных более чем в одной строке. Табличный подзапрос не может быть использован в операциях сравнения. Они используются вместе с предикатом IN, который указывает на принадлежность скалярного значения некоторому множеству:

Нр, составить перечень фильмов, снятых режиссерами, имеющих доход от 10 000 до 20 000 включительно.

```
SELECT title, id_prod
FROM movie
WHERE id_prod IN (SELECT id_prod
          FROM prod
```

```
WHERE money BETWEEN 10000 AND 20000
```

К подзапросам применяются следующие правила и ограничения.

1. В подзапросах не должна использоваться фраза ORDER BY, хотя она может присутствовать во внешнем запросе.

2. Список в предложении SELECT подзапроса должен состоять из имен отдельных столбцов или составленных из них выражений — за исключением случая, когда подзапрос используется с ключевым словом EXISTS.

3. По умолчанию имена столбцов в подзапросе относятся к таблице, имя которой указано в его предложении FROM. Однако допускается ссылаться и на столбец таблицы, указанной во фразе FROM внешнего запроса, для чего используя квалифицированные имена столбцов.

4. Если подзапрос является одним из двух операндов, участвующих в операции сравнения, то запрос должен указываться в правой части этой операций

Если результат запроса зависит от строки, рассматриваемой главным запросом, то он называется коррелированным. Соответственно, если результат подзапроса никак не зависит от внешнего запроса, то — некоррелированным.

Например, выбрать фильмы в названии которых присутствует фамилия их режиссера.

```
SELECT Title
FROM Movie
WHERE ID_Pr = (SELECT ID_Pr
        FROM Prod
        WHERE Movie.Title LIKE FIO+'*');
```

Ключевые слова ANY, ALL, EXISTS

Ключевые слова ANY и ALL могут использоваться с подзапросами, возвращающими один столбец чисел. Если подзапросу будет предшествовать ключевое слово ALL, условие сравнения считается выполненным только в том случае, если оно выполняется для всех значений в результирующем столбце подзапроса. Если записи

подзапроса предшествует ключевое слово ANY, то условие сравнения будет считаться выполненным, если оно выполняется хотя бы для одного из значений в результирующем столбце подзапроса. Если в результате выполнения подзапроса будет получено пустое значение, то для ключевого слова ALL условие сравнения будет считаться выполненным, а для ключевого слова ANY — невыполненным.

Найти всех продюсеров, чей доход больше, чем у любого актера.

```
SELECT id_pr, fio, money
FROM prod
WHERE money > ALL (SELECT money FROM star);
```

Выбрать всех продюсеров, чей доход больше хотя бы одного актера.

```
SELECT id_pr, fio, money
FROM prod
WHERE money > ANY (SELECT money FROM star);
```

Ключевое слово EXISTS так же как и IN, ALL, ANY используется в подзапросах. Для EXISTS результат равен TRUE только в том случае, если в результирующей таблице есть хотя бы одна строка. Если таблица пуста, результат равен FALSE.

Нр, выбрать всех актеров, которые также являются и продюсерами.

```
SELECT Fio
FROM star
WHERE EXIST (SELECT *
        FROM prod
        WHERE star.fio=prod.fio);
```

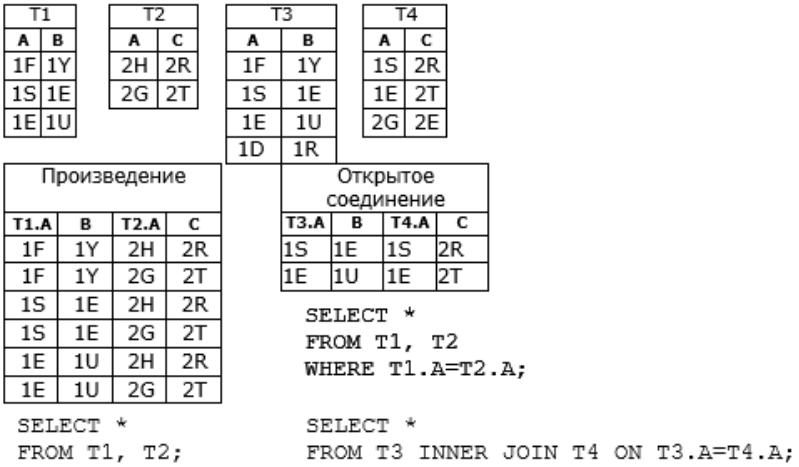
Существует отрицательный вариант — NOT EXIST, который соответственно возвращает TRUE, если таблица пуста.

Многотабличные запросы

В тех случаях, когда необходимо выбрать данные из нескольких таблиц необходимо пользоваться либо подзапросами, либо многотабличными запросами. Они используются, когда результат запроса должен содержать столбцы из нескольких таблиц. Если в предложении FROM указать имена двух и более таблиц, перечислив их

через запятую, то результатом запроса будет декартово произведение этих таблиц.

Для выполнения соединения двух таблиц необходимо указать условие соединения только тех строк, в которых совпадают одноименные атрибуты (рис. 8.3).



T1

A	B
1F	1Y
1S	1E
1E	1U

T2

A	C
2H	2R
2G	2T

T3

A	B
1F	1Y
1S	1E
1E	1U
1D	1R

T4

A	C
1S	2R
1E	2T
2G	2E

Произведение

T1.A	B	T2.A	C
1F	1Y	2H	2R
1F	1Y	2G	2T
1S	1E	2H	2R
1S	1E	2G	2T
1E	1U	2H	2R
1E	1U	2G	2T

```
SELECT *
FROM T1, T2;
```

Открытое соединение

T3.A	B	T4.A	C
1S	1E	1S	2R
1E	1U	1E	2T

```
SELECT *
FROM T1, T2
WHERE T1.A=T2.A;

SELECT *
FROM T3 INNER JOIN T4 ON T3.A=T4.A;
```

Рис. 8.3. Соединение таблиц

В последнем запросе фраза INNER JOIN связывает таблицу T3 с таблицей T4 по одноименному атрибуту A. Результат аналогичен операции отбора по условию, выполненной над декартовым произведением.

Открытые соединения бывают левое, правое и полное:

При левом открытом соединении в результирующую таблицу попадают не только строки, которые имеют полное соответствие по указанным атрибутам (T3.A=T4.A), но и те строки из первой (левой) таблицы, которые не находят соответствия во второй. При этом строки, которые не имеют соответствующих атрибутов в правой таблице дополняются пустыми значениями.

При правом соединении строки из второй (правой таблицы) не имеющие совпадения по указанным атрибутам включаются в

результирующую таблицу, дополняясь при этом слева значениями NULL.

Левое открытое соединение				Правое открытое соединение			
T3.A	B	T4.A	C	T3.A	B	T4.A	C
1F	1Y			1S	1E	1S	2R
1S	1E	1S	2R	1E	1U	1E	2T
1E	1U	1E	2T			2G	2E
1D	1R						

```
SELECT *
FROM T3 LEFT JOIN T4 ON
T3.A=T4.A;
```

```
SELECT *
FROM T3 RIGHT JOIN T4
ON T3.A=T4.A;
```

Если имена таблиц длины, а квалификационные имена используются достаточно часто, то к именам таблиц можно добавить псевдонимы (алиасы) через пробел. Псевдонимы могут использоваться вместо имени таблицы везде, где оно может употребляться.

Чаще всего многотабличные запросы применяются к таблицам, связанными как одинкомногим. Нр, найти фильмы, снятые режиссером Л. Гайдаем.

```
SELECT title, fio
FROM movie AS m, prod AS p
WHERE m.id_pr=p.id_pr AND p.fio='Л. Гайдай';
```

Здесь m и p — псевдонимы таблиц movie и prod соответственно

При помощи псевдонимов можно выполнять произведение одной таблицы на нее же.

Нр, получить пары актеров, проживающих по одному адресу:

```
SELECT s1.Fio, s2.Fio
FROM star S1, star S2
WHERE s1.adres=s2.adres;
```

При таком запросе в результат попадают также сцепки записей таблицы Star с записями этой же таблицы, совпадающие по полю адрес, а этому условию удовлетворяет, в том числе и сцепка каждого актера с

ним же. Чтобы исключить их нужно добавить условие исключения таких записей из результата.

```
SELECT s1.Fio, s2.Fio
FROM star S1, star S2
WHERE s1.adres=s2.adres AND s1.id_st<>s2.id_st;
```

Но и это решение не дает нужного результата. т.к. для записи S1 с ID_St=4 (Л. Голубкина) мы получим ее сцепление с записью S2.ID_St=6 (А. Миронов). А для записи S1.ID_St=6 парой будет S2.ID_St=4. Т.е. получаем дублирование пар. Поэтому правильным вариантом будет:

```
SELECT s1.Fio, s2.Fio
FROM star S1, star S2
WHERE s1.adres=s2.adres AND s1.id_st>s2.id_st;
```

Это условие исключает один (неважно какой) вариант сцепления записей и оставляет другой.

```
SELECT s.fio, p.fio
FROM star AS s, prod AS p
WHERE s.adres=p.adres AND s.fio<>p.fio;
```

Объединение, пересечение, разность

Эти теоретикомножественные операции РА также имеют место в языке SQL и обозначаются соответственно словами UNION, INTERSET и EXCEPT.

Пример получить список всех персон киноиндустрии:

```
(SELECT fio, 'star' AS WHO
 FROM star)
UNION
(SELECT fio, 'prod' AS WHO
 FROM prod)
```

Таблицы, участвующие в объединении должны быть совместимы по объединению, т.е. они должны иметь один и тот же набор атрибутов, кроме того, значения соответствующих столбцов должны быть определены на одном и том же домене.

Пример пересечения: получить список продюсеров, являющихся актерами:

(SELECT fio FROM prod)
INTERSECT
(SELECT fio FROM star)

Пример разности: выбрать актеров, не являющихся продюсерами.

(SELECT fio FROM star)
EXCEPT
(SELECT fio FROM prod)

При этих операциях происходит исключение дубликатов по умолчанию. В тех случаях, когда требуется присутствие всех значений добавляется ключевое слово ALL.

Лекция 9. ВВЕДЕНИЕ В ТЕХНОЛОГИЮ КЛИЕНТ-СЕРВЕР

В локальных БД, как это следует из названия, базы данных располагаются на машине клиента, т.е. на той же машине, что и использующее их приложения. При желании других пользователей манипулировать теми же данными, нужно предоставить дисковый ресурс с БД для общего использования. В этом случае машина с БД будет называться файлсервером.

В файлсерверных БД данные располагаются на сетевом файлсервере, который может быть доступен одновременно нескольким пользователям, поэтому к таким БД возможен многопользовательский режим доступа. Данные в БД хранятся в единственном экземпляре, а каждый клиент в каждый момент времени работает с некоторой локальной копией этих данных, причем управление данными целиком возлагается на клиентские программы. Именно они должны заботиться о синхронизации локальных копий данных на каждом клиентском месте с содержимым основной (и единственной) базы данных.

В обоих случаях BDE располагается на машине клиента и вместе с программой образует локальную СУБД, количество копий которой равно количеству пользователей. В таких системах, в принципе, возможен удаленный доступ, если используемая в них сеть позволяет пользователю находиться на удалении от офиса — в другом здании или в другом городе (вариант разветвленных локальных сетей, региональные служебные сети, наконец, сети с выходом в Internet).

Архитектура клиентсервер. Архитектура файлсервер неэффективна по крайней мере в двух отношениях.

1. При выполнении запроса к БД, расположенной на файловом сервере, в действительности происходит запрос к локальной копии данных на компьютере пользователя. Поэтому перед выполнением запроса данные в локальной копии в полном объеме обновляются из реальной БД. Так, если таблица БД состоит из 10 000 записей, а для выполнения запроса нужно только 10 записей, все равно клиенту передаются все 10 000 записей. Таким образом, не нужно иметь слишком много пользователей и запросов от них, чтобы серьезно

загрузить сеть, что, конечно же, не может не сказаться на ее быстродействии.

2. Целостность БД обеспечивается клиентскими программами. Это потенциальный источник ошибок, нарушающих физическую и логическую целостность данных, поскольку различные клиенты могут производить контроль целостности поразному или не проводить такого контроля вовсе. Намного эффективнее управлять БД из единого места и по единым законам. Поэтому безопасность при работе в архитектуре файлсервер невысока и всегда присутствует элемент неопределенности. В такой архитектуре трудно обеспечить секретность данных и их защиту: таблицы хранятся на сервере в виде обычных файлов, поэтому любой, кто имеет доступ к каталогу (каталогам) сетевого сервера, где хранится БД, может изменять таблицы, копировать их, заменять и т. д. В архитектуре клиентсервер между BDE и базой данных появляется важное промежуточное звено — сервер БД — специальная программа, управляющая базой данных.

Клиент формирует запрос к серверу на языке запросов SQL (Structured Query Language — структурированный язык запросов), являющемся промышленным стандартом для реляционных БД. SQLсервер обеспечивает интерпретацию запроса, его выполнение, формирование результата и выдачу этого результата клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса. Клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и предоставляет пользователю.

Так как клиентскому приложению посылается результат выполнения запроса, по сети передаются только те данные, которые в действительности нужны клиенту. В итоге снижается нагрузка на сеть. Кроме того, SQL сервер, если это возможно, оптимизирует полученный запрос таким образом, чтобы он был выполнен за минимально возможное время. Все это повышает быстродействие системы и снижает время ожидания результата запроса.

При выполнении запросов сервером существенно повышается степень безопасности данных, поскольку правила целостности данных определяются на сервере и являются едиными для всех приложений,

использующих эту БД. В результате исключается возможность определения противоречивых правил поддержания целостности. Мощный аппарат транзакций, поддерживаемый SQLсерверами, блокирует одновременное изменение одних и тех же данных различными пользователями и предоставляет возможность откатов к первоначальным значениям при внесении в БД изменений, закончившихся аварийно.

Данные в серверной БД обычно физически хранятся на диске в виде одного большого файла, что в сочетании с назначаемыми каждому пользователю паролями и привилегиями существенно повышает защиту данных от намеренной порчи и хищений.

В архитектуре клиентсервер используются промышленные серверы данных типа Oracle, Gupta, Informix, Sybase, MS SQL Server, DB2, InterBase и ряд других.

Архитектура с сервером приложений. Развитие идей архитектуры клиентсервер привело к появлению трехзвенной архитектуры доступа к базам данных (в литературе ее также называют многозвенной архитектурой, Ntier или multitier архитектурой).

Архитектура клиентсервер — двухзвенная: первым звеном в ней является программа клиента, а вторым — сервер БД и сама БД.

В трехзвенной архитектуре создается вспомогательная программа, в которую включаются все компонентынаборы данных, бывшие ранее (в двухзвенной архитектуре) собственностью клиентских приложений, а также вспомогательные компоненты TDatabase и TSession. Затем эта программа регистрируется в качестве COM или CORBAсервера, после чего она становится сервером приложений. Теперь клиентские машины могут не иметь BDE, а клиентские программы уже не включают в себя громоздкие коды компонентовнаборов и многих других вспомогательных компонентов. Для получения доступа к серверным данным они обращаются к удаленному (т.е. находящемуся на другой машине) серверу приложений, который и реализует необходимый обмен данными. Заметим, что слева на рисунке показан вариант размещения сервера приложений на машине сервера БД. Такой вариант наиболее популярен, т. к. снижает загрузку сети и гарантирует одновременную работу обоих серверов. Вообще говоря, это не

обязательно: сервер приложений может располагаться на любой сетевой машине, оснащенной BDE. Разумеется, в этом случае каталог его размещения должен быть доступным другим сетевым машинам, а сама машина сервера приложений должна быть включена в период работы с сервером данных. Более того, в этой архитектуре можно использовать файлсерверный вариант размещения данных (на рис. — справа).

С помощью словарей БД можно перенести в компонентыисточники и связанные с ними поля часть бизнесправил, касающуюся различных ограничений на значения вводимых данных. В этом случае неправильные данные будут отвергаться сервером приложений и не будут передаваться в сервер БД.

В клиентской программе, которая в этой архитектуре называется облегченным или тонким клиентом (thin client), размещается клиентский набор данных, представляющий собой копию части данных из БД. Все изменения, которые пользователь вносит в данные, изменяют эту локальную копию и могут до нужной поры не передаваться в БД (режим отложенной обработки данных). Кроме того, при работе с громоздкими таблицами можно потребовать от сервера приложений передавать в локальный набор записи таблицы порциями, достаточными для одновременного отображения на экране клиента. Все эти меры существенно снижают загрузку сети и, следовательно, уменьшают время ожидания результата запроса.

Лекция 10. ОПРЕДЕЛЕНИЕ ДАННЫХ В ЯЗЫКЕ SQL

Целые

INTEGER — целое число 4 байта (–2 147 483 648… +2 147 483 647),

SMALLINT — короткое целое 2 байта (–32 788 … +32 767)

Числа с фиксированной точкой

NUMERIC (w, d), DECIMALS (w, d)

w — разрядность (1... 18) — общее количество цифр (в целой и дробной частях без учета десятичной точки);

d — количество десятичных разрядов (0.. w).

Фактически, числа с фиксированной точкой относятся к порядковым типам, т.е. все возможные значения можно пересчитать по порядку. Для них выделяется определенное количество байт в зависимости от разрядности, в которых хранится двоичное представление целого числа, а для этого целого числа просто определяется положение десятичной точки.

Например, если мы определяем столбец как NUMERIC (3,2), это не означает, что максимальное значение, которое можно в нем хранить — 9,99. Для этого столбца выделяется 2 байта и т.о. макисмальное значение — 327,67, а минимальное — –327,68.

При w = 1—4 для NUMERIC выделяется 2 байта, для DECIMALS — 4 и это вся разница между этими типами. Например,

CREATE TABLE T1 (A NUMERIC (3, 1), B DECIMALS (3,1));

Если после создания такой таблицы посмотреть на ее структуру, то INTERBASE покажет: A NUMERIC (3, 1), B NUMERIC (3,1), тем не менее максимально допустимые значения будут: для A — 3276,7, а для B — 2 147 483 64,7.

При w = 5—9 и для NUMERIC и для DECIMALS выделяется по 4 байта.

При w = 10—18 — по 6 байт.

Вещественные

DOUBLE PRECISION — тип чисел с плавающей точкой для бухгалтерских и научных расчетов. Существует еще тип FLOAT, который не рекомендуется использовать по причине недостаточной точности, а также в нем быстро накапливаются ошибки округления.

Строковые

CHAR (Character) и VARCHAR (Character Varying).

Например,

CREATE TABLE T1 (A CHAR (200), B CHAR);

В результате поле A будет иметь длину 200 символов, а B — 1 символ.

Максимальная длина — 32768 символов.

Отличие между двумя строковыми типами в том, что фактическая длина значений типа CHAR всегда совпадает с максимальной. Т.е. даже если в поле A таблицы T1 занести значение 'ABC', то при выборке мы получим зачение 'ABC <197 пробелов> '.

Для значений типа VARCHAR понятия максимально возможная длина и фактическая различаются.

На практике использование типа CHAR довольно оганичено.

Для строкового типа важнейшей характеристикой является CHARACTER SET — набор символов. Набор символов определяется для всей БД и по умолчанию используется для всех символьных полей. Однако его можно явно переопределить при создании поля. Например:

```
CREATE TABLE T1 (
  A VARCHAR (20),
  B VARCHAR (20) CHARACTER SET WIN1251);
```

Для символьных полей возможно указавать порядок сортировки COLLATION ORDER. Для кирилического набора символов WIN1251 возможны два варианта:

```
CREATE TABLE T1 (
  A VARCHAR (20) COLLATION ORDER WIN1251,
  B VARCHAR (20) COLLATION ORDER PXW_CYRL);
```

A	B
Ф	Ф
б	б
Ц	Ц
ф	ф
Б	Б
А	А
ц	ц

SELECT *
FROM T1
ORDER BY A;

A	B
А	А
Б	Б
Ф	Ф
Ц	Ц
б	б
ф	ф
ц	ц

SELECT *
FROM T1
ORDER BY B;

A	B
А	А
б	б
Б	Б
ф	ф
Ф	Ф
ц	ц
Ц	Ц

Дата-время

DATE — дата в диапазоне 1 января 100 г до 29 февраля 32768 года;

TIME — время с точностью до 1/10 000 доли секунды, 00:00.. 23:59:59.9999

TIMESTAMP — дата и время.

С полями этих трех типов возможны операции + и –. С типом DATE операции выполняются в днях, с TIME — в секундах, с TIMESTAMP — в сутках, с учетом доли суток.

Например, 31.10.04 – 29.10.04 = 2; 30.12.04 + 3 = 2.01.05

07:00:00 – 08:00:01 = –3601; 23:00:00 + 7200 = 01:00:00

27.10.2004 7:00:00 – 25.10.2004 1:00:00 = 2,25

Типы данных BLOB

Binary Large OBject — динамически расширяемый тип данных. В полях этого типа могут хранится данные неограниченного размера. Как правило это примечания, графическая, аудио, видео информация. Достигается это за счет физического размещения BLOBполей отдельно от обычных.

Несмотря на сложность реализации, объявить BLOBстолбец очень просто:

CREATE TABLE T1 (A BLOB);

Массивы

Полямассивы являются расширением традиционной реляционной модели, по которой каждый атрибут (поле) должен быть атомарным, т.е. неделимым.

```
CREATE TABLE T1 (
  A INTEGER [1:12],
  B CHAR [5:5, 10],
  C VARCHAR (20) [6, 6, 6]);
```

В таблице T1 объявлено 3 массива, каждый из которых содержит данные различных типов: A — одномерный (можно писать A INTEGER [12]), B — двумерный 11□10, C — трехмерный 6□6□6.

Реализованы массивы на базе полей типа BLOB.

На данный момент массивы являются нововведением в базах данных, и еще не разработано компонентов DELPHI/ C++Builder поддерживающих работу с ними, поэтому их использование ограничено. Хотя существуют библиотеки сторонних разработчиков (FIBPlus), которые могут корректно работать с массивами.

Определение таблиц

выполняется оператором CREATE TABLE. В нем определяются столбцы будущей таблицы и ограничения. Его синтаксис:

```
CREATE TABLE <table_name> [EXTERNAL [FILE] "<file_spec>"]
(<col_def1> [<coldef2>, ... ] [<constraint_def>, ...]);
```

где <table_name> — имя таблицы, если указано [EXTERNAL [FILE] "<file_spec>"], то будет создана т.н. внешняя таблицы, которая хранится в отдельном файле <file_spec>. Работа с внешними таблицами ограничена операциями вставки и выборки.

<col_def> — определение столбца (поля) таблицы. В свою очередь имеет синтаксис:

```
<col_def> = col_name { datatype | domain | COMPUTED [BY] (<expr>)
}
[DEFAULT {<default_value> | NULL | USER}]
[NOT NULL]
[<col_constraint>]
[COLLATE collation]
```

Здесь, <col_name> — имя столбца, для которого должен быть указан либо тип данных (datetype), либо имя домена (domain), предварительно определенного, либо значение в стобце будет вычислется (COMPUTED BY), но только при выборке по этому стобцу, т.е. в этом поле ничего не хранится.

Значение столбца может быть быть определено по умолчанию (DEFAULT). Т.е. если при вставке записи в таблицу пользователь оставляет не заполненным это поле, то ему автоматически при¬сваивается либо значение <default_value>, либо значение NULL, либо имя пользователя, который вставляет запись (USER).

На столбец может быть наложено требование NOT NULL, что не позволит ни при каких операциях (вставка, изменение) оставить это поле без значения. Часто опцию NOT NULL сочетают с DEFAULT.

<CHECK> определяет дополнительные ограничения на возможные значения столбца для реализации бизнесправил.

<COLLATE> определяет порядок сортировки для символьных столбцов в случае различного регистра.

В качестве примера создадим таблицу Prod.

```
CREATE TABLE prod (
   id_pr INTEGER NOT NULL,
   fio VARCHAR(30) NOT NULL,
   bd DATE,
   adres VARCHAR(50),
   money INTEGER DEFAULT 0);
```

При определении таблиц создаются особые объекты БД — ограничения — CONSTRAINT. Существуют следующие виды ограничений:

- первичный ключ — PRYMARY KEY;
- уникальный ключ — UNIQUE KEY;
- внешний ключ — FOREIGN KEY;
- проверки — CHECK.

При определении столбца синтаксис ограничений следующий:

```
<col_constraint> =
[CONSTRAINT constraint_name]
{ UNIQUE  |  PRYMARY KEY  |  CHECK <condition>  |
```

```
    REFERENCES table_name [(column_list)]
    [ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET
NULL}]
    [ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET
NULL}]
  }
  CREATE TABLE prod (
    id_pr INTEGER NOT NULL CONSTRAINT C1 PRIMARY KEY,
    fio VARCHAR(30) NOT NULL,
    bd DATE,
    adres VARCHAR(50),
    money INTEGER DEFAULT 0 CONSTRAINT C2 CHECK (money
>= 0)   );
```

На таблицу PROD наложены два ограничения C1 (id_pr — PRYMARY KEY) и C2 (money должен быть положительным) Объявление столбца id_pr в качестве первичного ключа, автоматически накладывает на него требование уникальности, но требование непустого значения первичного ключа NOT NULL нужно явно указывать. Если требуется уникальность значений столбца не являющегося первичным ключом, то используется ключевое слово UNIQUE.

Как видно из синтаксиса требование явного именования ограничений не является обязательным. Можно было бы написать:

```
  CREATE TABLE prod (
    id_pr INTEGER NOT NULL PRIMARY KEY,
    fio VARCHAR(30) NOT NULL,
    bd DATE,
    adres VARCHAR(50),
    money INTEGER DEFAULT 0 CHECK (money >= 0)    );
```

В этом случае ограничения получат системные имена, которые хранятся в системной таблице RDB$RELATION_CONSTRAINTS.

Ограничения могут быть созданы не только при определении столбца, но и наряду с определениями столбцов в таблице. В этом случае в них могут быть задействованы несколько полей (например, составной ключ) и синтаксис поэтому немного иной:

<constraint_def> = [CONSTRAINT constraint_name]
{ PRIMARY KEY | UNIQUIE (column_list) } |
FOREIGN KEY (column_list) REFERENCES <table_name> [(column_list)]
[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]
[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

Например, создание таблицы PROD может выглядеть:

```
CREATE TABLE prod (
        id_pr INTEGER NOT NULL,
        CONSTRAINT C1 PRIMARY KEY (id_pr),
        fio VARCHAR(30) NOT NULL,
        bd DATE,
        adres VARCHAR(50),
        money INTEGER DEFAULT 0,
        CONSTRAINT C2 CHECK (money >=0)      );
```

Общий синтаксис условия в предложении CHECK:

<search_condition> =
{<val> <operator> { <val> | (<select_one>) }
| <val> [NOT] BETWEEN <val> AND <val>
| <val> [NOT] LIKE <val> [ESCAPE <val>]
| <val> [NOT] IN (<val> [, <val> ...] | <select_list>)
| <val> IS [NOT] NULL
| <val> [NOT] {= | < | >} | >= | <=} {ALL | SOME | ANY } (<select_list>)
| EXISTS (<select_expr>)
| SINGULAR (<select_expr>)
| <val> [NOT] CONTAINING <val>
| <val> [NOT] STARTING [WITH] <val>
| (<search_condition>)
| NOT <search_condition>
| <search_condition> OR <search_condition>
| <search_condition> AND <search_condition>}

Проверка CHECK относится только к текущей записи. Не следует строить выражения с использованием других записей этой же таблицы.

Поле может иметь только одно CHECK ограничение.

Если при определении столбца использовался домен со своим ограничением, то его нельзя переопределить на уровне конкретного поля.

Определение домена

Домен представляет собой некоторый базовый тип данных с наложенным на него фильтром (условием). Домен является объектом всей БД, а не отдельной таблицы и может использовтся при определении столбцов различных таблиц наряду со стандартными типами.

В нашей базе данных у таблиц PROD и STAR имеет смысл определить домен для поля MONEY.

```
CREATE DOMAIN D_Money INTEGER DEFAULT 0 CHECK (VALUE >= 0)
```

Определение домена практически полностью повторяет определение столбца за исключением того, что при наложении ограничения CHECK вместо имени столбца используется ключевое слово VALUE — просто значение.

```
CREATE TABLE prod (
        id_pr INTEGER NOT NULL,
        CONSTRAINT INT1 PRIMARY KEY (id_pr),
        fio VARCHAR(30) NOT NULL,
        bd DATE,
        adres VARCHAR(50),
        money D_Money)          );
```

Разумеется нет смысла создавать домен для его одноразового использования. Чем в больше количество таблиц, использующих один домен, тем яснее преимущество. Помимо этого очевидного плюса, домены сокращают и количество ограничений в БД, т.к. ограничение будет одно для каждого домена.

В нашем примере для таблицы PROD будет создано только одно ограничение (на id_pr — PK).

Связи и ссылочная целостность

Как известно, для создания связи между двумя таблицами необходимы два ключевых поля — PRIMARY KEY и FOREIGN KEY. Известно также, что FK может принимать значения только из множества значений PK или быть пустым. Это есть не что иное как ограничение. Поэтому в синтаксисе ограничений есть фраза и для объявления FK (в синтаксисе на основе определения столбца присутствует неявно).

Очевидно, что зависимая (дочерняя) таблица должна быть создана только после создания главной.

В рассматриваемом примере таблица MOVIE является зависимой от таблицы PROD по полю id_pr:

```
CREATE TABLE movie (
    id_m INTEGER NOT NULL PRIMARY KEY,
    title VARCHAR(40) NOT NULL UNIQUE,
    year INTEGER NOT NULL CHECK (year>1910),
    len INTEGER NOT NULL CHECK (len>20),
    kind CHAR(10) CHECK (kind IN ('Комедия','Боевик','Мелодрама')),
    id_pr INTEGER,
    FOREIGN KEY (id_pr) REFERENCES PROD );
```

Предложение FOREIN KEY определяет вторичный ключ id_pr для связи дочерней таблицы MOVIE с родительской таблицей PROD.

Рассмотрим подробнее формат определения:

FOREIN KEY (<список столбцов внешнего ключа>)

REFERENCES <имя родительской таблицы>

[<список столбцов родительской таблицы>]

[ON DELETE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

[ON UPDATE {NO ACTION | CASCADE | SET DEFAULT | SET NULL}]

Список столбцов внешнего ключа определяет столбцы дочерней таблицы, по которым строится внешний ключ.

Имя родительской таблицы определяет таблицу, в которой описан первичный ключ (или столбец с атрибутом UNIQUE). На этот ключ

(столбец) должен ссылаться внешний ключ дочерней таблицы для обеспечения ссылочной целостности.

Параметры ON DELETE, ON UPDATE определяют способы изменения подчиненных записей дочерней таблицы при удалении или изменении поля связи в записи родительской таблицы. Перечислим эти способы:

- NO ACTION — запрет удаления/изменения родительской записи при наличии подчиненных записей в дочерней таблице;
- CASCADE — для оператора ON DELETE: при удалении записи родительской таб¬лицы происходит удаление подчиненных записей в дочерней таблице; для ON UPDATE: при изменении поля связи в записи родительской таблицы происходит изменение на то же значение поля внешнего ключа у всех подчиненных записей в дочерней таблице;
- SET DEFAULT — в поле внешнего ключа у записей дочерней таблицы заносится значение этого поля по умолчанию, указанное при определении поля (параметр DEFAULT); если это значение отсутствует в первичном ключе, возбуждается исключение; причем используется значение по умолчанию, имевшее место на момент определения ссылочной целостности; если впоследствии это значение будет изменено, ссылочная целостность при SET DEFAULT все равно будет использовать прежнее значение;
- SET NULL — в поле внешнего ключа у записей дочерней таблицы заносится значение NULL.

Удаление таблицы

Осуществляется оператором DROP TABLE <имя_таблицы>

Удаление невозможно для родительских таблиц, если в дочерних таблицах имеются ссылки по внешнему ключу этих таблиц и удаление главной таблицы приведет к нарушению целостности БД. Поэтому нужно сначала либо удалить наложенные ограничения ссылочной целостности, либо удалить сначала дочерние таблицы, а только затем главные.

Изменение таблицы

Оператор ALTER TABLE позволяет добавить|удалить (ADD | DROP) столбец, добавить|удалить (ADD | DROP CONSTRAINT) ограничения.

Изменить атрибуты столбца непосредственно нельзя. Для этого нужно проделать определенную процедуру.

Перед изменением какихлибо атрибутов столбца данные, которые хранятся в нем, нужно сохранить. Для этого в таблице определяют временный столбец, в точности повторяющий все характеристики того столбца, который планируется изменить. Затем данные из изменяемого столбца копируют во временный столбец (используя, например, оператор UPDATE). После этого столбец, подлежащий изменению, попросту удаляют из таблицы, а на его месте создают новый, одноименный столбец с желаемыми атрибутами. В заключение в него копируют данные из временного столбца, а временный столбец уничтожают.

Пусть, например, необходимо изменить характеристики столбца FIO, изменив тип столбца с VARCHAR(4O) на CHAR(50).

1. Добавляем в таблицу новый временный столбец FIO_TMP, полностью повторяющий характеристики изменяемого столбца FIO:

```
ALTER TABLE PROD
ADD FIO_TMP VARCHAR(40);
```

2. Копируем данные из FIO в FIO_TMP:

```
UPDATE PROD SET FIO_TMP = FIO;
```

3. Удаляем столбец FIO:

```
ALTER TABLE PROD DROP FIO;
```

4. Создаем новый столбец FIO с необходимыми характеристиками:

```
ALTER TABLE PROD ADD FIO CHAR(50);
```

5. Переписываем данные из столбца FIO_TMP в новый столбец FIO:

```
UPDATE PROD SET FIO = FIO_TMP;
```

6. Удаляем временный столбец FIO_TMP:

```
ALTER TABLE PROD DROP FIO_TMP;
```

Замечание. Следует помнить, что изменение характеристик столбца, а также удаление столбца невозможно, если:

– столбец приобретает атрибуты PRIMARY KEY или UNIQUE, но старые значение в столбце нарушают требования уникальности данных;
– удаляемый столбец входил как часть в первичный или внешний ключ, что привело к нарушению ссылочной целостности между таблицами;
– столбцу были приписаны ограничения целостности CHECK на уровне таблицы;
– столбец использовался в иных компонентах БД — в просмотрах, триггерах, в выражениях для вычисляемых столбцов.

Все вышесказанное свидетельствует о том, что в случае необходимости изменения атрибутов столбца или в случае удаления столбца сначала необходимо тщательно про¬анализировать, какие последствия для таблицы и базы данных в целом может повлечь такое изменение или удаление.

Оператор INSERT

Оператор INSERT применяется для добавления записей в объект. В качестве объекта может выступать ТБД или просмотр (VIEW), созданный оператором CREATE VIEW. В последнем случае записи могут добавляться сразу в несколько таблиц.

Формат оператора INSERT:

INSERT INTO <объект> [(столбец1 [, столбец2 ...])]

{VALUES (<значение1> [, <значение2> ...]) | <оператор SELECT>}

Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список столбцов может быть опущен. В этом случае подразумеваются все столбцы объекта, причем в том порядке, в котором они определены в данном объекте.

Поставить в соответствие столбцам списки значений можно двумя способами. Первый состоит в явном указании значений после слова VALUES, второй — в формировании значений при помощи оператора SELECT.

Например, добавить в таблицу

```
CREATE TABLE movie (
    id_m INTEGER NOT NULL PRIMARY KEY,
    title VARCHAR(40) NOT NULL UNIQUE,
```

year INTEGER NOT NULL CHECK (year>1910),
len INTEGER NOT NULL CHECK (len>20),
kind CHAR(10) CHECK (kind IN ('Комедия','Боевик','Мелодрама')),
id_pr INTEGER,
FOREIGN KEY (id_pr) REFERENCES PROD);

еще один фильм:

INSERT INTO MOVIE (id_m, title, year, length, kind, id_pr)
VALUES (19, ‘Двенадцать стульев’, 1986, 455, ‘комедия’, 6);

Если добавляемые значения указываются для всех столбцов, нет необходимости перечислять столбцы таблицы:

INSERT INTO MOVIE
VALUES (19, ‘Двенадцать стульев’, 1986, 455, ‘комедия’, 6);

Значения присваиваются столбцам по порядку следования тех и других в операторе: первому по порядку столбцу присваивается первое значение, второму столбцу — второе значение и т. д.

Существует другой вариант добавления записей — при помощи оператора SELECT. Нр, отразить в БД, что режиссер Э. Рязанов стал актером.

INSERT INTO star (fio, bd, adres, money)
SELECT fio, bd, adres, money FROM prod
WHERE fio=’Э. Рязанов’;

Т.к. оператор SELECT способен вернуть множество записей, то можно добавить несколько записей сразу. Нр, предположим, что продюсерами стали сразу все актеры с доходом более 100000.

INSERT INTO prod (fio, bd, adres, money)
SELECT fio, bd, adres, money FROM star
WHERE money>=100000;

Следует обратить внимание, что приведенные примеры носят показательный характер, т.к. практически в таблицу добавляются записи для которых отсутствует идентификатор id_pr и id_st, которые являются в своих таблицах первичными ключами и не могут быть пустыми.

Для этих целей в составе БД используют генераторы — механизм сервера БД, возвращающий уникальные значения, никогда не

совпадающие со значениями, выданными этим же генератором в прошлом.

Оператор UPDATE

Оператор UPDATE используется для изменения значения в группе записей или (в частном случае) в одной записи объекта. В качестве объекта могут выступать ТБД или просмотр, созданный оператором CREATE VIEW. В последнем случае могут изме¬няться значения записей из нескольких таблиц.

Формат оператора UPDATE:

UPDATE <объект>

SET столбец1 = <значение1> [,столбец2 = <значение2>...]

[WHERE <условие поиска >]

При корректировке каждому из перечисленных столбцов присваивается соответст¬вующее значение. Корректировка выполняется для всех записей, удовлетворяющих условию поиска, которое задается так же, как в операторе SELECT.

Обратите внимание: если опустить WHERE <условие поиска>, в объекте будут изменены все записи!

Нр, увеличить доход всех актеров, занятых в фильме «Операция Ы» на 20%.

```
UPDATE star
SET star.money = star.money*1.2
WHERE id_st IN (SELECT id_st
        FROM starin si INNER JOIN movie m ON (si.id_m=m.id_m)
        WHERE title='Операция "Ы"');
```

Оператор DELETE

Предназначен для удаления группы записей из объекта. В качестве объекта могут выступать ТБД или просмотр VIEW. Формат оператора:

DELETE FROM <объект>

WHERE <условие поиска>;

Из объекта будут удалены все записи, удовлетворяющие условию поиска. Как и в случае оператора UPDATE, если условие WHERE не указано, будут удалены все записи.

Нр, удалить фильмы режиссера Э. Рязанова.

```
DELETE FROM movie
WHERE id_pr IN (SELECT id_pr FROM prod
                WHERE fio = 'Э. Рязанов')
```

Индексы

Индексы предназначены для быстрого поиска записи. Как компьютеру, так и человеку поиск в какомлибо множестве удобнее всего осуществлять по упорядоченным элементам этого множества. Поэтому, если требуется найти запись в таблице, то нужно все записи упорядочить. По какому принципу? Очевидно — по тому полю, по значению которого мы ищем запись. Например, при поиске актера по его фамилии, если расположить записи в алфавитном порядке следования фамилий, то поиск будет присходить значительно быстрее. Но, в то же время, требуется поиск и по идентификатору актера. Понятно, что алфавитный порядок следования фамилий не соппадает с порядком следования идентификаторов. Сортировать таблицу каждый раз для каждого поиска по затратам выйдет даже дороже, чем поиск по неотсортированной таблице. Поэтому, поступают так: основную таблицу оставляют без изменений, записи в ней располагаются в порядоке их естественного добавления, отдельно составляют упорядоченный список значений поля и каждому значению ставят в соответствие указатель на местоположение соответствующей записи. Такой список, сохраненный в БД вместе с таблицей и есть индекс (рис. 10.1).

Рис. 10.1. Индексы

Понятно, что индекс занимает объем равный объему данных в индексируемом поле плюс объем указателей, т.е. суммарный объем индексов может превышать размер таблицы.

Индексы требуют постоянной поддержки, т.е. при модификации таблицы (удалении, добавлении записей или изменении индексного поля), разумеется индекс также должен быть преобразован. Однако, затраты на поддержание индексов с лихвой окупаются скоростью поиска. Хотя на каждую таблицу можно создать до 64 индексов, вполне очевидно, что индексы нужно создавать только по тем полям, по которым действительно требуются поиск и сортировка. В случае необоснованно большого количества индексов, поддержка индексов может сильно сказаться на производительности всей БД.

Кроме быстрого поиска, индексы позволяют пользователю видеть записи в том или ином порядке, отличном от ее физического размещения, т.е. индекс позволяет отделить хранение данных от их представления.

Т.о. индекс — это отдельный объект БД, содержащий упорядоченные значения одного или нескольких полей таблицы и связанный с ними указатель на страницу, где расположена запись, содержащая поле с этим значением. Индескы предназначены для быстрого поиска и сортировки.

Общий формат оператора CREATE INDEX:

CREATE [UNIQUE] [ASC[ENDING] | DESC[ENDING]]

INDEX <index_name> ON <table_name> (col1 [,col2 ...]);

- UNIQUE — требует создания уникального индекса, не допускающего одинако¬вых значений индексных полей для разных записей таблицы;
- ASC[ENDING] — указывает на необходимость сортировки значений индексных полей по возрастанию (режим принят по умолчанию);
- DESC[ENDING] — указывает на необходимость сортировки значений индекс¬ных полей по убыванию;
- index_name — имя создаваемого индекса;
- table_name — имя таблицы, для которой создается индекс;
- colN — имена столбцов, по которым создается индекс.

Индексы по полям, на которые наложены ограничения: PRIMARY KEY, FOREIGN KEY, UNIQUE создаются автоматически. Для PRIMARY KEY и UNIQUE они необходимы, т.к. при добавлении или изменении записи нужно проверять наличие такого же значения поля для обеспечения его уникальности. Индекс по FOREIGN KEY при изменении/удалении записи родительской таблицы дает возможность быстрого поиска связанных с ней записей дочерней таблицы для выполнения действий по поддержанию ссылочной целостности.

Индексы необходимо создавать в случае, когда по столбцу или группе столбцов:

- часто производится поиск в БД (столбец или группа столбцов часто перечисля¬ются в предложении WHERE оператора SELECT);
- часто строятся соединения таблиц (JOIN);
- часто производится сортировка (то есть столбец или столбцы часто использу¬ются в предложении ORDER BY оператора SELECT).

Не рекомендуется строить индексы по полю или полям, которые:

- редко используются для поиска, объединения и сортировки результатов запросов;
- часто меняют значение, что приводит к необходимости часто обновлять индекс и способно существенно замедлить скорость работы с БД;
- содержат небольшое число вариантов значения (например, пол).

Помимо двух вышеуказанных причин — занимаемый индексами объем и постоянная поддержка индексов, по которым не следует строить черезчур много индексов, существует еще ряд замечаний.

Как известно, язык SQL является непроцедурным, т.е. позволяет сформулировать ЧТО нужно получить в результате, а КАК добиться нужно результата, т.е. алгоритм или план выполнения запроса, строит т.н. оптимизатор запросов. К сожалению, оптимизатор далеко не идеален и при сложных запросах и при большом количестве индексов способен построить далеко не оптимальный план, задействовав не самые эффективные индексы.

Если запрос возвращает более 20% записей, то использование индексов может существенно замедлить выполнение запроса. Конечно

цифра 20% весьма приблизительная, но она показывает тот порог, когда эффективность индекса можно поставить под сомнение.

Улучшение производительности индекса

После многократного внесения изменений в таблицу БД индексы этой таблицы мо¬гут быть разбалансированы. Разбалансировка приводит к тому, что «глубина» индекса (depth) возрастает сверх критического значения. «Глубина» индекса — параметр, пока¬зывающий максимальное количество операций, необходимых для нахождения иско¬мого значения в таблице БД с использованием данного индекса. В случае разбалансировки индекса его ценность при выполнении запросов снижается изза увеличения времени выполнения запроса.

Поэтому время от времени необходимо выполнять либо перестройку индекса, либо удалить и заново создать индекс:

Перестройка индекса заключается в пересоздании и балансировке индекса. Эти операции начинаются после деактивизации индекса и последующей его активизации

ALTER INDEX <имя индекса> INACTIVE;

ALTER INDEX <имя индекса> ACTIVE;

Деактивизация индекса полезна также в том случае, когда в таблицу БД вставляет¬ся большое число записей одновременно: при активном индексе изменения в него вносятся при добавлении каждой записи, что замедляет доступ к данным.

Удаление индекса: DROP INDEX <имя индекса>;

Нельзя удалить или перестроить индекс:

- созданный сервером по столбцам с ограничениями PRIMARY KEY, FOREIGN KEY, UNIQUE);
- используемый в данный момент в других запросах;
- нет соответствующих соответствующие привилегии досту¬па к БД.

Кроме того, при восстановлении БД из архивной копии, в которой хранится лишь определение индекса, а не его данные, индекс будет построен заново.

Четвертый способ улучшить производительность индекса — собрать статистику по индексу при помощи команды

SET STATISTICS INDEX <index_name>

Статистика таблицы — это величина в пределах от 0 до 1, значение которой зависит от числа различных (неодинаковых записей в таблице). Оптимизатор использует статистику для определения эффективности применения того или иного индекса в запросе.

SET STATISTICS не перестраивает индекс, поэтому свободен от ограничений, налагаемых на ALTER|DROP INDEX, за исключением того, что пересчет статистики имеет право выполнять либо создатель индекса, либо администратор.

Представления

В БД может быть определено представление (просмотр) — виртуальная таблица, в которой представлены записи из одной или нескольких таблиц. Представление реализовано как запрос, хранящийся на сервере и выполняющийся всякий раз, когда происходит обращение к представлению.

Для создания представления применяется оператор

CREATE VIEW <view_name> [(column_view1 [,column_view2 ...])]

AS <select_statement> [WITH CHECK OPTION];

в котором после имени представления следует необязательный список столбцов, оператор <select> есть полнофункциональный оператор SELECT.

Вертикальный срез таблицы

```
CREATE VIEW movie_vert AS
SELECT title, kind
FROM movie;
```

Горизонтальный срез таблицы

```
CREATE VIEW movie_hor AS
SELECT * FROM movie
WHERE year = 1990;
```

После этого к нему можно обращаться как к обычной таблице БД:

```
SELECT * FROM movie_vert;
```

В том числе и для построения другого представления:

CREATE VIEW V1 AS

SELECT title FROM movie_vert WHERE kind='комедия';

Для удаления представления используется оператор

DROP VIEW <view_name>;

Модифицироваать представление нельзя. Можно только удалить и создать его заново.

Порядок формирова¬ния записей в представлении определяется оператором SELECT, а точнее — оптимизатором запросов.

В случае если список имен столбцов опущен, имена столбцов считаются идентичными именам полей, возвращаемых оператором SELECT.

Если в операторе SELECT столбец определяется как выражение, тогда список столбцов обязателен.

Список столбцов будущего представления содержит только их имена. Их тип будет таким, как у столбцов, возвращаемых оператором SELECT. Определяющим т.о. является только соответствие количества столбцов, возвращаемых SELECTом и перечисленных в CREATE VIEW.

1. Создать просмотр, содержащий имя, дату рождения, адрес всех работников кинобизнеса:

```
CREATE VIEW all_persons (fio, bd, adres, rank) AS
    SELECT fio, bd, adres, 'S' FROM prod
      UNION
    SELECT fio, bd, adres, 'P' FROM star
```

2. Создать просмотр, в котором бы для каждого года 21 века содержалось бы количество фильмов, выпущенных в этом году:

```
CREATE VIEW V2 (Y, C) as
  SELECT year, count(*)
  FROM movie
  WHERE year >1999
  GROUP BY year
```

При создании представлений НЕЛЬЗЯ использовать фразу ORDER BY. Сортировки можно добиться только в запросе, использующим данное представление.

Кроме того, в качестве источника данных для формирования представления НЕ МОГУТ выступать хранимые процедуры.

Преимущества создания просмотров:

- сложные запросы, выбирающие данные из множества таблиц становятся проще для понимания, снижая вероятность ошибки;
- просмотр может предоставлять ограниченный набор данных, что важно для обеспечения сохранности данных и, возможно, усиления безопасности: пользователь не будет иметь никакого представления о таблицах, лежащих в основе;
- дав пользователю в распоряжение просмотры, а не таблицы, можно менять запрос и сами таблицы, лежащие в основе просмотра, не меняя клиентских приложений.

Обновляемые и необновляемые представления

Коль скоро представление – это виртуальная таблица, то к нему, как к любой таблице могут применятся операторы INSERT, UPDATE и DELETE. Но представление формируется на основе данных физической таблицы, то изменение данных просмотра невозможно без соответствующего изменения исходных данных. А это не всегда возможно сделать.

Обновляемым (модифицируемым) считается просмотр для который позволяет изменение своих данных. А для этого необходимо выполнение следующих условий:

- просмотр должен формироваться из записей только одной таблицы;
- в просмотр должен быть включен каждый столбец таблицы, имеющий атрибут NOT NULL;
- оператор SELECT просмотра не должен использовать статистических функций, режима DISTINCT, предложения HAVING, соединения таблиц, хранимых процедур и функций, определенных пользователем.

В следующем просмотре:

```
CREATE VIEW notupdate AS
SELECT title, kind FROM movie;
```

нельзя добавлять записи, т.к. при добавлении в новой записи окажутся пустыми остальные поля, определенные как NOT NULL. Изменять и удалять записи из такого просмотра можно.

В следующем просмотре нельзя добавлять, корректировать и удалять записи, т.к. он соединяет две таблицы:

```
CREATE VIEW A AS
SELECT m.title, p.fio
FROM movie m, prod p
WHERE m.id_pr = m.id_pr;
```

Вышеперечисленные условия, ограничивающие модификацию представления, можно обойти используя триггеры.

Использование CHECK OPTION

Если для обновляемого просмотра указан параметр CHECK OPTION, будут отвергаться все попытки добавления новых или изменения существующих записей таким образом, чтобы нарушалось условие WHERE оператора SELECT данного просмотра.

В следующий просмотр нельзя добавить записи со значением поля money более 100 000:

```
CREATE VIEW money_check AS
SELECT *
FROM star
WHERE money <= 100000 WITH CHECK OPTION;
```

Хранимые процедуры

Хранимая процедура — это модуль, написанный на процедурном языке, компилятор которого встроен в ядро Interbase и хранящийся в базе данных как метаданные (то есть как данные о данных). Хранимую процедуру можно вызывать из программы.

Существует две разновидности хранимых процедур:

Процедуры выбора могут возвращать более одного значения. В приложении имя хранимой процедуры выбора подставляется в оператор SELECT вместо имени таблицы или просмотра.

Процедуры действия вообще могут не возвращать данных и используются для реализации какихлибо действий.

Хранимым процедурам можно передавать параметры и получать обратно значения параметров, измененные в соответствии с алгоритмами работы хранимых процедур.

Преимущества использования хранимых процедур:

- одну процедуру можно использоваться многими приложениями;
- разгрузка приложений клиента путем переноса части кода на сервер и вследствие этого — упрощение клиентских приложений;
- при изменении хранимой процедуры все изменения немедленно становятся доступны для всех клиентских приложений; при внесении же изменений в приложение клиента требуется повторное распространение новой версии клиентского приложения между пользователями;
- улучшенные характеристики выполнения, связанные с тем, что хранимые процедуры выполняются сервером, в частности — уменьшенный сетевой трафик.

Хранимая процедура создается оператором

```
CREATE PROCEDURE <stored procedure name>
  [(<in_parametr> <datatype>
  [,<in_parametr > <datatype>...])]
[RETURNS
   (<out_parametr> <datatype>
  [,<out_parametr > <datatype>...])]
AS
[<declare local variable>]
BEGIN
  < statements>
END
```

Входные параметры служат для передачи в процедуру значений из вызывающего приложения. Входные параметры по области видимости приравниваются к локальным переменным, т.е. существуют только в данной процедуре. Изменять значения входных параметров в теле процедуры можно, но следует помнить об их области видимости.

Выходные параметры служат для возврата результирующих значений. Значения выходных параметров устанавливаются в теле

процедуры и после окончания ее рабо¬ты передаются в вызывающее приложение.

И входные и выходные (вместе со словом RETURNS) параметры могут быть опущены, если в них нет необходимости.

Тип должен быть указан для каждого параметра в отдельности.

Следующая хранимая процедура FIND_MAX_LEN возвращает в выходном параметре max_len максимальную длину фильмов режиссера, имя которого передается во входном параметре in_name_prod:

```
CREATE PROCEDURE find_max_len (in_name_prod VARCHAR(30))
RETURNS(max_len INTEGER)
AS
BEGIN
  SELECT MAX(len)
  FROM movie
  WHERE id_pr = (SELECT id_pr FROM prod
                 WHERE fio = :in_name_prod)
  INTO :max_len;
  SUSPEND;
END
```

В приведенной процедуре за ненадобностью отсутствует определение локальных переменных.

Для написания тела хранимой процедуры применяют особый алгоритмический язык. Этот язык применяется также для написания триггеров.

Рассмотрим конструкции алгоритмического языка хранимых процедур и триггеров.

Объявление локальных переменных

Локальные переменные, если они определены в процедуре, имеют срок жизни от начала выполнения процедуры и до ее окончания. Вне процедуры такие локальные переменные неизвестны, и попытка обращения к ним вызовет ошибку. Локальные пе¬ременные используют для хранения промежуточных значений.

Формат объявления локальных переменных:

DECLARE VARIABLE <имя переменной > <тип данных>;

Также как и для параметров, тип должен быть указан для каждой переменной. Нр,

DECLARE VARIABLE A INTEGER, B INTEGER, C INTEGER;

Операторные скобки BEGIN ... END

Операторные скобки BEGIN... END, вопервых, ограничивают тело процедуры, а вовторых, могут использоваться для указания границ составного оператора.

Под простым оператором понимается единичное разрешенное действие, например: Len = 92;

Под составным оператором понимается группа простых или составных операторов, заключенная в операторные скобки BEGIN... END.

Оператор присваивания

Оператор присваивания служит для занесения значений в переменные. Его формат:

Имя переменной = выражение;

где в качестве выражения могут выступать переменные, арифметические и строковые выражения, в которых можно использовать встроенные функции, функции, опреде¬ленные пользователем, а также генераторы. Пример:

OUT_TITLE = UPPER(TITLE);

Оператор IF ... THEN ... ELSE

Условный оператор IF ... THEN ... ELSE имеет такой же формат, как и в Object Pascal:

IF (<условие>) THEN < оператор 1> [ELSE < оператор 2>]

В случае, если условие истинно, выполняется оператор 1, если ложно — оператор 2. В отличие от Object Pascal условие всегда должно заключаться в круглые скобки.

Оператор SELECT

Оператор SELECT используется в хранимой процедуре для выдачи единичной строки. По сравнению с синтаксисом обычного оператора SELECT в процедурный оператор добавлено предложение

INTO :переменная [, :переменная...]

Оно служит для указания переменных или выходных параметров, в которые долж¬ны быть записаны значения, возвращаемые оператором SELECT (те результирующие значения, которые перечисляются после ключевого слова SELECT).

Приводимый ниже оператор SELECT возвращает среднее и сумму по столбцу LENGTH и записывает их соответственно в AVG_LEN и SUM_LEN, которые мо¬гут быть как локальными переменными, так и выходными параметрами процедуры. Расчет среднего и суммы по столбцу LEN производится только для записей, у ко¬торых значение столбца TITLE совпадает с содержимым IN_TITLE (входной пара¬метр или локальная переменная).

```
SELECT AVG(LEN), SUM(LEN)
FROM MOVIE
WHERE TITLE = :IN_TITLE
INTO :AVG_LEN, :SUM_LEN;
```

Использование символа <:> перед именем переменной (или параметра) принято только в операторах внутренних запросов, с тем чтобы отличать имя столбца от имени переменной. В теле ХП вне запросов идентификаторы переменных используются без <:>.

Оператор FOR SELECT ... DO

Оператор FOR SELECT... DO имеет следующий формат:

FOR <оператор SELECT> DO <оператор>

Оператор SELECT представляется в расширенном синтаксисе для алгоритмическо¬го языка хранимых процедур и триггеров, то есть в нем может присутствовать пред¬ложение INTO.

Алгоритм работы оператора FOR SELECT ... DO заключается в следующем. Выполняется оператор SELECT, и для каждой строки полученного результирующего набора данных выполняется оператор, следующий за словом DO. Этим оператором часто бывает SUSPEND (см. ниже), который приводит к возврату выходных параметров в вызывающее приложение.

Следующая процедура выдает все фильмы режиссера, определяемого входным параметром IN_PROD.

```
CREATE PROCEDURE HIS_MOVIES (IN_PROD VARCHAR(30)
CHARACTER SET WIN1251)
```

```
RETURNS (OUT_TITLE VARCHAR(40) CHARACTER SET WIN1251,
         OUT_YEAR INTEGER,
         OUT_LEN INTEGER)
AS
BEGIN
  FOR SELECT TITLE, "YEAR", LEN
    FROM MOVIE M, PROD P
    WHERE (FIO = :IN_PROD) AND (M.ID_PR = P.ID_PR)
    INTO :OUT_TITLE, :OUT_YEAR, :OUT_LEN
  DO SUSPEND;
END
```

Рассмотрим логику работы оператора FOR SELECT... DO этой процедуры. Сначала выполняется оператор SELECT, который возвращает название фильма, его год и длину для каждой записи, у которой столбец FIO содержит значение, идентичное значению во входном параметре IN_PROD. Указанные значения записываются в выходные параметры (соответственно OUT_TITLE, OUT_YEAR, OUT_LEN). Имени параметра в этом случае предшествует двоеточие После выдачи каждой записи результирующего НД выполняется оператор, следующий за словом DO. В данном случае это оператор SUSPEND. Он возвращает значения выходных параметров вызвавшему приложению и приостанавливает выполнение процедуры. До запроса следующей порции выходных параметров от вызывающего приложения.

Такая процедура является процедурой выбора, поскольку она может возвращать множественные значения выходных параметров в вызывающее приложение. Обычно запрос к такой хранимой процедуре из вызывающего приложения осуществляется при помощи оператора SELECT, например:

```
SELECT MAX_LEN FROM FIND_MAX_LEN("Э.Рязанов")
```

Оператор SUSPEND

Оператор SUSPEND передает в вызывающее приложение значения результирующих параметров (перечисленных после слова RETURNS в описании функции), имеющие место на момент выполнения

SUSPEND. После этого выполнение хранимой про¬цедуры приостанавливается. Когда от оператора SELECT вызывающего приложения приходит запрос на следующее значение выходных параметров, выполнение хранимой процедуры возобновляется.

Процедура MOVIESTARLIST возвращает актерский состав каждого фильма (у которых он есть) в виде иерархии:

```
CREATE PROCEDURE MOVIESTARLIST
RETURNS (OUT_PARAM VARCHAR(50))
AS
DECLARE VARIABLE "tmp" VARCHAR(50);
DECLARE VARIABLE ID_M INTEGER;
begin
  for select M."TITLE", M."ID_M"
      from "MOVIE" M
      into :"OUT_PARAM", :"ID_M"
  do
    begin
     if (EXISTS (SELECT *
                 FROM "StarIN"
                 WHERE "ID_M"=:"ID_M")) then
      begin
       suspend;
       "tmp" = "OUT_PARAM";
       for select S."FIO"
           from "StarIN" SI INNER JOIN "STAR" S
               ON S."ID_ST"=SI."ID_S"
           where SI."ID_M"=:"ID_M"
           into :"OUT_PARAM"
       do
         begin
          "OUT_PARAM" = '   ' || "OUT_PARAM";
           suspend;
         end
      end
    end
END
```

Оператор WHILE ... DO

Оператор имеет такой формат:

WHILE (<условие>) DO < оператор>

Алгоритм выполнения оператора: в цикле проверяется выполнение условия; если оно истинно, выполняется оператор. Цикл продолжается до тех пор, пока условие не перестанет выполняться.

Рассмотрим процедуру SUM_0_N, которая подсчитывает сумму всех чисел от 0 до числа, определяемого входным параметром N. Вычисление суммы реализовано в цикле с использованием оператора WHILE...DO.

```
CREATE PROCEDURE BBB (N INTEGER)
RETURNS (S INTEGER)
AS
BEGIN
  WHILE (N>0) DO
   BEGIN
     S=S+N;
     N=N1;
   END
END;
```

Оператор EXIT

Оператор EXIT инициирует прекращение выполнения процедуры и выход в вызывающее приложение.

Процедура MAX_VALUE возвращает максимум из двух чисел, передаваемых как входные параметры; в случае, если одно из чисел имеет значение NULL, процедур завершается (в этом случае выходной параметр содержит значение NULL):

```
CREATE PROCEDURE MAX_VALUE (A INTEGER, B INTEGER)
RETURNS (M_V INTEGER)
AS
BEGIN
  IF (A IS NULL OR B IS NULL) THEN EXIT;
  IF (A > B) THEN M_V = A;
          ELSE M_V = B;
END
```

EXIT пользуются достаточно редко и в основном для того, чтобы прервать цикл при достижении некоторого условия. При этом, последняя извлеченная строка не будет возвращена. Если же эта строка нужна, то стоит воспользоваться последовательностью операторов:

```
SUSPEND;
EXIT;
```

Например, требуется сформировать набор данных: ежемесячные выплаты по кредиту с целью определения количества месяцев, необходимых для полного погашения. Входными параметрами будут: сумма кредита (50000), годовой процент (15), ежемесячная плата (10000).

```
CREATE PROCEDURE KREDIT (
    summa_kredita numeric(15,2),
    procent integer,
    pay_in_month numeric(15,2))
returns (month_num integer, summa numeric(15,2))
AS
declare variable tmp numeric(15,2);
begin
  month_num=0;
  while (0=0) do
    begin

summa_kredita=summa_kreditapay_in_month+summa_kredita*procent/10
0/12;
      summa=summa_kredita;
      month_num=month_num+1;
      if (summa_kredita<0) then exit;
      suspend;
    end
end
```

В зависимости от того, когда вызывается оператор EXIT (до SUSPEND или после) последняя строчка или попадает в результат или нет.

Оператор EXECUTE PROCEDURE

Оператор

EXECUTE PROCEDURE имя [параметр [, параметр ...]];

[RETURNING_VALUES параметр [, параметр ...]];

вызывает другую хранимую процедуру. При этом после имени вызываемой процедуры перечисляются входные параметры, если они есть, а после RETURNING_VALUES выходные параметры.

Перепишем приведенную выше процедуру STAR_LIST таким образом, чтобы из ее тела вызывалась другая процедура, AVG_LEN, возвращающая среднюю длину фильмов указанного жанра:

```
CREATE PROCEDURE AVG_LEN (
   KND VARCHAR(20))
RETURNS (
   OUT_AVG_LEN INTEGER)
AS
BEGIN
  SELECT AVG(LEN)
  FROM MOVIE
  WHERE KIND = :KND
  INTO :OUT_AVG_LEN;
  SUSPEND;
END
CREATE PROCEDURE STAR_LIST1 (IN_KIND VARCHAR(20))
RETURNS (FIO_STAR VARCHAR(20), DOHOD VARCHAR(20))
AS
DECLARE VARIABLE AVG_LENGTH INTEGER;
BEGIN
  EXECUTE PROCEDURE AVG_LEN (IN_KIND)
  RETURNING_VALUES AVG_LENGTH;
  FOR SELECT FIO
    FROM (MOVIE M INNER JOIN "StarIN" SI ON
M.ID_M=SI.ID_M)
        INNER JOIN STAR S ON SI.ID_S=S.ID_ST
    WHERE LEN > :AVG_LENGTH AND Kind=:IN_KIND
    INTO :FIO_STAR, :DOHOD
```

```
  DO
    BEGIN
      IF (DOHOD IS NULL) THEN DOHOD = 'Доход не указан';
      SUSPEND;
    END
END
```

Нетрудно заметить, что EXECUTE PROCEDURE возвращает только единственное значение каждого выходного параметра. Если сама ХП является процедурой выбора, которая возвращает набор данных, то в этом случае выходные параметры получат значения из первой строки этого набора.

Рекурсивные процедуры

Хранимые процедуры могут быть рекурсивными, т.е. вызывать сами себя. Одно из наиболее распространенных применений рекурсивных ХП — обработка древовидных структур. Например, ТОВАРЫ:

```
CREATE TABLE tovar
(id_t INTEGER NOT NULL PRIMARY KEY,
 id_parent INTEGER,
 name VARCHAR (30),
 FOREIGN KEY (id_parent)
   REFERENCES tovar (id_t)
   ON UPDATE CASCADE);
```

Таблица ссылается сама на себя при помощи вторичного ключа id_parent, который является указателем родительского узла дерева.

```
CREATE PROCEDURE GETFULLNAME (
   IN_ID_TOV INTEGER)
RETURNS (
   FULL_NAME VARCHAR(1000),
   ID_CHILD INTEGER)
AS
DECLARE VARIABLE CURR_CHILD_NAME VARCHAR(80);
BEGIN
  FOR SELECT T1.ID_T, T1.NAME
     FROM tovar T1
     WHERE T1.ID_PARENT=:IN_ID_TOV
```

```
    INTO :id_child, :full_name
  DO
   IF (NOT EXISTS (SELECT * FROM tovar
              WHERE tovar.id_parent=:id_child))
   THEN SUSPEND;
   ELSE
    BEGIN
     CURR_CHILD_NAME=full_name;
     FOR SELECT ID_CHILD, FULL_NAME
        FROM GETFULLNAME (:id_child)
        INTO :id_child, :full_name
     DO
       BEGIN
        FULL_NAME=CURR_CHILD_NAME||' '||FULL_NAME;
        SUSPEND;
       END
    END
END

CREATE PROCEDURE TOVAR_RECURSE_GET_TREE (
   IN_ID_T INTEGER,
   OFFSET VARCHAR(18))
RETURNS (
   OUT_NAME VARCHAR(48))
AS
DECLARE VARIABLE ID_T INTEGER;
BEGIN
  FOR SELECT :"OFFSET" || "NAME"
     FROM "TOVAR"
     WHERE "ID_T"=:"IN_ID_T"
     INTO :"OUT_NAME"
  DO  SUSPEND;

  "OFFSET" = "OFFSET" || '  ';
   /* ДЛЯ КАЖДОГО ТОВАРА, ДЛЯ КОТОРОГО
```

```
      ВХОДНОЙ ТОВАР ЯВЛЯЕТСЯ ПРЕДКОМ*/
    FOR SELECT "ID_T", :"OFFSET" || "NAME"
      FROM "TOVAR"
      WHERE "ID_PARENT"=:"IN_ID_T"
      INTO :"ID_T", :"OUT_NAME"
    DO
      IF (EXISTS (SELECT * FROM "TOVAR"
            WHERE "ID_PARENT"=:"ID_T"))
      THEN
        FOR SELECT "OUT_NAME"
          FROM "TOVAR_RECURSE_GET_TREE" (:"ID_T",
:"OFFSET")
          INTO :"OUT_NAME"
        DO SUSPEND;
      ELSE
        SUSPEND;
  END
```

Обработка исключений и ошибок

Эти расширенные возможности позволяют перенести часть бизнеслогики с уровня приложения на уровень БД.

Исключения схожи с исключениями в языках программирования высокого уровня, но со своими особенностями. Исключение Interbase — это сообщение об ошибке, которое имеет собственное, задаваемое программистом имя и текст сообщения об ошибке.

Создается оператором:

CREATE EXCEPTION <имя исключения> <текст исключения>

Например, CREATE EXCEPTION My_Except 'Ошибка'

Удаляются исключения: DROP EXCEPTION <имя исключения>

Изменить: ALTER EXCEPTION <имя исключения> <текст исключения>

Использовать исключение в хранимой процедуре или триггере:

EXCEPTION <имя исключения>

Например, ХП получает два параметра: делимое и делитель и возвращает частное. В случае деления на 0 должно возникнуть предварительно созданное исключение.

```
CREATE EXCEPTION "Div_Zero" 'На ноль делить нельзя!';
CREATE PROCEDURE DELENIE (
   A NUMERIC(15,2),
   B NUMERIC(15,2))
RETURNS (
   C NUMERIC(15,2))
AS
begin
  if (B=0 OR B IS NULL) then
     begin
       EXCEPTION "Div_Zero";
       C=0;
     end
  else C=A/B;
end
```

В результате вызова этой процедуры из приложения с параметром B=0 получим сообщение вида:

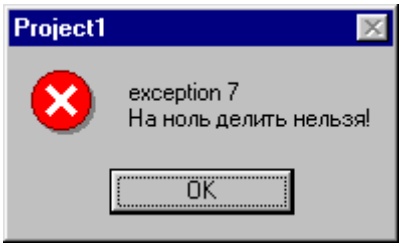


Рис. 10.2. Сообщение об ошибке

Это результат обработки нашего исключения сервером Interbase.

Когда возникает исключение, сервер прерывает работу ХП или триггера и откатывает все изменения, сделанные в текущем блоке BEGIN...END, причем если процедура является процедурой выборкой,

то отменяются действия лишь до последнего оператора SUSPEND, исключая сам SUSPEND.

Например, приведенная ниже тривиальная процедура вернет следующий НД (плюс сообщение исключения):

```
CREATE PROCEDURE NEW_PROCEDURE
RETURNS (NAME VARCHAR(20))
AS
begin
  FOR SELECT "NAME"
     FROM Tovar
     ORDER BY 1
     INTO :NAME
  DO
    BEGIN
      IF (C LIKE 'Од%') then EXCEPTION "My_Except";
      SUSPEND;
    END
end
```

Для обработки исключений существует конструкция алгоритмического языка:

```
WHEN EXCEPTION <Имя исключения> DO
BEGIN
  /*обработка исключения */
END
```

Использование этой конструкции позволяет избежать сообщения об ошибке и реализовать собственные действия по обработке исключения.

Алгоритм действия следующий: когда происходит исключение выполнение ХП или триггера прерывается и Interbase ищет конструкцию WHEN EXCEPTION ... DO в текущем блоке BEGIN...END. Если не находит — поднимается на уровень выше (в смысле вложенных блоков BEGIN...END или вызова ХП одной из другой) и ищет обработчик там. И т.д. пока не закончится вложенность уровней ХП или не будет найден обработчик. В первом случае будет

выдано стандартное сообщение об ошибке, содержащее текст исключения. Если обработчик найден, то выполняются действия в его блоке после DO, а затем управление передается на оператор следующий за обработчиком.

Предыдущий пример с обработчиком:

```
CREATE PROCEDURE NEW_PROCEDURE
RETURNS (NAME VARCHAR(20))
AS
begin
  FOR SELECT "NAME"
     FROM Tovar
     ORDER BY 1
     INTO :NAME
  DO
    BEGIN
     if (NAME LIKE 'Од%') then EXCEPTION "My_Except";
     SUSPEND;
     WHEN EXCEPTION "My_Except" DO
      begin
       NAME='аджедО';
       SUSPEND;
      end
    END
end
```

Если опустить выделенный оператор SUSPEND, то значение 'аджедО' не будет передано в НДрезультат.

Что за исключение "My_Except" — не принципиально.

Обработчик можно использовать также и для обработки ошибок Interbase по тому же принципу: сервер ищет обработчик последовательно по всем уровням вложенности, начиная с того, на котором возникла ошибка. Конструкция обработчика:

```
WHEN GDSCODE|SQLCODE <код_ошибки> DO
BEGIN
  /*обработка ошибки */
END
```

В зависимости от GDSCODE|SQLCODE обрабатываются ошибки либо Interbase, либо SQL. Например, попытка указать значение ID_PARENT отличное от NULL и не являющееся одним из значений ID_T (первичного ключа),

```
CREATE PROCEDURE WHEN_SQLCODE_DO
AS
Begin
  INSERT INTO TOVAR VALUES (NULL, 100, 'Проба', 0);
end
```

приводит к ошибке:

```
violation of FOREIGN KEY constraint "INTEG_25" on table "TOVAR"
```

Номер этой ошибки (см. "www.ibase.ru/v6/doc/Langref.pdf") — 530. Поэтому добавляем в процедуру обработчик SQLошибки 530.

```
CREATE PROCEDURE WHEN_SQLCODE_DO
AS
begin
  INSERT INTO TOVAR VALUES (NULL, 100, 'Проба', 0);
  WHEN SQLCODE 530 DO
    INSERT INTO TOVAR VALUES (NULL, NULL, 'Неудачная вставка', 0);
end
```

Таким образом, внутри ХП можно "перехватить" практически любую ошибку и нужным образом на нее отреагировать. Можно перечислить несколько обработчиков, чтобы определить реакцию на различные ошибки, а можно описать один обработчик на все возможные ошибки SQL и Interbase и исключения. Для этого существует конструкция:

```
WHEN ANY DO
BEGIN
  /*действия при любой ошибке или исключении */
END
```

С помощью использования описанных механизмов можно сделать приложения БД значительно более отказоустойчивыми и дружественными.

Генераторы

Часто в состав первичного или уникального ключа входят цифровые поля, значения ко¬торых должны быть уникальны, то есть не повторяться ни в какой другой записи таблицы. В одних случаях такое значение является семантически значимым и формируется пользовате¬лем по определенному алгоритму — например, номер лицевого счета в банке. В других случаях лучше предоставить выработку такого значения приложению или серверу БД.

В InterBase отсутствует аппарат автоинкрементных столбцов. Вместо этого для ус¬тановки уникальных значений столбцов можно использовать аппарат генераторов.

Генератором называется хранимый на сервере БД механизм, возвращающий уни¬кальные значения, никогда не совпадающие со значениями, выданными тем же самым генератором в прошлом.

Для создания генератора используется оператор

CREATE GENERATOR <ИмяГенератора>;

Для генератора необходимо установить стартовое значение при помощи оператора

SET GENERATOR <ИмяГенератора> TO <СтартовоеЗначение>;

При этом СтартовоеЗначение должно быть целым числом.

Для получения уникального значения к генератору нужно обратиться с помощью функции

GEN_ID (<ИмяГенератора>, <шаг>) ;

Эта функция возвращает увеличенное на шаг предыдущее значение, выданное генера¬тором (или увеличенное на шаг стартовое значение, если ранее обращений к генерато¬ру не было). Значение шага должно принадлежать диапазону –231...+231 1.

Замечание. Не рекомендуется переустанавливать стартовое значение генера¬тора или менять шаг при разных обращениях к GEN_ID. В противном случае генератор может выдать неуникальное значение, и как следствие будет возбу¬ждено исключение при попытке запоминания новой записи в ТБД.

Пусть в БД определен генератор, возвращающий уникальное значение для столбца ID_T в таблице TOVAR:

CREATE GENERATOR GEN_TOVAR_ID;

SET GENERATOR GEN_TOVAR_ID TO 23;

Обращение к генератору непосредственно из оператора INSERT:

INSERT INTO PROD VALUES (GEN_ID(GEN_TOVAR_ID, 1), 14, "Ушанка", 20);

Триггеры

Триггер — это особый вид хранимой процедуры, автоматически вызываемой SQLсервером при обновлении, удалении или добавлении новой записи. Непосредственно вызвать триггер нельзя. Нельзя и передавать им входные параметры и получать от них значения выходных параметров. Триггеры всегда реализуют действие.

Триггер всегда привязан к определенной таблице.

По событию ТБД триггеры различаются на вызываемые при:

- добавлении новой записи;
- изменении существующей записи;
- удалении записи;
- универсальные

По отношению к событию, влекущему их вызов, триггеры различаются на:

- выполняемые до наступления события;
- выполняемые после наступления события.

В клоне Yaffil 1.0 реализована поддержка универсальных триггеров, срабатывающих при любой операции.

Преимущества использования триггеров:

- Автоматическое обеспечение каскадных воздействий в дочерних таблицах при изменении, удалении записи в родительской таблице выполняется на сервере. Пользователю нет необходимости заботиться о программной реализации каскадных воздействий. Поскольку каскадные воздействия выполняет сервер, нет необходимости пересылать изменения в таблицах БД из приложения на сервер, что снижает загрузку сети.
- Изменения в триггерах не влекут необходимости изменения программного кода в клиентских приложениях и не требуют распространения новых версии клиентских приложений.

Замечание. При откате транзакции откатываются также и все изменения, внесенные в БД триггерами.

Создание триггеров

Триггер создается оператором

CREATE TRIGGER ИмяТриггера FOR ИмяТаблицы

[ACTIVE | INACTIVE]

{BEFORE | AFTER}

{DELETE | INSERT | UPDATE)

[POSITION номер]

AS <тело триггера>

- ACTIVE | INACTIVE — указывает, активен триггер или нет. Можно определить триггер «про запас», установив для него INACTIVE. В дальнейшем можно переопределить триггер как активный. По умолчанию действует ACTIVE.
- BEFORE | AFTER — указывает, будет выполняться триггер до (BEFORE) или после (AFTER) запоминания изменений в БД.
- DELETE | INSERT | UPDATE — указывает операцию над ТБД, при выполнении которой срабатывает триггер.
- POSITION номер — указывает, каким по счету будет выполняться триггер в случае наличия группы триггеров, обладающих одинаковыми характеристиками операции и времени (до, после операции) вызова триггера. Значение номера задается числом в диапазоне 0..32767. Триггеры с меньшими номерами выполняются раньше.

Например, если определены триггеры

CREATE TRIGGER A FOR Movie BEFORE INSERT POSITION 1 ...

CREATE TRIGGER C FOR Movie BEFORE INSERT POSITION O...

CREATE TRIGGER D FOR Movie BEFORE INSERT POSITION 44 ...

CREATE TRIGGER B FOR Movie AFTER INSERT POSITION 100 ...

CREATE TRIGGER E FOR Movie AFTER INSERT POSITION 44 ...

для операции добавления новой записи в таблицу MOVIE они будут выполнены в последовательности C, A, D, E, B.

Для определения тела триггера используется процедурный язык. В него добавляется возможность доступа к старому и новому значениям столбцов изменяемой записи OLD и NEW — возможность, недоступная при определении тела хранимых процедур. Структура тела триггера:

[<объявление локальных переменных>]

```
BEGIN
  < оператор>
END
```

Определение заголовка триггера

Заголовок триггера имеет формат

```
CREATE TRIGGER ИмяТриггера FOR ИмяТаблицы
[ACTIVE | INACTIVE]
{BEFORE | AFTER}
{DELETE | INSERT | UPDATE)
[POSITION номер]
```

Значения OLD и NEW

Значение OLD.ИмяСтолбца позволяет обратиться к состоянию столбца, имевшему место до внесения возможных изменений, а значение NEW.ИмяСтолбца — к состоянию столбца после внесения изменений.

В том случае, если значение в столбце не изменилось, OLD.ИмяСтолбца будет равно NEW.ИмяСтолбца.

Следующий триггер, используя генератор, присваивает уникальное значение ключевому полю ID_T, но только в том случае, если пользователь сам не указал значения идентификатора:

```
CREATE TRIGGER TRIG_TOVAR_BI FOR TOVAR
ACTIVE BEFORE INSERT POSITION 0
AS
begin
  IF (NEW."ID_T" IS NULL) THEN NEW."ID_T" = GEN_ID ("GEN_TOVAR_ID",1);
end
```

Теперь оператор добавления записей может выглядеть следующим образом:

```
INSERT INTO PROD (ID_PARENT, NAME, PRICE)
VALUES (14, "Ушанка", 20);
```

Здесь не указано значение первичного ключа. Перед вставкой этой записи сработает триггер TRIG_TOVAR_BI, который подставит новое значение генератора в поле ID_T, т.к. его значение не указано (NULL).

Самостоятельно: в этом же триггере корректно обработать ситуацию, когда значение первичного ключа указано, но оно не является уникальным.

Использование контекстных переменных иллюстрирует следующая таблица:

	NEW	OLD
BEFORE INSERT	Full Access	No access
AFTER INSERT	Read Only	No access
BEFORE UPDATE	Full Access	Read Only
AFTER UPDATE	Read Only	Read Only
BEFORE DELETE	No access	Read Only
AFTER DELETE	No access	Read Only

Примеры использования триггеров

Бизнесправило: в таблице товаров изменять цену можно только для конечных товаров (листьев дерева) и при этом цена всех родительских товаров должна изменяться автоматически. Написать триггеры, обеспечивающие эти правила.

```
CREATE EXCEPTION "NoPriceForParent"
'Нельзя изменять цену для родительского узла';
CREATE TRIGGER "No_Update_For_Parent" FOR TOVAR
ACTIVE BEFORE UPDATE POSITION 0
AS
begin
  if (
    (NEW."Price"<>OLD."Price")
    AND
    ...
    AND
    EXISTS (select * from tovar where tovar."ID_PARENT"=OLD."ID_T")
    )
```

```
  then
    EXCEPTION "NoPriceForParent";
end
```

Триггер отслеживает изменение только цены (смена, например, наименования бизнесправилами никак не оговаривается) и проверяет существование таких записей, для которых наша текущая запись является предком. При совместном соблюдении этих двух условий вызывается предварительно созданное исключение, которое откатывает изменение записи.

Триггер для каскадного изменения цены родительских объектов:

```
CREATE TRIGGER "Change_Price" FOR TOVAR
ACTIVE AFTER UPDATE POSITION 0
AS
DECLARE VARIABLE delta INTEGER;
DECLARE VARIABLE tmp INTEGER;
begin
  delta=NEW."Price"OLD."Price";
  if (delta<>0) then
   begin
     select "Price"
     From Tovar
     Where ID_T=OLD."ID_PARENT"
     into :tmp;
     ...
     update tovar set "Price"=:tmp+:delta where
"ID_T"=OLD."ID_PARENT";
    end
end
```

Триггер срабатывает после изменения цены конечного товара, вычисляет разницу (delta) и если цена изменилась, то в переменной tmp сохраняется цена родительского товара, а затем изменяет ее на величину delta.

Здесь, последний оператор UPDATE наталкивается на действие триггера NoPriceForParent, который запрещает изменение цены

родительских узлов. Первое, что напрашивается, это изменение активности триггера NoPriceForParent:

ALTER TRIGGER "No_Update_For_Parent" INACTIVE;

но, это оператор относится к DDL и не может быть использован в ХП и триггерах.

Управлять состоянием активности триггеров можно недокументированным способом путем модификации системных таблиц:

```
UPDATE rdb$triggers trg
SET trg.rdb$trigger_inactive=1
WHERE trg.rdb$trigger_name='No_Update_For_Parent';
```

Однако, для нашей задачи этот метод тоже не подойдет. Дело в том, что триггеры работают в рамках той же транзакции, что и вызвавшее их изменение. Поэтому, если один триггер изменит состояние другого, то мехнизм "активных таблиц" который занимается запуском триггеров не увидит эти изменения, т.к. они еще не подтверждены.

В качестве решения этой проблемы можно предложить исключение из бизнесправила: цену родительского объекта можно изменять только на значение 1 и со значения 1. Используя эту "дыру" триггеры будут выглядеть:

```
CREATE TRIGGER "Change_Price" FOR TOVAR
ACTIVE AFTER UPDATE POSITION 0
AS
DECLARE VARIABLE delta INTEGER;
DECLARE VARIABLE tmp INTEGER;
begin
  delta=NEW."Price"OLD."Price";
  if (delta<>0) then
   begin
     select "Price"
     From Tovar
     Where ID_T=OLD."ID_PARENT"
     into :tmp;
     update tovar set "Price"=1 where "ID_T"=OLD."ID_PARENT";
```

```
        update tovar set "Price"=:tmp+:delta where
"ID_T"=OLD."ID_PARENT";
      end
    end
    CREATE TRIGGER "No_Update_For_Parent" FOR TOVAR
    ACTIVE BEFORE UPDATE POSITION 0
    AS
    begin
      if (
        (NEW."Price"<>OLD."Price")
        AND
        not (OLD."Price"=1 OR NEW."Price"=1)
        AND
        EXISTS (select * from tovar where
tovar."ID_PARENT"=OLD."ID_T")
        )
      then
        EXCEPTION "NoPriceForParent";
    end
```

Ведение журнала изменений

Журнал изменений в БД представляет собой таблицу БД, в которой фиксируются действия над всей базой данных или отдельными ее таблицами. В многопользовательских системах ведение такого журнала позволяет определить источник недостоверных или искаженных данных.

Определим в базе данных таблицу MOVIE_LOG:

```
    CREATE TABLE "MOVIE_Log" (
       "Time"       TIMESTAMP,
       "User"       VARCHAR(31),
       "Column"     VARCHAR(15),
       "Old_Value"  VARCHAR(100),
       "New_Value"  VARCHAR(100),
       "Action"     VARCHAR(3) CHECK ("Action" IN ('INS', 'UPD',
'DEL')),
```

```
    "ID_M"      INTEGER,
    CONSTRAINT "FK_Log" FOREIGN KEY (ID_M)
    REFERENCES MOVIE (ID_M) ON DELETE CASCADE ON
UPDATE CASCADE
    );
```

в которую будем автоматически записывать любые изменения, добавления, удаления в таблице MOVIE. При этом будем фиксировать время (Time), пользователя (USER), измененный столбец (COLUMN), старое и новое значение столбца (OLD_VALUE и NEW_VALUE), операцию (INS, UPD, DEL) над таблицей, а также идентификатор той записи, над которой происходит изменение (ID_M).

```
CREATE TRIGGER MOVIE_AI0 FOR MOVIE
ACTIVE AFTER INSERT POSITION 0
AS
BEGIN
  INSERT INTO "Log"
  VALUES (current_timestamp, current_user, 'MOVIE', null, null, null,
       'INS', NEW."ID_M");
END
CREATE TRIGGER MOVIE_AU0 FOR MOVIE
ACTIVE AFTER UPDATE POSITION 0
AS
begin
 if (OLD."ID_M"<>NEW."ID_M") then
   INSERT INTO "Log"
   VALUES (current_timestamp, current_user, 'ID_M',
        OLD."ID_M", NEW."ID_M", 'UPD', NEW."ID_M");

 if (OLD."TITLE"<>NEW."TITLE") then
   INSERT INTO "Log"
   VALUES (current_timestamp, current_user, 'TITLE',
        OLD."TITLE", NEW."TITLE", 'UPD', NEW."ID_M");

 if (OLD."ID_PR"<>NEW."ID_PR") then
   INSERT INTO "Log"
```

```
    VALUES (current_timestamp, current_user, 'ID_PR',
        OLD."ID_PR", NEW."ID_PR", 'UPD', NEW."ID_M");

  if (OLD."YEAR"<>NEW."YEAR") then
    INSERT INTO "Log"
    VALUES (current_timestamp, current_user, 'YEAR',
        OLD."YEAR", NEW."YEAR", 'UPD', NEW."ID_M");

  if (OLD."LEN"<>NEW."LEN")then
    INSERT INTO "Log"
    VALUES (current_timestamp, current_user, 'LEN',
        OLD."LEN", NEW."LEN", 'UPD', NEW."ID_M");
  else
    if ((OLD."LEN" IS NULL) OR (NEW."LEN" IS NULL)) then
      INSERT INTO "Log"
      VALUES (current_timestamp, current_user, 'LEN',
          OLD."LEN", NEW."LEN", 'UPD', NEW."ID_M");

  if (OLD."KIND"<>NEW."KIND") then
    INSERT INTO "Log"
    VALUES (current_timestamp, current_user, 'KIND',
        OLD."KIND", NEW."KIND", 'UPD', NEW."ID_M");
END

CREATE TRIGGER PROD DEL_LOG FOR PROD
ACTIVE
AFTER UPDATE
AS
BEGIN
  INSERT INTO PROD_LOG(DAT_IZM, ACT, OLD_TQVAR,
NEW_FIO)
  VALUES ("NOW","DEL",OLD.FIO,"") ;
END
```

Пусть в таблицу MOVIE внесены некоторые изменения. Тогда, выполнив оператор

SELECT * FROM MOVIE_LOG;

получим историю изменений таблицы MOVIE:

DAT_IZM	DEISTV	OLD_FIO	NEW_FIO
30-JUN-1997	ADD		И. Рязанов
30-JUN-1997	UPD	И. Рязанов	Э. Рязанов
30-JUN-1997	DEL	Э. Рязанов	

Функции, определяемые пользователем

В InterBase есть возможность использовать функции, определяемые пользователем (User Defined Functions).

Функция должна находится в DLL и регистрироваться в БД:

```
DECLARE EXTERNAL FUNCTION Имя_функции
[список входных параметров]
RETURNS {тип_данных [BY VALUE] | CSTRING (Длина)}
ENTRY_POINT "Имя_функции_в_DLL"
MODULE_NAME "Имя_файла_DLL"
```

Список входных параметров необязателен, он задается указанием типа данных для каждого параметра, через запятую. После RETURNS указывается тип передаваемого результата и способ его передачи: по значению (BY VALUE) или по ссылке (тип CSTRING соответствует типу PCHAR).

Текст библиотеки:

```
library Project1;
uses
  SysUtils, Classes;
{$R *.RES}
function Upper_Rus (InStr: PChar): PChar; cdecl; export;
begin
  Result:=PChar(ANSIUpperCase(String(InStr)))
end;
exports
   Upper_Rus;
begin
end.
```

После компиляции Project1.dll размещается в каталоге UDF (для версий Interbase 6.x и клонов) или там же где размещается сам сервер C:\Program Files\InterBase Corp\Interbase\bin\)

Затем пишется запрос:

```
DECLARE EXTERNAL FUNCTION UPPER_RUS
CSTRING (256)
RETURNS CSTRING (256)
ENTRY_POINT 'UPPER_RUS'
MODULE_NAME 'PROJECT1'
```

Теперь можно обращаться к функции:

```
SELECT UPPER_RUS(Name) FROM Tovar;
```

Лекция 11. ФУНКЦИИ ЗАЩИТЫ БАЗЫ ДАННЫХ

Транзакции и параллелизм

При работе СУБД возникает необходимость защиты БД от возможных случайных или преднамеренных ситуаций, когда существует вероятность потери данных. Например, при доступе к БД сразу нескольких пользователей возможно повреждение или неправильная запись данных, что в свою очередь может привести к непредсказуемым последствиям. Очевидно, что из таких ситуаций СУБД должна уметь корректно выходить. Одним из способов решения этих проблем является механизм транзакций.

Поддержание механизма транзакций — показатель уровня развитости СУБД. Корректное поддержание транзакций одновременно является основой обеспечения целостности БД. Транзакции также составляют основу изолированности пользователей во многопользовательских системах.

Под транзакцией понимается неделимая с точки зрения воздействия на БД логическая последовательность операторов манипулирования данными (чтения|SELECT, удаления|DELETE, вставки|INSERT, модификации|UPDATE) такая что возможны два итога:

- результаты всех операторов, входящих в транзакцию, соответствующим образом отображаются в БД;
- воздействие всех этих операторов полностью отсутствует.

Классическим примером транзакции, переводящей БД из одного целостного состояния в другое является бухгалтерская проводка, когда некоторая сумма S должна быть списана со счета A и занесена на счет B. Только успешное выполнение этих двух операций гарантирует целостность информации в БД. Если сумма S снимется со счета A, затем произойдет сбой, то сумма не будет записана на счет B. Поэтому, в этом случае, результаты первой операции должны быть отменены.

Существуют некоторые свойства, которыми должна обладать любая из транзак¬ции. Ниже представлены четыре основных свойства (ACID — аббревиатура, состав¬ленная из первых букв их английских названий).

– Атомарность. Это свойство типа "все или ничего". Любая транзакция представляет собой неделимую единицу работы, которая может быть либо выполнена вся целиком, либо не выполнена вовсе.
– Согласованность. Каждая транзакция должна переводить базу данных из одного согласованного состояния в другое согласованное состояние.
– Изолированность. Все транзакции выполняются независимо одна от другой и промежуточные результаты незавершенной транзакции не доступны другим транзакциям.
– Продолжительность. Результаты успешно завершенной (зафиксированной) транзакции должны сохраняться в базе данных постоянно и не должны быть утеряны в результате последующих сбоев.

Все операции в БД осуществляются в рамках транзакции, как минимум одной.

Даже если разработчик клиента БД и не использует транзакции явно, это лишь означает, что работу по управлению транзакциями брал на себя инструмент разработки. В простейшем случае транзакция запускалась и подтверждалась одновременно с запуском приложенияклента. Поэтому, при аварийном завершении программы происходил откат транзакции и потеря всех изменений.

Определить какие именно операции должны входить в состав транзакции (другими словами определить моменты начала и окончания транзакции) в состоянии только само приложениеклиент или пользователь этого приложения исходя из бизнеслогики (см. пример).

По умолчанию, начало и конец транзакции совпадают с началом и окончанием работы приложения, т.е. все операции, выполняемые приложением считаются выполняющимися в рамках одной транзакции.

Итак, транзакция может закончится или подтверждением (COMMIT) произведенных ей изменений или откатом всех изменений (ROLLBACK). При этом решение о том или ином исходе никак не связано с успешностью или неуспешностью операций внутри транзакции.

Ничто не может помешать пользователю откатить результаты успешных действий исходя из своих высших соображений и наоборот, подтвердить результаты неуспешных.

Понятие транзакции имеет непосредственную связь с понятием целостности БД. Очень часто БД может обладать такими ограничениями целостности, которые просто невозможно не нарушить, выполняя изменение только одним оператором.

Для поддержания подобных ограничений целостности допускается их нарушение внутри транзакции с тем условием, чтобы к моменту завершения транзакции условия целостности должны быть соблюдены. Но, здесь вступает в силу свойство изолированности транзакций, говорящее, что изменения внутри транзакции (т.е. до их подтверждения) не видны в других транзакциях.

Реализация механизма транзакций различна. В INTERBASE реализована многоверсионная архитектура (MGA — Multi Generation Architecture). Суть ее в том, что любые действия над записью производятся не над самой записью, а над ее копией (версией). Когда транзакция изменяет запись, для нее создается копия над которой и совершаются изменения в рамках этой транзакции. При этом изменять данные может только одна транзакция, остальные могут только читать. Когда изменяющая транзакция подтверждается, исходная версия записи помечается для удаления, а текущая версия становится основной.

Каждая транзакция при запуске получает уникальный идентификатор (transaction ID | TID), при помощи которого все транзакции учитываются на странице учета транзакций (Transaction Inventory Page | TIP). Таким образом, для каждой версии записи указывается какой транзакции она принадлежит, а также указатель на следующую версию той же записи.

Транзакция может быть в одном из четырех состояний: активная (active), подтвержденная (commited), отмененная (rolled back) и с неопределенным статусом (limbo /при двухфазном подтверждении/).

Когда, транзакция commeted, то в БД не происходит никаких изменений, а лишь изменяется состояние транзакции с активной на подтвержденную на TIP. Поэтому читающие транзакции, прежде чем

прочитать запись, должны пройтись по ее существующими версиям и взять из подтвержденных версий ту, TID которой больше (т.е. последнюю).

Если же транзакция rolled back то, ее версия записи помечается для удаления и считается мусором (garbage) и подлежат удалению, но удаляются они не той транзакцией, которая породила мусор, а следующей за ней, которая ищет последнюю подтвержденную версию записи и во время поиска удаляет все старые версии записи. Этот процесс называется "сборкой мусора" и является кооперативным, т.е. им занимаются все транзакции, в т.ч. и читающие.

При этом если, транзакция не выполнила никаких изменений (по флагу Update Flag) и завершается отменой, то вместо roll back вызывается commit, что бы не нагружать сервер.

Когда транзакция удаляет запись, запись лишь помечается для удаления и после подтверждения транзакции станет мусором.

Таким образом, мусором являются:

- старые версии записей, когда подтвердилась изменяющая транзакция и сделала свою версию основной;
- отмененные изменения, когда изменяющая или добавляющая транзакция откатилась, оставив свою версию записи;
- удаленные записи.

Транзакция может быть запущена в двух режимах: чтения/записи (write) или только чтения (read). Первый режим устанавливается по умолчанию. Транзакции, которые не изменяют данных и запущены в режиме чтения меньше нагружают сервер, т.к. не создают лишних версий записей.

Существует такое понятие как конфликт. Нр, когда две транзакции пытаются изменить одну и ту же запись. Поэтому транзакция, которая обратилась к записи первой, блокирует ее. Тогда вторая транзакция должна будет либо немедленно возбудить исключение, либо подождать некоторое время: вдруг первая транзакция завершится и разблокирует доступ к записи. Отсюда вытекают два режима блокировки: wait (по умолчанию) и nowait. Рассмотрим наиболее типичные случаи блокировок:

1. Транзакция «T1» читает запись, транзакция «T2» также читает эту запись. На первый взгляд конфликта не возникает. Но на самом деле все будет зависить от уровня изоляции этих транзакций. Но об этом ниже.

2. Транзакция «T1» вставляет запись, но еще не завершилась. Транзакция «T2» также пытается вставить запись с тем же самым первичым ключом. Здесь все зависит от режима блокировки:

- если «T2» запущена в режиме wait, то она будет ожидать завершения «T1», и если «T1» завершится подтверждением commit, то вставляемая «T2» запись будет признана ошибочной; а если «T1» завершится откатом rollback, то «T2» сможет осуществить вставку.
- если «T2» запущена в режиме nowait, то немедленно возникнет ошибка lock conflict on no wait transaction.

3. Транзакция «T1» изменяет запись, но еще не подтвердилась. Транзакция «T2» пытается удалить или тоже изменить эту же запись. Здесь также влияет режим блокировки:

- если «T2» запущена в режиме wait, то она будет ждать завершения «T1»; если «T1» подтверждается, то в «T2» возникнет ошибка «Deadlock — update conflict with concurrent update»; а если «A» откатится, то «T2» сможет подтвердится.
- если «T2» запущена в режиме nowait, то немедленно возникнет ошибка lock conflict on no wait transaction.

4. Взаимоблокировка (deadlock — мертвая блокировка).

- Транзакция «T1» блокирует запись A и работает с ней;
- «T2» блокирует запись B и работает с ней;
- следующим шагом в «T1» является работа с записью B, она пытается получить к ней доступ, но поскольку запись заблокирована «T2», то «T1» будет ждать снятия блокировки.
- следующим шагом в «T2» является работа с записью A, и она также перейдет в режим ожидания.

Из ситуации взаимобокировки, любой сервер БД должен уметь выходить. Самый простой способ — откатить любую из конфликтующих транзакции. Еще вариант решения — откатить обе транзакции, а затем запустить их заново с другим интервалом (метод временных меток). Приоритет будет иметь та транзакция, момент запуска которой (временная метка) более ранний. Более сложный путь

— попытка предварительно распознавать транзакции, которые могут войти во взоимобокировку (т.н. оптимистическая технология).

Понятно, что на практике ситуцаия взаимных блокировок транзакций возникает крайне редко, поэтому из соображений производительности, механизм решения взаимоблокировки не запускается сразу в момент конфликта, а по истечении некоторого интеравала времени (параметр DEADLOCK_TIMEOUT в конфигурации сервера Interbase).

Имеет место некоторая непонятность: DEADLOCK — взаимоблокировка, а в сообщениях об ошибках, возникающих при конфликтах далеких от взаимного блокирования, это слово также обозначает конфликт вообще.

Как говорилось выше, транзакции обладают свойством изолированности, т.е. происходящие в рамках транзакции изменения не видны из вне (т.е. другими транзакциями) до подтверждения транзакции.

Уровень изолированности транзакции определяет какие изменения, сделанные в других транзакциях будут видны в данной транзакции.

Установка параметров транзакций (уровня изоляции, режима записи, режима блокировки) осуществляется при помощи перечисления набора констант:

Параметры	Константа	Описание
Уровень изоляции	read_commited rec_version	Возможность читать подтвержденные записи других транзакций, в том числе и те которые заблокированы (разумеется только последнюю подтвержденную версию)
	read_commited no_rec_version	Также видит все подтвержденные изменения, сделанные другими транзакциями, но НЕ может читать записи, имеющие неподтвержденные версии (т.е. заблокированные).
	concurrency	Видит состояние БД только на момент запуска. Не видит никаких изменений, сделанных другими транзакциями.
	consistency	Аналогично уровню concurrency, но таблица (целиком) блокируется на запись другими транзакциями.
Режим доступа	read	Транзакция может только читать
	write	Разрешает запись в рамках транзакции
Режим блокировки	wait	Отсроченное разрешение конфликтов
	nowait	Не ждать разрешения конфликта

В Interbase таких уровня три:

1. READ COMMITTED (читать подтвержденные данные). Транзакция с данным уровнем изоляции видит все результаты всех подтвержденных транзакций. Этот уровень используется для получения самого «свежего» состояния БД. Существует две разновидности такого чтения:

- rec_version. Этот вариант используется по умолчанию и означает, что при чтении записи считывается последняя версия каждой записи независимо есть неподтвержденные версии или нет.
- no_rec_version. Этот вариант требует, чтобы на момент чтения записи у нее не существовало неподтвержденных версий. При чтении записи в такой транзакции производится проверка не существует ли у этой записи неподтвержденной версии. Если

существует, то наша транзакция ждет, пока не завершится транзакция, изменяющая эту запись, при условии, что наша запущена в режиме wait. Если же в nowait, то немедленно возникнет ошибка Deadlock. Очевидно, что такой уровень может привести к частым конфликтам, поэтому использовать его нужно с большой осторожностью.

2. SNAPSHOT. Задается параметром concurrency. Транзакция с данным уровнем изоляции не видит никаких изменений (кроме своих конечно), видит только состояние БД на момент своего запуска (как бы моментальный снимок БД). При попытке в этой транзакции изменить данные, измененные другими транзакциями уже после ее запуска возникает конфликт. Этот режим обычно используется для длительных по времени запросов, для генерации отчетов.

3. SNAPSHOT TABLE STABILITY. Задается параметром consistency. Уровень изоляции почти такой же что и SNAPSHOT, но дополнительно блокирует таблицу на запись. Т.е. если наша транзакция с уровнем consistency производит изменения в таблице, то транзакции с уровнем read_commited или concurency смогут только читать эту таблицу, а транзакции с уровнем consistency не смогут даже читать. Обычно этот уровень используют только для коротких обновляющих транзакций, которые запусаются, проводят очень короткое по времени изменение и сразу завершаются. Другие транзакции в зависимости от режима блокировки wait|nowait либо ждут, либо возбуждают исключение.

Практические аспекты использования транзакций нужно решать для каждой задачи индивидально. Обычно все запросы к БД можно разделить на:

- запросы на чтение самого «свежего» состояния БД;
- запросы не текущие изменения;
- запросы на чтение вспомогательных таблиц;
- запросы на чтение данных для построения отчетов и т.д.

Для рассматриваемого нами примера ФИЛЬМЫ, имеется форма с сеткой DBGrid, где простматривается таблица MOVIE с невозможностью редактирования данных в сетке. Очевидно что здесь требуется иметь самые свежие данные, поэтому параметры этой транзакции следует выбрать:

read
read_commited
rec_version
nowait

При таких параметрах производится чтение всех подтвержденных другими транзакциями записей причем без конфликтов в другими читающими или пишущими транзакциями. Такую транзакцию можно долго держать открытой — серевер не нагружается версиями записей.

Далее: для изменения данных используется отдельная форма, на которую вынесены компоненты для редактирования. Запрос на изменение одной записи должен быть очень коротким, чтобы свести к минимуму возможность конфликтов, которые можно отслеживать как исключения при помощи блока try... except. Параметры такой транзакции будут такими:

write
consistensy
nowait

Нам не нужно в рамках этой транзакции самых свежих данных, что только замедляет выполнение. Также мы сможем сразу узнать о конфликте, если запись заблокирована и запретить изменение/удаление записи другими транзакциями, в которых возникнет ошибка.

Для запросов, которые строят отчеты (особенно для тех, которые делают несколько проходов по таблице) необходимо видеть только те данные, которые существовали на момент старта запроса и будет неизменными за все время его выполнения. Параметры:

read
concurrency
nowait

Как уже говорилось, все действия в БД осуществляются в рамках транзакций, однако существуют объекты, которые находятся вне контекста транзакций. Это генераторы и внешние таблицы.

Генераторы располагаются на специально для них выделенной странице и все транзакции видят значения генераторов в либой момент

времени. Это позволяет организовать бесконфликтные конкурентные вставки в параллельно выполняющихся транзакциях.

Внешние таблицы представляют собой файлы, находящиеся за пределами основного файла БД. Над ними разрешены только операции INSERT/SELECT. Отсутствие обновлений и позволяет отказатся от версионной структуры и, следовательно, от механизма транзакций.

Существует такое понятие как двухфазное подтверждение транзакций. Interbase позволяет организовать распределенные транзакции между разными БД и даже между разными серверами. Т.е. клиент может запустить транзакцию сразу на двух серверах. При этом возможна ситуация, когда в рамках одной БД транзакция способна подтветдится, а на другой — нет. Чтобы синхронизировать этот процесс вводится особое состояние Prepared, которое говорит о том, что транзакция готова подтвердится, когда на другом сервере транзакция тоже будет в состоянии Prepared. Если же одна из транзакций откатится, то и транзакция из состояния Prepared также откатятся.

Отсюда возникают транзакции limbo. Если между серверами разровется соединение в тот момент, когда одна транзакция перешла в состояние Prepared и готова подтвердится, то сервер не может решить подтвердить или откать ее изменения этой транзакции, поэтому не следует использовать двухфазное подтверждение на медленных каналах связи.

Физическая структура базы данных INTERBASE

Сама БД Interbase представляет собой один (реже более) файл как правило с раширением .GDB, в котором хранится информация обо всем, что связано с этой БД: сами данные, индексы, триггеры, ХП и т.д., ислючая информацию о пользователях, которая хранится на уровне всего сервера в системной базе данных ISC4.GDB. Этот файл ограничен размером в 32 Гбайта, что вполне достаточно для большинства задач. При желании можно разбить БД на несколько файлов.

Фактически файл представляет набор страниц жестко определенного размера, поэтому размер файла всегда кратен размеру

страницы. Размер страницы задается только один раз при создании БД и не может быть изменен в течении ее жизненного цикла. Возможные размеры страниц 1, 2, 4, 8 и 16 (только для Interbase 6.5, Firebird 1.0) Кбайт.

Чтение и запись в БД осуществляются постранично, поэтому для оптимальной производительности нужно выбирать размер страницы равным (или кратным) размеру кластера. Многие характеристики БД и сервера исчисляются в страницах, например размер буфера (Datebase cache).

В зависимости от своего назначения страницы бывают 11 различных типов:

0 — неопределенный тип страницы, возможно страница свободна

1 — Database header page. Страница заголовка базы данных.

2 — Page Inventory Page (Space Inventory Page | SIP). Страница, хранящая информацию о распределении страниц

3 — Transaction Inventory Page | TIP. Страница учета транзакций

4 — Pointer page. Страница указателей

5 — Data page. Страница данных

6 — Index root page. Страница вершины индекса.

7 — Index (Btree) page. Страница индексов

8 — Blob data page. Страница для хранения BLOB данных.

9 — Genids. Страница генераторов.

10 — Write ahead log information. Не используется

Страницы не упорядочены. Они располагаются в той последовательности, как они создавались сервером по мере необходимости. Условно взаимосвязи таблиц можно представить так:

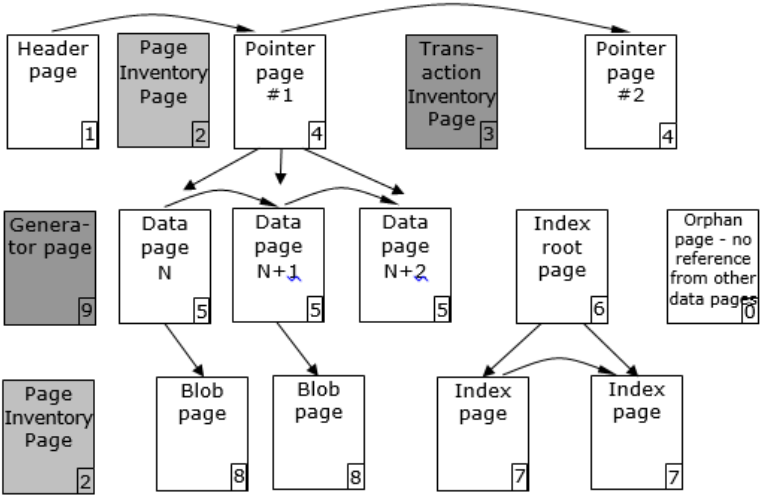


Рис. 11.1. Взаимосвязь таблиц

При регистрации БД сервер считывает первые 1024 байта (минимальный размер страницы) GDBфайла и определяет действительно ли это файл базы данных и читает размер страницы этой БД, а затем считывает уже страницу целиком и определяет параметры БД, которые хранятся на первой заголовочной странице: дата создания БД, диалект, размер буфера, режим чтения/записи, контрольная сумма, версия и т.д. На странице заголовка находится также ссылка на первую страницу указателей, к которой сервер и переходит. Страница указателей хранит в виде таблицы список номеров страниц данных (data page vector), по которому сервер переходит к первой странице данных и начинает построение внутреннего представления базы данных, которое в дальнейшем и используется им для всех операций с БД.

На странице учета транзакций содержится массив из двухбитовых значений, определяющих состояние транзакции: 00 — транзакция активна, 01 — транзакция завершилась commit, 10 — транзакция завершилась Roll Back, 11 — limbo транзакция, что позволяет одновременно выполняющимся транзакциям узнавать о состоянии друг друга.

После страницы заголовка, т.е. второй попорядку расположена страница учета страниц (PIP | SIP). Их может быть несколько и следуют они через равные промежутки, т.е. между ними располагается фиксированное число страниц других типов. На страницах учета страниц описывается состояние других страниц в БД, а сами они не учитываются на страницах указателей. Состояние страницы может быть одним из трех: не распределенное (not allocated), распределенное (allocated with space), распределенное и заполненное (allocated and full). Когда появляется необходимость в дополнительном пространстве для новых данных, сервер проверяет PIP на предмет наличия нераспределенных страниц, если такая находится, то сервере меняет ее состояние на распределенное, если таких страниц нет, то к БД дописавается новая страница данных.

Как только страница распределена, сервер записывает ее состояние на PIP, затем пишет саму страницу, а затем, чтобы присоединить ее, например, к множеству страниц данных, нужно на последней странице этого множества поместить ссылку на вновь созданную страницу. Если после записи на PIP сервер прервал работу, не успев присоединить страницу к какомулибо множеству. То такая страница становится потерянной, т.е. найти ее по ссылкам с других страниц сервер не сможет.

Заголовочная страница, страницы указателей, страницы учета транзакций и страницы учета страниц относятся к служебным типам страниц, которые используются только сервером и недоступным пользователям Intebase.

Страницы генераторов представляют собой массив 4байтных чисел, которые показываеют состояние генератора. Фактически генератор — это обычный счетчик. Имена генераторов хранятся не на этой странице, а в системной таблице rdb$generators.

Каждой таблице, вне зависимости от того имеет она индексы или нет, соответствует, по крайней мере, одна страница вершины индекса, которая содержит указатели на страницы индексов для соответствующей таблицы. Index root page играет для индексов примерно ту же роль, что и страница указателей для страниц данных.

Кроме ссылок на страницы индексов, она содержит системную информацию об индексах: поля, флаги, порядок и т.д.

Непосредственно индексы (Вдеревья) содержат индексные страницы (index pages).

Вся пользовательская информация хранится на страницах данных и страницах, содержащих BLOBзначения (BLOBpages). Если значение BLOBполя помещается на странице вместе со всей записью, то отдельной BLOBстраницы для него не создается. Помимо самих данных, страница содержит идентификатор таблицы, которой принадлежат данные, номер следующей страницы с данными и список записей виде: сдвиг записи (номер байта) и длина записи. Причем здесь же хранятся и версии записи, отмеченные с своими идентификаторами транзакций. Заполняется страница от конца к началу, т.е. для страницы размером 8 Кбайт = 8192 байт будем видеть примерно такой список: 8156 / 34, 8124 / 30, 8092 /30 и т.д.

Безопасность баз данных

Вопросы безопасности и целостности — еще одна из важнейших сторон работы СУБД. Под безопасностью понимают защиту БД от несанкционированного доступа, а также от искажения данных в результате различных сбоев. Систему можно считать безопасной только в том случае, если пользователю допускается выполнять только разрешенные действия. Целостность БД связана с корректным выполнением этих действий.

Остальные проблемы, касающиеся защиты баз данных совпадают с вопросами защиты компьютерных систем в целом: угрозы со стороны внешней среды (стихийные бедствия, радиация), кража оборудования, вирусы, неблагонадежность персонала и т.д. Эти моменты, поэтому, в вопросе защиты баз данных как таковых не рассматриваются.

В СУБД традиционно поддерживаются избирательный и обязательный подходы обеспечения безопасности данных.

При избирательном подходе к управлению безопасностью каждый пользователь обладает различными правами (полномочиями) при работе с тем или иным объектом БД.

В случае обязательного подхода каждому объекту БД присваивается уровень доступа, а пользователям — уровни допуска. Разумеется, для получения доступа к объекту пользователь должен обладать соответствующим уровнем допуска.

Оба подхода реализуются в СУБД в виде особых правил безопасности, предусматривающих опознание источника запроса. Существует три варианта идентификации пользователя:

1. чтолибо, что знает только сам пользователь;
2. чтото, чем он обладает (пластиковая карточка, ключ);
3. идентификация физических параметров (сетчатка глаза, отпечаток пальца, идентификация голоса и т.п.)

Самым дешевым и потому самым распространенным методом идентификации является первый. Что же может знать только пользователь:

1. Пароль. Преодолеть эту защиту можно перебором возможных значений, если знать длину пароля и алфавит.
2. Заранее введенные вопросы и ответы на них, задаваемые системой при подключении пользователя в произвольном порядке. Нр, когда у Вашей дочери день рожденья? Кем вам приходится Иван Петрович? Как девичья фамилия Вашей бабушки? Недостатком метода является его неоперативность.
3. Некоторый алгоритм преобразования над числом, которое случайно выдает компьютер при попытке входа. Нр, система при входе выдает число 345. Алгоритмом преобразования для конкретного пользователя являются следующие действия переставить цифры в числе и вычесть из результата 10. Таким образом, в данном случае пользователь должен ввести 533.

Полученное в результате преобразования число должно совпасть в результатом вычислений компьютера. Если попытка входа отклоняется, то число уже не повторяется и поэтому подсматривать то, что вводит пользователь не имеет смысла.

Иногда делают ограниченное число попыток входа, чтобы нельзя было подобрать правильное значение.

Итак, личность пользователя обязательно устанавливается!

В основу Interbase положен именно избирательный подход. На сервере существует список пользователей, каждому из которых можно назначить различные права для различных объектов СУБД. Фактически пользователь – это регистрационная запись, состоящая из имени пользователя и его пароля. По одной р/з может работать несколько реальных людей.

Существует существенное отличие Interbase от других СУБД. Регистрационные записи хранятся не в самой базе данных, а на сервере, в служебной базе данных ISC4.gdb. Это значит, что внутри БД объекты никак не шифруются и не защищаются. Для каждого объекта в БД существует список разрешенных операций для конкретного пользователя. Сервер разрешает доступ к объекту путем сравнения прав на него с правами пользователя, запустившего транзакцию.

Т.е. на СЕРВЕРЕ_1 существует БД и регистрационная запись АБД — sysdba. Пользователь не знает пароля этой РЗ, но имеет доступ к файлу БД (gdbфайлу). Подключив эту БД на СЕРВЕРЕ_2, где ему известен пароль регистрационной записи sysdba, этот пользователь может получить доступ ко всем объектам в соответствии с р/з АБД.

Причиной такого подхода является особый подход к безопасности файлов БД, который нужно обеспечивать на уровнях операционной системы, системного администратора и службы безопасности.

На уровне ОС каталог с файлом БД не должен быть доступен в сети.

На уровне СА доступ к компьютеру должен быть только для зарегистрированных пользователей.

На уровне СБ не должно быть проникновения в помещение посторонних.

Кроме того, на сервере для всех БД используется единое пространство имен пользователей, что часто упрощает настройку и администрирование. Изначально на сервере существует пользователь SYSDBA — системный администратор, с паролем masterkey. Рекомендуется этот пароль сразу же сменить. Добавлять новых пользователей может только SYSDBA.

Кроме пользователей существуют т.н. роли. Они служат для объеди¬нения пользователей с одинаковыми правами. Это дает возможность:

– при установке прав нового пользователя достаточно только указать к какой группе он будет принадлежать;
– при изменении прав изменить права группе, а не каждому пользователю;
– права пользователей хранятся в системной таблице, размер которой может оказаться достаточно большим, если заносить в нее права каждого пользователя, что скажется на скорости выполнения запросов, т.к. для каждого запроса сервер проверяет права пользователя, выполнившего запрос.

Права — это разрешение пользователю, процедуре или триггеру совершить какуюлибо операцию над объектом базы данных. Существуют следующие права:

– Для таблиц и их полей — права на выполнение операций SELECT, INSERT, UPDATE, DELETE и REFERENCES (это право дает пользователю возможность создавать ограничения внешнего ключа FOREIGN KEY на данную таблицу)
– Для просмотров VIEW и их полей — права на выполнение операций S/I/U/D. Т.к. просмотр формируется динамически по запросу, то права на изменение данных (I/U/D) фактически относятся к таблице, на которой он основан.
– Права на выполнение хранимых процедур — EXECUTE.

Т.о. объектом в данном контексте считаются таблицы, поля, просмотры и хранимые процедуры.

Помимо прав на объекты, которые выдаются пользователям, права могут иметь также и ХП/триггеры. Если для них специально не указаны права, нр, на вставку записей в таблицу, то процедура руководствуется правами пользователя ее запустившего. Если же для процедуры указаны права, нр разрешена вставка записей в таблицу, то пользователь, запустивший процедуру и не обладающий правом вставки сможет успешно добавить запись.

Раздача прав осуществляется при помощи оператора GRANT со множеством опций.

Например,

GRANT Select ON Table1 TO User1

дает пользователю User1 право чтения таблицы Table1.

GRANT Select, Insert ON Table1 TO User1

дает пользователю User1 возможность выборки и вставки записей таблицы Table1.

GRANT All ON Table1 TO User1, User2

дает пользователям User1 и User2 все возможные права на Table1.

GRANT Select ON Table1 TO User1, User2

дает ВСЕМ пользователям право чтения Table1.

GRANT EXECUTE ON PROCEDURE Proc1 TO User2

дает пользователю User2 право запускать хранимую процедуру Proc1.

Как говорилось пользователей можно объединять в группы (роли) при наличии у них одинаковых прав с целью уменьшения количества выдаваемых разрешений. Для этого нужно создать роль:

CREATE ROLE Reader;

Чтобы выдать этой роли какиелибо права используется тот же GRANT, где в предложении TO указывается имя роли:

GRANT Select ON Table1 To Reader;

Чтобы присвоить пользователю роль:

GRANT Reader TO User3;

Теперь пользователь может указывать эту роль при подключении к БД. Т.е. пользователь может иметь членство в различных ролях и какие права у него будут при данном соединении зависит от того, какую роль он указал при подключении.

Права может раздавать не только АБД, но также и владелец объекта, т.е. пользователь, который создал объект. Кроме того, права может выдавать пользователь, который получил их от владельца или администратора с параметром WITH GRANT OPTION. Нр,

GRANT Select ON Table1 TO User1 WITH GRANT OPTION

дает пользователю User1 право чтения таблицы Table1 с возможностью передачи этого права (только SELECT) другим пользователям.

Права можно и забирать у пользователя оператором REVOKE с тем же самым синтаксисом, что и GRANT, за исключением того, что права не выдаются (TO User1), а забираются (FROM User1). Нр:

REVOKE All ON Table1 FROM User1;

При этом если User1, получивший право Select с возможностью его дальнейшей передачи (WITH GRANT OPTION), передал это право User2, User2 также потеряет его. Т.е. отмена прав носит каскадный характер.

Пользователь SYSDBA автоматически обладает всеми возможными правами на все объекты, и их список нельзя ни ограничить, ни расширить.

Системная таблица, а которой хранятся права пользователей называется rdb$user_privileges.

Избирательное управление доступом задается правилами. которые должны включать следующее:

- имя правила — представляет собой структуру, по которой это правило идентифицируется системой;
- собственно правила или привилегии — набор директив, составляющих способ и возможность доступа, модификаций и т. п. объектов БД;
- диапазон применения привилегий;
- идентификаторы пользователей, обладающих вышеперечисленными привилегиями;
- действие при нарушении правила — здесь указывается поведение системы в случае, если пользователь нарушил правило безопасности. Как правило, оно заключается в отказе от выполнения запрашиваемого действия.

Для примера рассмотрим правило безопасности для отношения MOVIE.

СОЗДАТЬ ПРАВИЛО БЕЗОПАСНОСТИ RULE1

ДЛЯ МОДИФИКАЦИИ И УДАЛЕНИЯ MOVIE (ID_M, TITLE, YEAR, LEN, KIND)

ДЛЯ KIND = "КОМЕДИЯ"

ПОЛЬЗОВАТЕЛИ: Ivan, Denis

НЕ ВЫПОЛНЯТЬ ПРИ НАРУШЕНИИ ПРАВИЛА

Приведенное выше правило содержит все пять оговоренных элементов. Фактически создано правило безопасности с именем RULE1, позволяющее модификацию и удаление кортежей (ID_M, TITLE, YEAR, LEN, KIND) отношения MOVIE. относящихся к жанру «комедия» для пользователей с идентификаторами Ivan и DENIS. Если

правило будет нарушено (например, запрашивается оговариваемое действие со стороны иного пользователя, то в запрашиваемом действии будет отказано.

Обязательное управление доступом к БД реализуется при выполнении следующих правил:

- пользователь имеет возможность работы (но не модификации) с объектом, если уровень его допуска больше или равен уровню доступа объекта;
- пользователь имеет возможность модифицировать объект, если уровень его допуска равен уровню доступа объекта.

Правило безопасности в этом случае для пользователя о идентификатором DENIS можно, например, сформулировать следующим образом:

СОЗДАТЬ ПРАВИЛО БЕЗОПАСНОСТИ RULE2

ДЛЯ ПОЛЬЗОВАТЕЛЯ Denis

УСТАНОВИТЬ УРОВЕНЬ ДОПУСКА =5

А для отношения MOVIE, например, вот так:

СОЗДАТЬ ПРАВИЛО БЕЗОПАСНОСТИ RULE3

ДЛЯ МОДИФИКАЦИИ И УДАЛЕНИЯ MOVIE (ID_M, TITLE, YEAR, LEN, KIND)

УСТАНОВИТЬ УРОВЕНЬ ДОПУСКА 5

НЕ ВЫПОЛНЯТЬ ПРИ НАРУШЕНИИ ПРАВИЛА

Тогда пользователь DENIS имеет доступ для модификации и удаления кортежей отношения MOVIE. поскольку уровень его допуска и уровень доступа к отношению соответствуют друг другу.

Язык SQL включает операторы GRANT и REVOKE, предназначенные для организации защиты таблиц в базе данных. Применяе¬мый механизм защиты построен на использовании идентификаторов пользователей, предоставляемых им прав владения и привилегий.

Каждый созданный в среде SQL объект имеет своего владельца. Владелец задается идентификатором пользователя, определенным в предложении AUTHORIZATION той схемы, которой этот объект принадлежит. Исходно владелец объекта является единственной

персоной, которая знает о существовании данного объекта и имеет право выполнять с этим объектом любые операции.

Оператор GRANT используется для предоставления указанным пользователям при¬вилегий в отношении поименованных объектов базы данных. Этот оператор обычно используется владельцем таблицы с целью предоставления доступа к ней другим пользователям. Оператор GRANT имеет следующий формат:

GRANT{privilege_list | ALL PRIVILEGES}
ON object_name
TO {authorization_list | PUBLIC}
[WITH GRANT OPTION]

Параметр privilege_list представляет собой список, состоящий из одной или бо¬лее привилегий, разделенных запятыми:

SELECT
DELETE
INSERT [{column_name [, ...])]
UPDATE [{coliimn_name [,...])]
REFERENCES [{column_name [, ...])]
USAGE

Привилегиями называют действия, которые пользователь имеет право выполнять в отношении данной таблицы базы данных или представления. В стандарте ISO определяется следующий набор привилегий:

SELECT — право выбирать данные из таблицы;

INSERT — право вставлять в таблицу новые строки;

UPDATE — право изменять данные в таблице;

DELETE — право удалять строки из таблицы;

REFERENCES — право ссылаться на столбцы указанной таблицы в описаниях требований поддержки целостности данных;

USAGE — право использовать домены, проверки, наборы символов и трансляции.

Привилегии INSERT и UPDATE могут ограничиваться лишь отдельными столбцами; таблицы, в этом случае пользователь может модифицировать значения указанных столбцов, но не может изменять значения остальных столбцов таблицы. Аналогичным образом,

привилегия REFERENCES может распространяться только на отдельные столбцы таблицы, что позволит использовать их имена в формулировках требований защиты целостности данных — например, во фразах CHECK и FOREIGN KEY, входящих в определения других таблиц, — тогда как применение для подобных целей остальных столбцов будет запрещено.

Когда пользователь с помощью оператора CREATE TABLE создает новую таблицу, он автоматически становится ее владельцем и получает по отношению к ней полный набор привилегий. Остальные пользователи исходно не имеют какихлибо привилегий в отношении вновь созданной таблицы. Чтобы обеспечить им доступ к ней, владелец должен явным образом предоставить им необходимые права, для чего используется оператор GRANT.

Когда пользователь создает представление с помощью оператора CREATE VIEW, он автоматически становится владельцем этого представления, однако совсем необязательно получает по отношению к нему полный набор прав. Для создания представления пользователю достаточно иметь привилегию SELECT для всех входящих в данное представление таблиц и привилегию REFERENCES для всех столбцов, упоминаемых в определении этого представления. Привилегии INSERT, UPDATE и DELETE в отношении созданного представления пользователь получит только в том случае, если он имеет соответствующие привилегии в отношении всех используемых в представлении таблиц.

Кроме того, из соображений упрощения, в операторе GRANT можно указать ключе¬вое слово ALL PRIVILEGES, что позволит предоставить указанному пользователю все шесть существующих привилегий без необходимости их перечисления. Кроме того, в этом операторе может указываться ключевое слово PUBLIC, означающее предоставление доступа указанного типа не только всем существующим пользователям, но также и всем тем пользователям, которые будут определены в базе данных впоследствии. Параметр object_name может представлять собой имя таблицы базы данных, представления, домена, набора символов, проверки или трансляции.

Фраза WITH GRANT OPTION позволяет всем указанным в списке параметра authorization_id_list пользователям передавать другим пользователям все предоставленные им в отношении указанного объекта привилегии. Если эти пользователи также передадут соб¬ственные полномочия другим пользователям с указанием фразы WITH GRANT OPTION, то последние, в свою очередь, также получат право передавать свои полномочия другим пользователям. Если эта фраза не будет указана, получатель привилегии не сможет передать свои права другим пользователям. Таким образом, владелец объекта может четко контролировать, кто получил право доступа к объекту и какие полномочия ему предоставлены.

В языке SQL для отмены предоставленных пользователям посредством оператора GRANT привилегий используется оператор REVOKE. С помощью этого оператора могут быть отменены все или некоторые из привилегий, предоставленных указанному пользователю раньше. Оператор REVOKE имеет следующий формат:

REVOKE [GRANT OPTION FOR] {privilege_list | ALL PRIVILEGES}

ON ob]ect_name

FROM {authorization_id_list | PUBLIC} [RESTRICT | CASCADE]

Ключевое слово ALL PRIVILEGES означает, что для указанного пользователя отменяются все привилегии, предоставленные ему ранее тем пользователем, который ввел данный оператор. Необязательная фраза GRANT OPTION FOR позволяет для всех привилегий, переданных в исходном операторе GRANT фразы WITH GRANT OPTION, отменять возможность их передачи независимо от самих этих привилегий. Назначение ключевых слов RESTRICT и CASCADE аналогично тому, которое они имеют в операторе DROP TABLE.

Поскольку наличие привилегий необходимо для создания определенных объектов, удаление привилегии может удалить право, за счет использования которого был создан тот или иной объект (подобные объекты принято называть брошенными). Выполнение оператора REVOKE будет отменено, если в его результате могут появиться брошенные объекты (например, представления), если только в нем не указывается ключевое слово CASCADE. Если ключевое слово

CASCADE в операторе присутствует, то для любых брошенных объектов (представлений, доменов, ограничений или проверок), возникающих при выполнении исходного оператора REVOKE, будут автоматически выданы операторы DROP.

Привилегии, которые были предоставлены указанному пользователю другими пользователями, не могут быть затронуты данным оператором REVOKE. Следовательно, если другой пользователь также предоставил данному пользователю удаляемую привилегию, то право доступа к соответствующей таблице у указанного пользователя сохранится.

Для обеспечения высокого уровня безопасности СУБД ведут журнал выполняемых операций. По этому журналу — с одной стороны можно осуществить восстановление данных, с другой — выявить когда, каким образом и кем были осуществлены несанкционированные действия над БД.

Обычно в файле журнала хранится следующая информация:

- исходный текст запроса;
- имя удаленного терминала, откуда был подан запрос;
- идентификатор пользователя, подавшего запрос;
- дата и время осуществления запроса:
- используемые запросом отношения, кортежи и атрибуты;
- значения данных, с которыми работали до их модификации;
- значения данных, с которыми осуществлялась работа после их модификации.
- Помимо вышесказанного, довольно часто для защиты данных используется хранение и передача шифрованных данных. Открытый (незашифрованный) текст шифруется при помощи специальных алгоритмов.

Другой стороной проблемы безопасности и целостности БД является точность и корректность хранимых в ней данных. Обычно этот вопрос решают с помощью ограничений целостности.

Традиционно различают два вида ограничений целостности: немедленно проверяемые и откладываемые. К немедленно проверяемым ограничениям целостности относятся такие ограничения, проверку которых не имеет смысла откладывать на более поздний период. Примером такого ограничения, проверку которого

откладывать бессмысленно, являются ограничения домена — длина фильма не может быть меньше 30 минут, или доход продюсера не может быть отрицательным. 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 БД, а не на какиелибо отдельные операции. Обычно такие ограничения проверяются в конце транзакции, и их нарушение вызывает завершение транзакции оператором ROLLBACK. В некоторых СУБД поддерживается специальный оператор 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 БД внутри транзакции.

При соблюдении обязательного требования поддержания целостности БД возможны следующие уровни изолированности транзакций:

- первый уровень — отсутствие потерянных изменений. Рассмотрим следующий пример совместного выполнения двух транзакций. Транзакция 1 изменяет объект базы данных S. До завершения транзакции 1 транзакция 2 также изменяет объект S. Транзакция 2 завершается оператором ROLLBACK, например, по причине нарушения ограничений целостности. Тогда при повторном чтении объекта S транзакция 1 не видит изменений этого объекта. произведенных ранее. Возникает ситуация потерянных изменений. Понятно, она противоречит требованию изолированности пользователей. Чтобы избежать такой ситуации в транзакции 1 требуется, чтобы до завершения транзакции 1 никакая другая транзакция не могла изменять объект S;
- второй уровень — отсутствие чтения данных, модифицируемых другой транзакцией. Рассмотрим такой пример совместного выполнения транзакций 1 и 2. Транзакция 1 изменяет объект базы данных S. Параллельно с этим транзакция 2 читает объект S. Поскольку операция изменения еще не завершена, транзакция 2

видит несогласованные данные. В частности, операция транзакции 1 может быть отвернута при контроле немедленно проверяемого ограничения целостности. Это тоже не соответствует требованию изолированности пользователей (каждый пользователь начинает свою транзакцию при согласованном состоянии БД и вправе ожидать видеть согласованные данные). Чтобы избежать ситуации чтения таких данных, до завершения транзакции 1, изменившей объект S, никакая другая транзакция не должна читать объект S;

– третий уровень — отсутствие неповторяющихся чтений. Рассмотрим следующий сценарий. Транзакция 1 читает объект базы данных S. До завершения транзакции 1 транзакция 2 изменяет объект S и успешно завершается оператором COMMIT. Транзакция 1 повторно читает объект S и видит его измененное состояние. Чтобы избежать неповторяющихся чтений, до завершения транзакции 1 никакая другая транзакция не должна изменять объект S.

Как уже было сказано выше. существует возможность обеспечения разных уровней изолированности для той или иной транзакции путем введения блокировки. Кроме того, может использоваться сериализация транзакций. Способ выполнения набора транзакций называется сериальным, если результат совместного выполнения транзакций эквивалентен результату некоторого последовательного выполнения этих же транзакций.

Сериализация транзакций — это такой механизм их выполнения по некоторому сериальному плану, который обеспечивается на уровне основных функций СУБД ответственных за управление транзакциями. Система, в которой поддерживается сериализация транзакций, обеспечивает реальную изолированность пользователей.

Основная проблема в реализации состоит в выборе метода сериализации набора транзакций, который не слишком ограничивал бы их параллельность. Простейшим 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 БД.

Обычно ограничения целостности применяют для описания базовых отношений — последние содержат данные, отражающие реальную действительность, поэтому их обрабатывают таким образом, чтобы данные были корректными.

В общем случае ограничение целостности должно содержать три основные части:

- имя ограничения — представляет собой структуру, по которой это ограничение идентифицируется системой;
- собственно ограничения — набор директив и команд, составляющих способ и возможность контроля, и представляющий в конечном итоге логическое выражение Ограничение удовлетворяется, если оно истинно, и нарушается — если оно ложно;
- действие при нарушении ограничения — здесь предписывается действие системы при нарушении ограничения.

Например, ограничение целостности для отношения MOVIE можно сформулировать следующим образом:

СОЗДАТЬ ОГРАНИЧЕНИЕ ЦЕЛОСТНОСТИ RULE 4
ДЛЯ ВСЕХ MOVIE (MOVIE.LEN>30 И MOVIE.LEN<600)
НЕ ВЫПОЛНЯТЬ ПРИ НАРУШЕНИИ ПРАВИЛА

В данном примере ограничение целостности накладывается на атрибут LEN таким образом, что игнорируются все попытки установить длину менее 30 и более 600. Точнее говоря, в ограничении оговорен допустимый интервал длин фильмов (больше 30 и меньше 600).

При создании ограничения целостности система сначала проверяет, удовлетворяет ли текущее состояние БД новому ограничению. Если это условие не выполняется, то создаваемое ограничение может быть отвергнуто. В противном случае оно принимается и в дальнейшем используется системой.

Различают четыре типа ограничений целостности:

- ограничение целостности домена — им определяется множество значений, из которых состоит домен. Особенности ограничения такого рода заключаются в том. что его имя должно совпадать с именем домена. Кроме того, поскольку домены сами по себе не обновляются, то отпадает необходимость предусматривать

реакцию на нарушение ограничения. Более того. эти ограничения можно устранить только за счет устранения самого домена;

- ограничение целостности атрибута — это фактически определение домена, из которого берутся значения для данного атрибута. Имя такого ограничения должно совпадать с именем соответствующего ограничения домена, то есть с именем домена Проверка осуществляется немедленно, и попытка выполнить действие, нарушающее ограничение, будет отвергнута сразу же. Наконец, ограничения целостности атрибута снимаются только с помощью устранения самого атрибута;
- ограничение целостности отношения — правило, задаваемое только для данного отношения БД. Ограничение целостности отношения всегда проверяется немедленно, то есть при любой попытке модификации отношения осуществляется контроль всех заданных условий. Действие такого ограничения происходит в том случае, когда ограничение (заданное логическим выражением) становится ложным;
- ограничение целостности БД — задается для двух или более связанных между собой отношений. В отличие от других ограничений, эта их разновидность помимо традиционных частей обязательно должно содержать, по крайней мере, одно условие соединения отношений: обычно это условие содержит две связанные переменные, определенные в двух разных отношениях. Следовательно, ограничение приводит к тому. что отношения будут связаны между собой. Ограничения целостности БД не проверяются немедленно, а их выполнение откладывается до конца выполнения транзакции В качестве действия на нарушение условия целостности в подавляющем большинстве случаев используется завершение транзакции оператором ROLLBACK.

Разновидностью традиционных ограничений целостности являются ограничения состояния и перехода. Смысл их заключается в том. что достаточно часто возникает потребность рассмотреть не только область допустимых значений кортежа, но и переход значений из одного состояния в другое. Рассмотрим пример такого ограничения для модифицированного отношения S (СТУДЕНТЫ), куда добавлен атрибут KURS — курс, на котором учится студент:

СОЗДАТЬ ОГРАНИЧЕНИЕ ЦЕЛОСТНОСТИ RULES
ДЛЯ ВСЕХ S(S.KURS >= S'.KURS)
НЕ ВЫПОЛНЯТЬ ПРИ НАРУШЕНИИ ПРАВИЛА

Такое ограничение накладывается на атрибут соответственно S'.KURS — до и S.KURS — после выполнения обновления. Приведенный пример ограничивает курс, на котором учится студент так. что его можно изменить либо в большую сторону, либо оставить без изменений. Действительно — ведь курс не должен уменьшаться. Ограничения состояния и перехода используют только для отношения или БД.

Таким образом, рассмотренные вопросы безопасности и целостности как БД в целом, так и ее элементов позволяют сформулировать и определить соответствующие ограничения целостности, что, в свою очередь, помогает решить проблему несовместимых данных.

Лекция 12. ТЕХНОЛОГИЯ ФИЗИЧЕСКОГО ХРАНЕНИЯ И ДОСТУПА К ДАННЫМ

Индексирование

Реляционные базы данных, хранят записи в таблицах в неупорядоченном виде, т. е. совершенно не заботятся о том, как физически располагаются записи в таблице. Неупорядоченность хранения означает, что две записи, добавляемые в таблицу одна за другой, совсем не обязательно окажутся "рядом". Более того, данные, извлекаемые из таблицы, также не имеют какоголибо порядка, кроме того, который явно должен быть указан пользователем, составляющим запрос на выборку.

Однако конечные пользователи приложений хотят видеть свои данные в определенном порядке (нр, фамилии людей по алфавиту). Задачу представления данных в упорядоченном виде решают индексы. Значения полей, входящих в индекс упорядочены и представлены в особом виде, оптимизированном для поиска нужных значений (а именно это и нужно для построения упорядоченных последовательностей). Отделение хранения данных от их представления дает дополнительные преимущества по сравнению с непосредственной сортировкой — исходную таблицу может потребоваться отсортировать поразному. Тогда и используются индексы — их может быть до 64 на каждую таблицу!

Если говорить о реализации индексов на физическом уровне, то они представляют двоичное дерево, узлы которого представляют собой пары {"значение поля в индексе" — "расположение данных в таблице"}. Поиск нужной записи в индексе идет с помощью механизма хешпоиска — одного из самых быстрых алгоритмов поиска.

Индексы — это структура данных, которая позволяет СУБД быстрее обнаруживать отдельные записи в файле, и, как следствие, сократить время выполнения запросов.

Индекс в базе данных аналогичен предметному указателю, приведенному в конце книги. Это структура, связанная с файлом и предназначенная для поиска информа¬ции по тому же принципу, что и предметный указатель в книге. Индекс позволяет избежать проведения

последовательного или пошагового сканирования файла в по¬исках нужных данных. При использовании индексов в базе данных искомым объектом может быть одна или несколько записей файла. Как и предметный указатель данной книги, индекс базы данных упорядочен, и каждый элемент индекса содер¬жит название искомого объекта, а также один или несколько указателей (идентификаторов записей) на место его расположения.

Структура индекса связана с определенным ключом поиска и содержит записи, состоящие из ключевого значения и адреса логической записи в файле, содержащей это ключевое значение. Файл, содержащий логические записи, назы¬вается файлом данных, а файл, содержащий индексные записи, — индексным файлом. Значения в индексном файле упорядочены по полю индексирования, которое обычно строится на базе одного атрибута. Если файл данных имеет по¬следовательное упорядочение, а поле индексирования является ключевым полем этого файла, т.е. гарантированно содержит уникальные значения для всех запи¬сей в файле, то такой индекс называется первичным индексом. Если поле индексирования (являющееся одновременно полем упорядочивания) не является ключевым полем файла и по этой причине одному индексному значению может соответствовать несколько записей файла, то такой индекс называется индексом кластеризации. Индекс, который определен на поле файла, отличном от поля его упорядочивания, называется вторичным индексом. Файл может иметь, по край¬ней мере, один первичный индекс или один индекс кластеризации, а также (дополнительно) несколько вторичных индексов. Индекс может быть разреженным (sparse) или плотным (dense): разреженный индекс содержит индексные записи только для некоторых ключевых значений поиска в данном файле, а плотный индекс имеет индексные записи для всех ключевых значений поиска в данном файле.

Индексно-последовательные файлы

Отсортированный файл данных с первичным индексом называется индексирован¬ным последовательным файлом, или индекснопоследовательным файлом. Эта структура является

компромиссом между файлами с последовательной и произволь¬ной организацией. В таком файле записи могут обрабатываться как последовательно, так и выборочно, с произвольным доступом, осуществляемым на основе поиска по заданному ключевому значению с использованием индекса. В случае роста или сокращения размера файла этот процесс управляется динамически, без необходимости периодического выполнения реорганизации.

Обычно большая часть первичного индекса может храниться в первичной памяти, что позволяет обрабатывать его существенно быстрее. Для ускорения поиска могут применяться специальные методы доступа, например, метод бинарного поиска (золотого сечения).

Операции вставки и удаления записей в упорядоченном файле усложняются необходимостью поддерживать установленный порядок записей. Для вставки новой записи нужно найти ее месторасположение в заданном упорядочении, а потом найти пространство для вставки. Если на нужной странице достаточно места для размеще¬ния новой записи, то тогда потребуется переупорядочить лишь только эту страницу, после чего вывести ее на диск. Если же свободного места недостаточно, то в таком случае потребуется переместить одну или несколько записей в следующую страницу. Аналогично, в следующей странице также может не оказаться достаточно свободного места, и из нее также потребуется переместить некоторые записи в следующую стра¬ницу и т.д. Таким образом, вставка записи в начало большого файла может оказать¬ся очень длительной процедурой. Для решения этой проблемы часто используется временный неотсортированный файл, который называется файлом переполнения (overflow file) или файлом транзакции (transaction file). При этом все операции вставки выполняются в файле переполнения, содержимое которого периодически сливается с основным отсортированным файлом. Таким образом, операции вставки выполняются более эффективно, но при этом некоторое отрицательное влияние ока¬зывается на выполнение операций извлечения данных. Если запись не может быть найдена во время бинарного поиска в отсортированном файле, то в таком случае придется выполнить линейный поиск в файле переполнения. И,

наоборот, при удалении записи их потребуется реорганизовать так, чтобы удалить пустующие места.

Упорядоченные последовательные файлы довольно редко используются для хра¬нения информации баз данных, за исключением тех случаев, когда для файла орга¬низуется первичный индекс. Основным методом индексации является использование многоуровневых индексов — Вдеревьев.

Вторичные индексы

Вторичный индекс также является отсортированным файлом, аналогичным первичному индексу. Однако связанный с первичным индексом файл данных всегда отсортирован по ключу этого индекса, тогда как файл данных, связанный со вторичным индексом, необязательно должен быть отсортирован по индексному ключу. Кроме того, ключ вторичного индекса может содержать повторяющиеся значения, что не допускается для значений ключа первичного индекса. Для работы с такими повторяющимися значениями ключа вторичного индекса обычно используются перечисленные ниже методы.

- Создание плотного вторичного индекса, который соответствует всем записям файла данных, но при этом в нем допускается наличие дубликатов.
- Создание вторичного индекса со значениями для всех уникальных значений ключа. При этом указатели являются многозначными, т.е могут указывать на несколько физических записей.

Вторичные индексы повышают производительность обработки запросов, в которых в качестве аргументов поиска используются обычные атрибуты, а не значения первичного ключа. Однако такое повышение производительности запросов должно быть сбалансировано с объемом дополнительной обработки, необходимой для поддержания индексов при обновлении информации в базе данных.

Индекс в виде сбалансированного B+ дерева

B+ дерево обеспечивает многоуровневое индексирование, эффективное как для прямой, так и для последовательной обработки. Дерево состоит из иерархически организованных индексных записей (ИЗ) и файла записей данных (рис. 12.1).

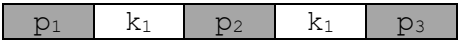


Рис. 12.1. Типичная индексная запись B+ дерева (n=3)

ИЗ B+ дерева содержит n указателей p и (n–1) ключей k. Число n определяется создателем дерева.

ИЗ верхнего уровня называется корнем, нижнего — листом. Дерево является сбалансированным — все листья находятся на одинаковом расстоянии от корня, что гарантирует доступ ко всем записям БД с одинаковой быстротой. Расстояние определятся в записях от корня до листа. Последовательность ИЗ, которые нужно просмотреть, чтобы достигнуть искомой называется путем. Чем больше n, тем меньше путь, тем быстрее доступ к искомым записям.

B+ дерево должно удовлетворять следующим условиям:

1. Каждая не листовая ИЗ содержит от n/2 до n указателей на ИЗ следующего уровня. n/2 округляется в большую сторону.
2. Листовая ИЗ содержит от (n–1)/2 до (n–1) указателей на записи файла.
3. Ключи в ИЗ упорядочены по значению k1 < k2 < ... < kn–1.
4. Все ключи в поддереве, на которое указывает p1 строго меньше k1.

Все ключи в поддереве, на которое указывает pi имеют значения больше либо равные ki–1 и строго меньшие ki.

Все ключи в поддереве, на которое указывает pn имеют значения больше либо равные kn–1.

Другими словами, для значения ключа в ИЗ, по указателю слева — поддерево со строго меньшими значениями, по указателю справа — с больше, либо равными.

5. Указатель pn листовой записи указывает на следующую листовую запись, так что при желании индекс можно просматривать последовательно.

Алгоритм создания B+ дерева (n=3):

1. Каждая не листовая ИЗ содержит от 2 до 3 указателей на ИЗ следующего уровня.
2. Обход дерева осуществляется следующим образом:

если искомая ИЗ < k1, то по указателю p1;

если k1 <= ИЗ < k2, то по p2;

если ИЗ >= k2, то по p3.

3. Добавление записи:

3.1. Спускаемся до листовой ИЗ согласно правилам обхода дерева.

3.2. Если лист, заполнен, то добавляем новый узел,

3.3. сортируем три записи в двух узлах,

3.4. добавляем индексирующую запись более высокого уровня (2 указателя),

3.5. поднимаемся на уровень выше,

3.6. повторяем с п. 3.2. с учетом ссылок на нижний уровень.

Пример создания B+ дерева (n=3): 31, 3, 18, 35, 29, 1, 33

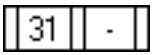


Рис. 12.2. Пример создания B+ дерева, шаг 1

Запись является и корневой, и листовой. Содержит 1 указатель (p1) на страницу с записью первичный ключ которой равен 31, что не противоречит правилу 2 (листовая запись содержит от (n–1)/2=1 до n–1=2 указателей на нижний уровень).

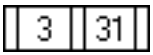


Рис. 12.3. Пример создания B+ дерева, шаг 2

Согласно правилу 3 значения в ИЗ д.б. упорядочены. Содержит два указателя (p1, p2) на страницы с записями.

Рис. 12.3. Пример создания B+ дерева, шаг 2

Согласно правилу 2 листовая ИЗ не может содержать более n–1=2 указателей, т.е. в одной ИЗ позволены только 2 ключа, поэтому ИЗ развивается на две, значения ключей упорядочиваются и первые n/2=2 ключа (3, 18) помещаются в левую ИЗ, остальные (31) — в правую.

Затем, необходимо проиндексировать полученные две ИЗ при помощи ИЗ верхнего уровня, согласно правилу 4.

Указатель p3 в левой ИЗ по правилу 5 указывает на правую ИЗ.

Рис. 12.4. Пример создания B+ дерева, шаг 3

Начинаем обход дерева с корня. 35 > 31, поэтому двигаемся по указателю p2. Первая листовая ИЗ свободна для добавления. Ключи в ИЗ упорядочены.

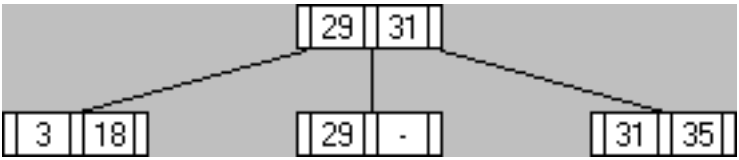


Рис. 12.5. Пример создания B+ дерева, шаг 4

Начинаем обход дерева с корня. 29 < 31, поэтому двигаемся по указателю p1. Листовая ИЗ (3, 18) заполнена, поэтому разбиваем ее на две, сортируем ключи и индексируем записью верхнего уровня (29, –),

но поскольку на этом уровне уже имеется ИЗ (31, –) и она не заполнена, то добавляем к ней 29 и сортируем.

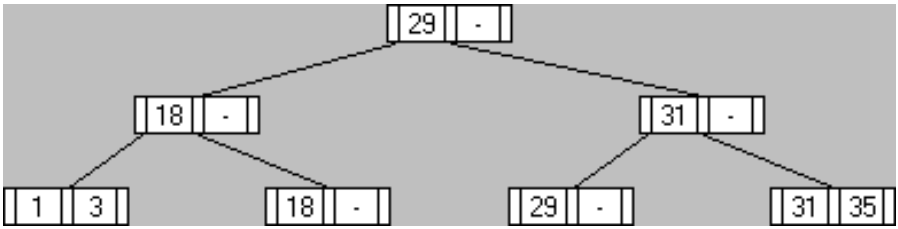


Рис. 12.6. Пример создания B+ дерева, шаг 5

Запись (3, 18), куда подошел бы ключ 1 заполнена. В результате разбиения, сортировки и индексации имеем два листа (1, 3), (18, –) и ИЗ верхнего уровня (18, –), которую нужно объединить с имеющейся (29, 31), что напрямую сделать невозможно, поэтому повторяем узел (29, 31) необходимо разбить, поместив по два указателя в каждую ИЗ (т.к. поместить 3 указателя в запись (18, 29) можно, а один указатель в (31, –) — нельзя), и индексируем их записью (29, –).

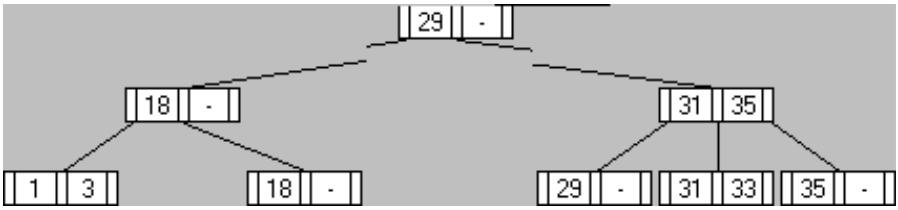


Рис. 12.7. Пример создания B+ дерева, шаг 6

33 > 29, двигаемся по p2, 33 > 31 — по p2, добавляем к (31, 35), упорядочиваем (31, 33), (35, –), индексируем: 35 добавляем к (31, –).

Пример создания B+ дерева (n=4): 14, 56, 95, 22, 64, 29, 35, 43, 25

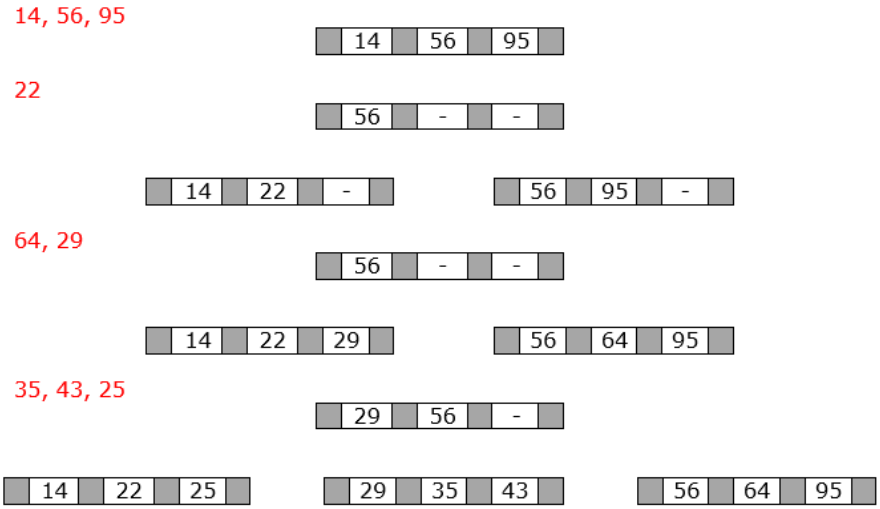


Рис. 12.8. Пример создания B+ дерева

Хэширование

Идея технологии хэширования, так же как и индексирования, состоит в быстром поиске записи. При всех достоинствах индексов очевидны два недостатка: поддержание самого индекса и то, что доступ к записи осуществляется минимум за два этапа: чтение индексной страницы и чтение страницы данных, содержащей запись.

Принцип хэширования основан на вычислении страницы с искомой записью по значению поиска. Поле, по которому возможно вычислить местонахождение записи называется хэшполем.

Простейший случай: запись занимает ровно одну страницу и записи только добавляются, тогда значение ключевого поля и будет номером страницы, на которой расположена запись. Абстрактность этого примера очевидна (переменный размер записей, возможность удаления, не суррогатный характер хэшполя), но идея в том, что и для первоначально размещения записи и для ее последующего поиска нужно вычислить номер ее страницы.

Усложняем пример: на одной странице может разместиться 20 записей, а всего их 500. Нужно 25 страниц. Ключевое поле имеет значения 0..499. Тогда на первой странице расположены записи с

ключами 0..19, на второй 20..39 и т.д. Чтобы вычислить страницу для записи с ключом, нр 345 нужно нацело разделить его на 20. Получим страницу номер 17.

Алгоритм свертки ключа в номер страницы носит название хэшфункции. В первом случае хэшфункция равна значению ключа.

Далее. Запись может иметь переменный размер. В этом случае не гарантируется равномерное распределение записей по страницам. В этом случае принимают нужное количество страниц несколько большим теоретического их отношение называется коэффициентом нагрузки. Для нашего примера возьмем 30 страниц и получим коэффициент загрузки 25/30≈83%. Хэшфункцией будет остаток от деления значения ключа на 30, что гарантирует получение номера страницы в диапазоне от 0 до 29 независимо от номера ключа. Найти запись на странице можно простым перебором.

Например, страница для записи с ключом 13672 — 22, с ключом 63245 — 5, с ключом 82 — также 22 и т.д.

Может иметь место ситуация (нр, все ключи кратны 30), когда на одну страницу претендует больше записей, чем она может вместить.

Одним из методов решения конфликта является преобразование значения ключа методом квадратичных частных: R+Q*i2+i, где R — значение остатка, Q — значение частного, i пробегает значения от 0 до тех пор, пока конфликт не исчезнет.

Например, для ключа 82 имеем R = 82 mod 30 = 22, Q = 82 div 30 = 2.

i=0 страница = (22 + 2*02+0) mod 30 = 24
i=1 страница = (22 + 2*12+1) mod 30 = 25
i=2 страница = (22 + 2*22+2) mod 30 = 2
i=3 страница = (22 + 2*32+3) mod 30 = 13
...
i=10 страница = (22 + 2*102+10) mod 30 = 22

пока не будет найдена незаполненная страница или не найдена нужная запись.

Другим методом решения конфликта может служить введение страницы переполнения, на которой разместятся новые записи, но это еще одна дисковая операция ввода/вывода. Страниц переполнения

может быть несколько, они связываются указателем. С увеличением файла количество совпадений адресов увеличивается, что приводит к увеличению среднего времени доступа, т.к. все больше времени придется тратить на поиск записи в возросшем количестве страниц. Это можно устранить, если реорганизовать записи, т.е. используя новую хешфункцию разместить их на большем пространстве, что решается при помощи расширяемого хеширования или динамических хэшфункций.

При использовании расширяемого хеширования необходимо, чтобы все значения хешполя были уникальны, а это может быть реализовано только в том случае, если хешполе является ключевым. Основные принципы работы метода расширяемого хеширования следующие:

- если в качестве хешфункции используется функция φ, а значение ключевого поля для некоторой записи z равно p, то в качестве результата хеширования значения ключевого поля будет получено значение псевдоключа для записи z в виде φ(p). Здесь псевдоключ используется не в качестве адреса записи, а лишь как косвенный указатель на место их хранения;
- хранимый файл имеет связанный с ним каталог, который также сохраняется на диске. Он состоит из заголовка, содержащего значение g, которое называется глубиной каталога, а также 2g указателей на страницы с несколькими записями данных на каждой.

Таким образом, с помощью каталога глубиной g можно организовать доступ к файлу, содержащему 2g различных страниц с данными.

Если рассматривать первые g бит псевдоключа как целое беззнаковое двоичное число b, то ій указатель в каталоге будет относиться к странице, содержащей все записи, для которых величина b равна i–1. Для того чтобы найти запись со значением ключевого поля, равным p, следует с помощью хешфункции вычислить значение псевдоключа, а затем по первым g бит псевдоключа определить численное значение i1 и найти в каталоге соответствующий ему ій указатель на страницу, содержащую искомую запись, что реализуется за два доступа к диску.

Здесь был описан лишь один из вариантов хеширования, однако существует множество других способов реализации этой основной идеи.

Основные этапы доступа к базе данных

Говоря о процессе работы с БД нельзя не остановиться на вопросах хранения и методов доступа к данным. Вообще говоря, основные проблемы, связанные с физическим хранением данных, вызваны медленностью доступа и поиска, а также низкой скоростью передачи, поэтому основной целью повышения производительности системы с этой точки зрения является минимизация числа дисковых операций вводавывода данных.

Для хранения данных могут быть использованы различные структуры, обладающие разной производительностью, однако идеального способа хранения данных не существует. По этой причине СУБД должна содержать несколько структур хранения данных для разных задач и частей системы, а также предусматривать возможность изменения способов хранения в зависимости от изменяющихся требований к производительности системы.

Начнем с описания основных этапов процесса доступа к БД. Сначала СУБД определяет искомую запись в БД, для чего в оперативную память помещается набор записей, в котором ищется запрашиваемая, а для извлечения записи запрашивается так называемый диспетчер файлов. Диспетчер файлов определяет страницу, на которой находится искомая запись, а затем для извлечения этой страницы запрашивается диспетчер дисков. Диспетчер дисков определяет физическое расположение страницы на устройстве хранения информации и посылает запрос на вводвывод данных.

Таким образом, СУБД рассматривает БД как множество записей, просматриваемых при помощи диспетчера файлов. Последний рассматривает БД как набор страниц, просматриваемых с помощью диспетчера дисков, который уже непосредственно работает с устройствами хранения информации.

Заметим, что диспетчер дисков является частью операционной системы, с помощью которого выполняются все дисковые операции вводавывода. Для того, чтобы выполнять эти операции, диспетчеру необходимо обладать информацией о значениях физических адресов на диске, где располагаются те или иные данные, однако, диспетчеру файлов такая информация совсем не нужна — вместо этого ему достаточно рассматривать диск как набор страниц строго фиксированного размера с уникальным идентификационным номером набора страниц. В свою очередь, каждая страница. обладает уникальным внутри данного набора идентификационным номером страницы, причем наборы не имеют общих страниц. При этом соответствие физических адресов на диске и номеров страниц достигается с помощью диспетчера дисков. Важнейшим преимуществом такой организации хранения данных является изоляция программного кода внутри диспетчера дисков. зависящего от конкретного устройства диска, за счет чего многие компоненты системы могут быть аппаратно независимыми.

Все страницы диска делятся на несвязанные наборы, а один из таких наборов, содержащий пустые страницы, соответственно содержит все имеющиеся свободные страницы, не используемые для размещения данных. Этот набор иногда называют свободным пространством на диске. При этом использование или освобождение страниц, из наборов страниц осуществляется диспетчером дисков по запросу диспетчера файлов.

Основные операции, выполняемые диспетчером дисков с наборами страниц по запросу со стороны диспетчера файлов, следующие:

- извлечь страницу s из набора страниц n;
- заменить страницу s из набора страниц n;
- добавить новую страницу в набор страниц n;
- удалить страницу s из набора страниц n.

При работе с диском, как с набором хранимых файлов, диспетчер файлов использует все имеющиеся средства диспетчера дисков, при этом каждый набор страниц может содержать один или несколько хранимых файлов.

Каждый хранимый файл имеет уникальные в рассматриваемом наборе страниц имя или идентификационный номер, а каждая хранимая запись обладает идентификационным номером записи, уникальным в пределах данного хранимого файла. С помощью операций с файлами в СУБД можно создавать структуры хранения и управлять, но следует иметь в виду. что в одних системах диспетчер файлов является компонентом операционной системы, а в других является частью СУБД однако принципы его работы существенно от этого не различаются. Основные операции с файлами, выполняемые диспетчером файлов, по запросу со стороны СУБД следующие:

- извлечь хранимую запись z из хранимого файла f;
- заменить хранимую запись z в хранимом файле f;
- добавить новую хранимую запись z в хранимый файл f;
- удалить хранимую запись z из хранимого файла f;
- создать новый хранимый файл f;
- удалить хранимый файл f.

При хранении данных используют принцип кластеризации данных, в основе которого находится подход как можно более близкого физического размещения на диске логически связанных между собой и часто используемых данных. Физическая кластеризация данных достаточно важное условие высокой производительности, при этом различают внутрифайловую кластеризацию. Когда она осуществляется в рамках одного хранимого файла. Например, если в системе часто требуется осуществлять доступ к данным согласно порядковому номеру, то все записи следует физически размещать таким образом, чтобы первая запись была возле второй записи, вторая запись — возле третьей и т.д.

Другой вариант кластеризации — межфайловая, когда ею охватывается сразу несколько файлов. Это используют, если в системе часто требуется осуществлять доступ к неким записям и к данным, связанными с ними, при этом первые стараются разместить рядом со вторыми.

Разумеется, в каждый момент времени кластеризацию файла или набора файлов можно осуществлять только одним из этих способов. Внутрифайловую и межфайловую кластеризацию СУБД может

осуществлять, размещая логически связанные записи на одной странице, если это возможно, или на соседних страницах. По этой причине СУБД важно иметь сведения не только о сохраненных файлах, но и о страницах: при создании в СУБД новой записи необходимо, чтобы она с помощью диспетчера файлов была размешена возле некоторой текущей страницы, т.е. на той же или, по крайней мере. на логически близкой странице В свою очередь, диспетчер дисков должен обеспечить, чтобы логически близкие страницы были физически близко расположены на диске.

Конечно, кластеризация внутри СУБД возможна только в том случае, если администратор БД организует ее, при этом часто предусматривается использование нескольких различных типов кластеризации данных из разных файлов, при необходимости выбирая тот или иной ее тип.

Управление страницами

Основной функцией диспетчера дисков является скрытие от диспетчера файлов всех деталей физических дисковых операций вводавывода и замена их логическими страничными операциями вводавывода. Эта функция диспетчера дисков называется управлением страницами, которая реализуется в следующей последовательности, например, если в каждой используемой таблице требуется логически упорядочить записи согласно ключевому полю.

На начальном этапе БД совсем не содержит данных, но в ней имеется набор пустых страниц, последовательно пронумерованных начиная с номера один, в котором содержатся все страницы диска, за исключением страницы с нулевым номером, которой отводится особая роль. Для размещения записей с данными X диспетчер файлов создаст набор страниц и разместит на них данные X. С этой целью диспетчер дисков переместит соответствующее количество страниц из набора пустых страниц и пометит их как набор страниц данных Y. Аналогичные действия будут выполнены для размещения данных Y, Z и т. д., но для простоты ограничимся тремя наборами. В результате будет создано четыре набора страниц: данные X, данные Y, данные Z и набор пустых страниц.

Предположим, что необходимо добавить новую запись с данными X, для этого диспетчер файлов вставляет новую хранимую запись, а диспетчер дисков осуществляет поиск первой пустой страницы, а затем добавляет ее к набору страниц данных X.

Если необходимо удалить хранимую запись с данными X, диспетчер файлов удаляет ее, а диспетчер дисков возвращает освободившуюся страницу данных X в набор пустых страниц.

Аналогичным способом вставляются или удаляются записи из наборов страниц с данными Y или Z, из чего можно сделать вывод о том, что после выполнения нескольких самых обычных действий нельзя гарантировать, что логически близкие страницы будут физически располагаться одна возле другой. Поэтому логическую последовательность страниц в данном наборе следует задавать с помощью указателей, а не на основе их физически близкого размещения на диске. Для этого каждая страница содержит заголовок страницы с информацией о физическом дисковом адресе страницы, которая логически следует за данной страницей. Особенности использования указателей следующие:

- заголовки страниц, в частности указатели следующей страницы, обрабатываются диспетчером дисков и должны быть скрыты для диспетчера файлов;
- желательно сохранять логически связанные страницы в физически близких фрагментах диска, поэтому диспетчер дисков обычно размещает или удаляет страницы в наборах не по одной, а целыми блоками физически связанных страниц;
- для получения информации о размещении различных наборов страниц диспетчером дисков, на диске организуется отдельная страница, в которой сохраняется вся необходимая для этого информация. Эта страница часто называется таблицей размещения или просто страницей 0, где перечислены все имеющиеся на данном диске наборы страниц вместе с указателями на первые страницы каждого из наборов.

Диспетчер дисков скрывает особенности физической организации вводавывода от диспетчера файлов и предоставляет ему возможность вести работу только на логическом уровне. Аналогично диспетчер файлов скрывает все подробности операций вводавывода на основе

страниц от СУБД и предоставляет ей возможность вести работу только с хранимыми записями и файлами. Такая работа, выполняемая диспетчером файлов, называется управлением хранимыми записями.

Предположим, что на одной странице могут быть размещены не одна. а несколько хранимых записей. При вставке небольшого количества хранимых записей, например, данных X, на соответствующей странице теперь может оставаться достаточно свободного пространства. Поэтому, если возникнет необходимость вставить новую запись данных X. то диспетчер файлов сохранит эту запись на той же странице вслед за последней сохраненной записью.

Теперь, при возникновении необходимости удалить существующую запись данных X, это выполнит диспетчер файлов и, если образовался пустой промежуток между записями на странице, он передвинет записи к началу страницы. Это говорит о том, что логическая последовательность хранимых записей для данной страницы может соответствовать физической последовательности, заданной внутри этой страницы. Для этого диспетчер файлов может передвигать отдельные записи вверх или вниз. размещая все записи в верхней части страницы и оставляя свободное пространство в нижней части страницы Более того. если СУБД необходимо вставить новую запись на рассматриваемую страницу, то диспетчер файлов разместит запись в соответствии с ее идентификационным номером.

Итак, хранимые записи идентифицируются с помощью идентификационного номера записи z. который состоит из двух частей: номера страницы s. на которой данная запись находится, и информации о смещении записи от конца страницы s. Последняя. в свою очередь, содержит информацию о смещения записи z от начала страницы s. Эта схема в некоторой степени сочетает быстроту непосредственной адресации и гибкость косвенной адресации. т.к. записи внутри страницы могут сдвигаться без изменения идентификационных номеров записей, а корректируются только значения локальных смещений в конце страницы. К тому же, если известен идентификационный номер записей, доступ к требуемой записи осуществляется достаточно быстро, поскольку используется только доступ к данной странице.

Иногда для работы с некоторой записью может потребоваться доступ к двум страницам. Такая ситуация может возникнуть, например, если длина записи превышает размер страницы. Тогда такая запись будет размещена на специальной странице переполнения. а исходный указатель на идентификационный номер прежней записи будет заменен новым. При следующем переполнении страницы запись опять перемещается на новую страницу переполнения с соответствующим изменением указателя.

Следует обратить внимание на то, что для некоторого хранимого файла всегда можно осуществить последовательный доступ ко всем хранимым записям, т. е. доступ согласно последовательности записей внутри страницы и последовательности страниц внутри набора страниц, чаще всего, в порядке возрастания идентификационных номеров записей. Такая последовательность называется физической, хотя надо понимать, что она не обязательно соответствует физическому расположению данных на диске. Доступ к хранимому файлу можно осуществить согласно физической последовательности, даже если несколько файлов находятся на одной и той же странице, при этом записи. Которые не относятся к искомому файлу, будут пропущены при последовательном просмотре содержания данной страницы.

Кроме того, поля хранимой записи используются в СУБД для роставления индексов и т.д., однако диспетчер файлов эту информацию не используют. Таким образом, важным отличием между диспетчером файлов и СУБД является представление хранимой записи: с точки зрения СУБД хранимая запись обладает внутренней структурой, а с точки зрения диспетчера файлов это всего лишь строка байтов.

Однако существуют более совершенные способы упорядочения записей и способы доступа по сравнению с физической последовательностью: использование индексирования, хеширования, цепочек указателей, а также технологии сжатия. При этом часто эти способы используются совместно один на основе другого.

Сжатие данных

С целью сокращения пространства, необходимого для хранения некоторого набора данных, часто используют технологии сжатия. При этом в результате экономится не только пространство на диске, но и количество дисковых операций вводавывода, т. к. доступ к данным меньшего размера требует меньше дисковых операций вводавывода. С другой стороны, для распаковки и извлечения сжатых данных требуются некоторые дополнительные манипуляции, но в целом преимущества сокращения операций вводавывода могут компенсировать недостатки, связанные с дополнительной обработкой данных.

Технологии сжатия основаны на малой вероятности того, что данные имеют совершенно беспорядочную структуру. Наиболее распространенной является технология сжатия на основе различий, при которой некоторое значение заменяется сведениями о его отличиях от предыдущего значения. Следует отметить, что для реализации такой технологии требуется размещать данные последовательно, поскольку для их распаковки необходимо иметь значение предыдущей величины. Такое сжатие весьма эффективно для данных, к которым необходим последовательный доступ, например, для записей в одноуровневом списке. Более того. в таких случаях наряду с данными допускается также сжать и указатели. Дело в том, что если логическая последовательность в файле соответствует физической последовательности размещения данных на диске, то соседние указатели будут незначительно отличаться друг от друга, а значит, сжатие указателей может оказаться весьма полезным и эффективным.

В качестве примера рассмотрим несколько записей со следующими данными:

Студент

Студентка

Студенческий

Предположим, что длина поля составляет 15 символов и справа от данных в несжатом виде содержится соответствующее количество пробелов. Один из способов применения сжатия на основе различий — это удаление повторяющихся символов в начале каждой записи с

указанием их количества, т. е. переднее сжатие В результате будет получено (числа соответствуют количеству повторяющихся символов в начале имени, пробелы справа до полной длины поля — 15 символов — не показаны):

0 — Студент

7 — ка

6 — ческий

При необходимости можно осуществить дополнительное ежа тие. удаляя пробелы с указанием их количества, т. е. выполнить так называемое заднее сжатие. Иногда еще допускается удаление с правого конца каждого значения всех повторяющихся символов в двух ближайших соседних значениях.

Другим способом уменьшения объема, занимаемого данными места является иерархическое сжатие. Предположим, что в хранимом файле задана некоторая физическая последовательность согласно выбранному полю, различные значения которого располагаются в нескольких последовательных записях этого файла.

Например, в хранимом файле успеваемости благодаря кластеризации, выполненной согласно полю PN с названиями предметов, отдельно содержатся все записи о студентах, сдавших тот или иной предмет. В таком случае набор всех записей с данными о студентах, сдавших тот или иной предмет, может быть успешно сжат в одну хранимую иерархическую запись, при этом название предмета будет упомянуто только один раз. а вслед за ним будут расположены данные о студентах (см. рис. 1.33).

Хранимая иерархическая запись состоит из двух частей: постоянной, в нашем примере это поля с названиями предметов, и переменной — записи с информацией о студентах. Такой набор значений переменного количества внутри одной записи обычно называется группой повторения.

Иерархическое сжатие такого типа применяют и для индекса, в котором несколько последовательно расположенных значений содержат одни и те же повторяющиеся значения данных, но различные значения указателей.

Аналогичным образом можно применять иерархическое сжатие на основе межфайловой кластеризации. Например, записи с данными о студентах как бы объединяются с данными о всех оценках по всем предметам для данного студента.

В заключение отметим, что структуру на основе цепочки указателей, вообще говоря, можно рассматривать как межфайловое сжатие, которое не требует межфайловой кластеризации, т. к. указатели позволяют логически достичь эффекта кластеризации.

Кодирование Хафмана — это еще одна технология кодирования символов, которая может быть весьма эффективной для сжатия различных символов, встречающихся с разной частотой. Основная идея этого метода заключается в кодировании отдельных символов битовыми строками различной длины, причем наиболее часто встречающиеся символы кодируются Строками наименьшей длины. Кроме того, код любого символа длиной п не должен совпадать с первыми n символами кода какоголибо другого символа.

Предположим, что некоторые данные записаны с помощью символов А. Б. В. Г, Д, тогда с учетом относительной частоты, с которой они встречаются, их коды приведены в таблице.

Символ	Частота, %	Код
А	35	1
В	30	01
Г	20	001
Д	10	0001
Б	5	0000

Символ А встречается чаще остальных, и потому имеет самый короткий код. состоящий из одного бита. Все остальные коды должны быть длиннее, однако нельзя использовать код на основе одного нуля, так как он будет совпадать с начальной частью других. более длинных кодов. Оценочно можно сказать, что в среднем общая длина закодированного текста на 40% меньше, чем при отсутствии кодирования.

Таким образом, рассмотрены важнейшие современные структуры хранения данных, описана общая схема функционирования программного обеспечения, предназначенного для доступа к данным, а также распределение выполнения этих задач между СУБД, диспетчером файлов и диспетчером дисков. Основной целью изложения было разъяснение общих идей и принципов без описания отдельных подробностей их реализации в разных практически используемых системах и структурах хранения.

SQL-сервер INTERBASE. Основные компоненты

SQLсервер InterBase предназначен для хранения и обработки больших объемов информации в условиях одновременной работы с БД множества клиентских приложений. Масштаб информационной системы при этом произволен — от системы уровня рабочей группы (под управлением Novell Netware или Windows 32 на базе IBMсовместимых ПК) до системы уровня большого предприятия (на базе серверов IBM, HewlettPackard, SUN).

Ниже рассматривается ряд технологий InterBase, использование которых обеспечивает максимальную вычислительную разгрузку клиентского приложения и гарантирует высокую безопасность и целостность информации.

Для задания ссылочной и смысловой целостности в БД определяются:

- отношения подчиненности между таблицами БД путем определения первичных (PRIMARY) ключей у родительских и внешних (FOREIGN) ключей у дочерних таблиц;
- ограничения на значения отдельных столбцов (CONSTRAINT); условия ограничений могут быть разнообразны — от требования удовлетворения вводимых значений определенному диапазону или соответствия некоторой маске до требуемого отношения с одной или несколькими записями из другой таблицы (или многих таблиц) БД;
- триггеры (TRIGGER) — подпрограммы, автоматически выполняемые сервером до или (и) после события изменения записи в таблице БД;

– генераторы (GENERATOR) для создания и использования уникальных значений нужных полей.
– Для ускорения работы клиентских приложений с удаленной БД могут быть определены хранимые процедуры (STORED PROCEDURE), которые представляют собой подпрограммы, принимающие и возвращающие параметры и способные выполнять запросы к БД, условные ветвления и циклическую обработку. Хранимые процедуры пишутся на специальном алгоритмическом языке. В них программируются часто повторяемые последовательности запросов к БД. Текст процедур хранится на сервере в откомпилированном виде. Преимущества в использовании хранимых процедур очевидны:
    1. отпадает необходимость синтаксической проверки каждого запроса и его компиляции перед выполнением, что убыстряет выполнение запросов;
    2. отпадает необходимость реализации в клиентской программе запросов, определенных в теле хранимых процедур;
    3. увеличивается скорость обработки транзакций, т. к. вместо подчас длинного SQLзапроса по сети передается относительно короткое обращение к хранимой процедуре.

В составе записи БД могут определяться BLOBполя (Binary Large Object — большой двоичный объект), предназначенные для хранения больших объемов данных в виде последовательности байтов. Таким образом могут храниться текстовые и графические документы, файлы мультимедиа, звуковые файлы и т. д. Интерпретация BLOBполя выполняется в приложении, однако разработчик может определить так называемые BLOBфильтры для автоматического преобразования содержимого BLOBполя к другому виду.

InterBase дает возможность использовать определяемые пользователем функции (User Defined Function, UDF), в которых могут реализовываться функциональности, отсутствующие в стандартных встроенных функциях InterBase (вычисление максимума, минимума, среднего значения, преобразование типов и приведение букв к заглавным). Например, в UDF можно реализовать извлечение из значения даты номера дня, года; определение длины символьного

значения; усечение пробелов; разные математические алгоритмы и т. п. Функция пишется на любом алгоритмическом языке, позволяющем разрабатывать DLL (библиотеки динамического вызова), например, на Object Pascal.

InterBase может посылать уведомления клиентским приложениям о наступлении какоголибо события (EVENT). Одновременно работающие приложения могут обмениваться сообщениями через сервер БД, вызывая хранимые процедуры, в которых реализована инициация нужного события.

Для обеспечения быстроты выполнения запросов и снятия с клиентского приложения необходимости такие запросы выдавать в БД можно определить виртуальные таблицы (или просмотры, VIEW), в которых объединяются записи из одной или более таблиц, соответствующих некоторому условию. Работа с просмотром из клиентского приложения ничем не отличается от работы с обычной таблицей. Поддерживает просмотр сервер, реагируя на изменение данных в БД. Просмотры могут быть изменяемыми или не допускающими внесения в них изменений.

Для доступа к БД используется утилита Windows Interactive SQL (WISQL). Она работает с БД напрямую через InterBase API, минуя BDE. С помощью WISQL можно писать любые запросы к серверу, будь то создание БД, таблиц, изменение структуры данных, извлечение данных из БД или их изменение, а также назначение прав доступа к информации для отдельных пользователей.

Для управления SQLсервером в целом и отдельными БД в частности используется утилита InterBase Server Manager. С ее помощью можно определять параметры SQLсервера, производить сохранение, восстановление БД, сборку «мусора», определять новых пользователей, их пароли и т.д.

Для просмотра БД, работы с таблицами, индексами, доменами, ограничениями и др. могут использоваться утилиты Database Desktop (весьма ограниченно) и SQL Explorer.

Для просмотра и анализа реальных процессов, происходящих на сервере при реализации пользовательского запроса, используется утилита SQL Monitor.

Технические характеристики

InterBase был разработан в начале 80х годов группой разработчиков из американской корпорации DEC. В дальнейшем разработка данного продукта велась независимыми компаниями InterBase Software и впоследствии слившейся с ней Ashton¬Tate. Borland приобрела права на InterBase у AshtonTate после слияния с нею.

InterBase активно используется в государственном и военном секторах США, что, видимо, и стало преградой для его продвижения в Россию. Интерес к этому серверу возрос только в последнее время в связи с включением его локальной (а начиная с Delphi 3, и 4пользовательской) версии в состав Delphi Client/Server Suiti и Delphi Enterprise. Внимание разработчиков БД и приложений InterBase привлек вопервых, потому, что это «родной» продукт Borland (а средства разработки приложений этой компании давно зарекомендовали себя с положительной стороны;

вовторых, потому что InterBase весьма прост в установке, настройке и администрировании по сравнению с другими SQLсерверами, и втретьих, потом что он обладает прекрасными функциональными возможностями.

Ниже приводятся некоторые технические характеристики сервера (К – 1024 байта).

Характеристика	Значение
Максимальный размер одной БД	Рекомендуется не выше 10 Гбайт Однако известны случаи объема одной БД в 10-20 Гбайт.
Максимальное количество таблиц в одной БД	65536
Максимальное количество полей (столбцов) в одной таблице	1 000 Не ограничено
Максимальное количество записей в одной таблице Максимальная длина записи	64 К (не считая полей *BLOB)*

Максимальная длина поля Максимальная длина поля BLOB. Максимальное количество индексов в БД	32 К (кроме полей *BLOB)* Не ограничена 65536
Максимальное количество полей в индексе Максимальное количество вложенностей SQL-запроса Максимальный размер хранимой процедуры или триггера	161648 К
Максимальное количество UDF в базе данных	Длина имени *UDF* - не более 31 символа, каждая *UDF* должна иметь уникальное имя, поэтому максимальное количество *UDF* ограничивается только требованием уникальности имен

База данных InterBase состоит из последовательно, начиная с 0, пронумерованных страниц. Нулевая страница является служебной и содержит информацию, необходимую для соединения с БД. Размер страницы 1 (по умолчанию), 2, 4 или 8 Кбайт. Он устанавливается при создании БД, но может быть изменен при сохранении и восстановлении базы. Одна страница читается сервером за один логический доступ к БД.

Объем буфера вводавывода для операций чтениязаписи определяется в количестве страниц (по умолчанию 75). Если БД будет чаще читаться, объем буфера следует увеличить. Если в нее будет чаще осуществляться запись, размер буфера можно уменьшить.

В InterBase поддерживается многоверсионная структура записей. При изменении записи какойлибо транзакцией создается новая версия записи, куда помимо данных записывается номер транзакции и указатель на предыдущую версию записи. Старая версия помечается как измененная; ее указатель на следующую версию записи содержит ссылку на вновь созданную версию. Каждая стартующая транзакция работает с последней версией записи, изменения для которой подтверждены. Таким образом, параллельно работающие с БД транзакции всегда используют разные версии записей, что позволяет снимать блокировки для клиентских приложений, одновременно

работающих с одними и теми же данными в БД. Более подробно об этом рассказано в разделе, посвященном управлению транзакциями. При удалении записи она также физически не удаляется с диска, а помечается как удаленная до тех пор, пока не завершена хотя бы одна активная транзакция, использующая эту запись.

InterBase располагает на одной странице БД версии одной записи таблицы БД. После удаления записей на странице образуются «дырки». При добавлении новой записи анализируется размер максимальной «дырки» и, если он меньше длины добавляемой записи, происходит компрессия страницы, в процессе которой «дырки» объединяются. Если освободившегося пространства не хватает для размещения новой записи, та записывается с новой страницы. Загрузка страницы считается нормальной в случае, если «дырки» занимают не более 20% объема страницы.

Выделение страниц никак не оптимизировано. На отдельной служебной странице БД хранятся номера всех свободных страниц. При выделении страниц не предпринимается никаких действий по выделению непрерывных страниц для хранения записей одной таблицы БД, а выделяется первая страница в списке свободных. Если свободной страницы нет, добавляется новая в конец БД. Только в этом случае размер БД возрастает.

Многоверсионная структура записей и неоптимальное выделение страниц ведет к высокой фрагментации БД и как следствие к замедлению работы с ней. Поэтому необходимо периодически производить дефрагментацию БД. Дефрагментированная БД характеризуется расположением записей таблиц БД на непрерывных страницах и отсутствием «мусора». Под мусором понимаются версии записей, с которыми не работает никакая активная транзакция: многоверсионный механизм гарантирует, что вновь стартующая транзакция не будет работать с ранней версией записи, если имеются ее более поздние версии.

Существует несколько способов проведения дефрагментации.

Первый состоит в сохранении БД на дисковом носителе и последующем ее восстановлении из сделанной резервной копии с помощью утилиты InterBase Server Manager. Этот способ является

предпочтительным, поскольку гарантирует сбор всего мусора (в момент сохранения и восстановления БД не должно быть активных подключений к БД со стороны иных пользователей и потому не может быть активных транзакций).

Второй способ состоит в автоматическом сборе мусора. Интервал, через который происходит сборка мусора, измеряется в транзакциях и по умолчанию составляет 20 000 транзакций. Это значение может быть изменено с помощью InterBase Server Manager. Данный способ дефрагментации БД менее предпочтителен, поскольку удаляются только те версии записей, для которых нет активных транзакций. В результате могут быть удалены не все старые версии. При большом числе активных транзакций процесс сборки мусора может существенно замедлить их выполнение.

Если на вашей машине установлен InterBase (локальный или многопользовательский), его старт происходит автоматически при загрузке

операционной системы. Об этом сигнализирует значок справа на нижней панели Windows 32. Щелкнув на этом значке правой кнопкой мыши, можно вызвать вспомогательное меню. Опция Startup Configuration этого меню позволяет просмотреть и переопределить стартовые установки InterBase. Опция Shutdown завершает работу SQLсервера. Опция Properties позволяет просматривать свойства InterBase и текущей сессии, например, число активных подключений и число используемых БД.

На заметку. За работой сервера внимательно следит утилита InterBase Guardian — это именно ее пиктограмма видна на нижней панели Windows. Эта утилита осуществляет начальный запуск сервера, а также его перезапуск, если по какимлибо причинам сервер «рухнул». Фактически восстановление работоспособности сервера происходит в доли секунды, так что многие пользователи даже, возможно, не заметят сбоя в его работе.

DELPHI и INTERBASE

Delphi предназначена для создания программ, работающих как с локальными, так и с удаленными данными. В последнем случае с

помощью Delphi создаются клиентские программы (в многозвенной архитектуре с помощью Delphi создаются также и серверы приложений — промежуточное звено между клиентом и SQLсервероы).

Создание программ в архитектуре клиентсервер имеет следующую специфику:

- не рекомендуется использовать компонент TTable, поскольку он требует передачи всех данных результата выполнения запроса к серверу; в отличие от этого компонент TQuery получает от сервера только ту их часть, которая должна быть визуализирована;
- изменение записей БД следует производить не методами Insert, Edit, Delete, Post, Cancel, которые оперируют с одной (текущей) записью, а при помощи SQLоператоров INSERT, UPDATE, DELETE, которые оперируют сразу множеством записей;
- необходимо особое внимание уделять управлению транзакциями и в первую очередь — выбору адекватного потребностям программы уровня изоляции транзакций;
- бизнесправила, где это возможно, нужно переносить на сервер, разгружая от них клиентское приложение;
- следует как можно чаще использовать хранимые процедуры, выполняющиеся быстрее обычных SQLзапросов и уменьшающие загрузку сети;
- следует везде, где это необходимо, явно вызывать методы Tdatabase.StartTransaction и TDatabase.Commit для старта и подтверждения транзакций;
- подтверждение единичных изменений БД неявно стартуемыми и завершаемыми (в режиме SQLPASSTHRU = SHARED A UTOCOMMIT) транзакциями ведет к возрастанию загрузки сети и, как следствие, к замедлению работы, часто весьма существенному;
- необходимо уделять существенное внимание оптимизации запросов к БД, особенно при чтении данных (SELECT), поскольку оптимально построенный запрос может выполняться в несколько раз быстрее и требовать меньшего количества ресурсов.

Лекция 13. РАСПРЕДЕЛЕННЫЕ БАЗЫ ДАННЫХ

Трехзвенная архитектура

В архитектуре клиент-сервер задачей сервера является обработка запросов клиентов, которые зачастую могут формироваться по достаточно сложному алгоритму, тем самым утяжеляя клиентские приложения и повышая требования к аппаратному обеспечению клиента (рис. 13.1).

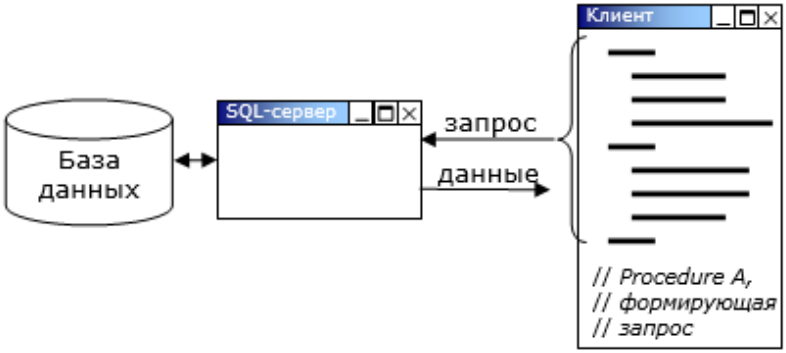


Рис. 13.1. Архитектура клиент-сервер

Трехзвенная архитектура является эволюцией архитектуры клиент-сервер. В ней между имеющимися двумя звеньями (клиентом и сервером) появляется промежуточное третье звено — сервер приложений (application server). Только он имеет доступ к серверу базы данных. Сервер приложений может располагатся либо на одной машине с сервером БД, либо на специально выделенной машине.

Сервер приложений реализует тот алгоритм, который раньше выполняли клиентские приложения.

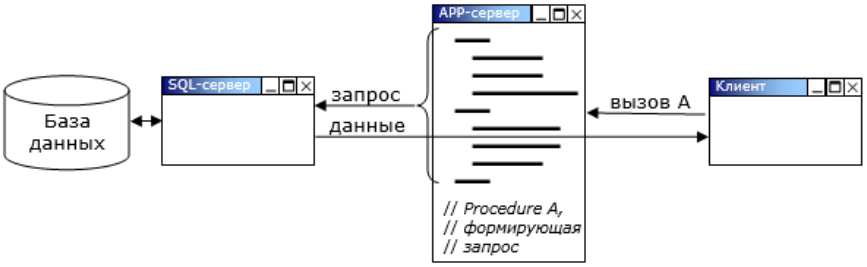


Рис. 13.2. Трехзвенная архитектура

В результате получается "облегченный" клиент (thin client), который достаточно нетребователен к ресурсам компьютера. Кроме того, в случае модификации процедуры, изменения никак не коснутся клиентов, которые попрежнему только запускают ее.

Третье преимущество, которое предоставляет эта технология заключается в том, что работа процедуры выполняется сервером, ресурсы которого, как правило, значительно лучше клиентских.

Связь клиента с сервером приложений реализуется при помощи технологии уделенного доступа, которая обеспечивает загрузку сервера приложений в адресное пространство машины сервера.

Ниже перечислены самые распространенные технологии удаленного доступа:

- DCOM — Distributed Component Object Model — распределенная компонентная модель объектов. Обязательное требование — сервер должен работать по управлением ОС Windows NT Server или Windows 2000 Server;
- сокетное соединение. Позволяет работать без Windows NT Server
- MTS — Microsoft Transaction Server — основан на DCOM и включает в себя некоторые дополнительные возможности по управлению системными ресурсами, а также с повышенной защищенностью данных;
- CORBA — Common Object Request Broker Architecture — общая архитектура брокеров объектных запросов;
- SOAP — Simple Object Access Protocol — простой протокол доступа к объекту.

Использование технологий распределенной обработки еще не делает базу данных распределенной.

Распределенные базы данных

В системах клиентсервер основной идеей является концентрация всех данных в одном месте под управлением сервера. Термин распределенная база данных говорит о децентрализации таких систем.

Ключевым моментом в определении распределенной базы данных является утверждение, что система работает с данными, физически распределнными в сети.

Одной из причин, по которой данные оказываются разделенными на фрагменты, является географическое удаление друг от друга рабочих мест, которые работают с едиными данными.

Основым принципом принципом при построении СУРБД является обеспечение прозрачности для конечного пользователя, т.е. он не замечает того факта, что данные с которыми он работает разделены на несколько фрагментов.

Каждый фрагмент распределенной базы данных находится под управлением локальной СУБД. Если тип каждой СУБД одинаков, то база данных называется гомогенной. Если же типы СУБД отличаются, то гетерогенной.

Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных БД, располагающихся в узлах вычислительной сети. с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем частям, как к единой БД. При этом должны обеспечиваться:

- простота использования системы;
- возможности автономного функционирования при нарушениях целостности сети или при административных потребностях:
- высокая степень эффективности.

Традиционно выделяют однородные и неоднородные распределенные БД. В однородных распределенных БД каждая локальная БД управляется одной и той же СУБД. В отличие от этого, в неоднородной системе локальные БД могут относиться к разным моделям данных, поэтому сетевая интеграция неоднородных БД — это актуальная, но очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удается справиться с главной проблемой — недостаточной эффективностью интегрированных

систем. Более успешно практически решается промежуточная задача — интеграция неоднородных SQLориентированных систем. Понятно, что этому в большой степени способствует стандартизация языка SQL и общее следование производителей СУБД принципам открытых систем.

Необходимость в распределенных БД возникла в связи с тем, что современные предприятия могут географически охватывать большие территории, причем работа с данными осуществляется в каждом подразделении и на определенном уровне. Распределение данных позволяет эффективно использовать имеющееся оборудование, рационально загрузить вычислительную сеть и повысить доступность БД.

Реализация распределенных БД преследует ряд целей, перечисленных ниже. В первую очередь при проектировании распределенной СУБД узлы системы следует делать в максимальной степени автономными, т. е. работа каждого узла как можно в большей степени не должна зависеть от успешного или неуспешного выполнения какойлибо операции в другом узле.

С другой стороны, все узлы распределенной БД должны рассматриваться системой как равноправные. Зависимость всей системы от какоголибо ее узла может привести к тому, что он станет наиболее уязвимым ее местом и, кроме того, в этом случае вряд ли можно будет реализовать условие автономности. Кроме того, распределенная БД должна обеспечивать высокую надежность и доступность.

Надежность рассматриваемой системы заключается в том, что система исправна и работает в заданный момент. Это реализуется за счет того, что система продолжает свою работу (иногда и с ограниченными возможностями) даже если имеет место сбой отдельного узла. Доступность системы определяется исправностью системы в течение длительного промежутка времени и возможностью быстрого выполнения манипуляций с данными.

Распределенная БД должна удовлетворять условию независимости от расположения данных — пользователю необходимо обеспечить такой режим работы с БД чтобы у него складывалось впечатление о

нераспределенном расположении данных, что позволяет, например, упростить реализацию пользовательских программ.

При использовании в распределенной БД разделения отношений для хранения, СУБД должна обеспечить независимость работы с данными, несмотря на то, что фрагменты могут быть расположены в различных узлах сети. Помимо этого, в системе должна поддерживаться независимость от репликации, если заданное отношение или его часть могут быть представлены в БД несколькими копиями (репликами), хранимыми на различных узлах.

И, наконец, распределенная БД не должна быть зависима от аппаратного обеспечения, операционной системы, вычислительной сети и от самой СУБД.

Основной целью построения распределенной БД можно считать обеспечение интеграции локальных БД. Для решения этой проблемы необходимо принять ряд проектных решений, касающихся декомпозиции исходного запроса, оптимального выбора способа выполнения запроса, согласованного выполнения транзакций, обеспечения синхронизации, обнаружения и разрешения распределенных тупиков, восстановления состояния БД после разного рода сбоев узлов сети.

Большинство современных СУБД автоматически обнаруживают текущее местоположение упоминаемых в запросе пользователя объектов данных: одна и та же прикладная программа. включающая предложения SQL, может быть выполнена в разных узлах сети. Обычно в каждом узле сети на этапе компиляции запроса выбирается наиболее оптимальный план выполнения запроса в соответствии с расположением данных в распределенной системе Традиционно, каждая локальная БД администрируется независимо от других. При этом возможны автономное подключение новых пользователей, смена версии автономной части системы и т.д.

Хорошо спроектированная распределенная система функционирует таким образом, что в ней не требуются централизованные службы именования объектов или обнаружения тупиков, а в отдельных узлах не требуется наличие общего знания об операциях, выполняющихся в других узлах сети. Кроме того, работа с доступными БД должна

продолжаться при выходе из строя отдельных узлов сети или линий связи.

Высокая степень эффективности системы является одним из наиболее ключевых требований к распределенным СУБД для достижения чего могут быть использованы два основных приема.

Вопервых, в СУБД ориентированных на распределенные данные. как правило, выполнению запроса предшествует его компиляция. В ходе этого процесса производится поиск употребляемых в запросе имен объектов БД в распределенном каталоге и замена имен на внутренние идентификаторы; проверка прав доступа пользователя, от имени которого производится компиляция, на выполнение соответствующих операций над данными и выбор наиболее оптимального глобального плана выполнения запроса, который затем подвергается декомпозиции и по частям рассылается в соответствующие узлы сети. После этого производится выбор оптимальных локальных планов выполнения компонентов запроса, т. е. большинство действий производится на стадии компиляции до реального выполнения запроса. Обработанный таким образом запрос может в дальнейшем выполняться много раз без дополнительных затрат вычислительной мощности и загрузки сети.

Вторым средством повышения эффективности системы является возможность перемещения удаленных отношений в локальную БД. Если это средство предусмотрено, то в ряде случаев оно может помочь добиться более эффективного прохождения транзакций.

Кроме того, хорошая система работы с распределенными БД должна иметь средства дублирования отношений в нескольких узлах с поддержкой согласованности копий и средства поддержания мгновенных снимков состояния БД в соответствии с заданным запросом.

Иногда в распределенных БД используют горизонтальное и вертикальное разделение отношений. В первом случае отношение делится на части по атрибутам, и полученные фрагменты хранятся в различных узлах сети. Второй случай связан с разделением отношения не по атрибутам, а по группам кортежей, которые тоже хранятся распределенными по узлам сети.

Такой подход позволяет понизить аппаратные требования ко всей системе, в ряде случаев увеличить скорость выполнения запросов и повысить надежность хранения данных, особенно при сбое. Трудности при этом возникают в обеспечении согласованности полученных разделов, а собственно разделенные отношения трудно использовать — при выполнении запросов учет наличия разделов отношения в разных узлах сети обычно производит оптимизатор, а, следовательно, количество потенциально возможных планов выполнения запросов, которые должны оцениваться оптимизатором, еще более возрастает.

Для реализации требования поддержки копий отношения в нескольких узлах сети должна производиться рассылка копий указанного отношения для хранения в именованных сегментах указанных узлов сети. Система должна автоматически поддерживать согласованность копий. Как и в случае разделенных отношений, кроме существенных проблем поддержания согласованности копий, проблемой является и разумное использование копий, наличие которых должно было бы учитываться оптимизатором.

Создание мгновенного снимка состояния БД в соответствии с заданным запросом обычно производится так, что результирующее отношение сохраняется под указанным пользователем именем в локальной БД в том узле. в котором выполняется запрос После этого мгновенный снимок периодически обновляется в соответствии с запомненным запросом,

Основное использование мгновенных снимков связано с тем. что их можно в некотором смысле рассматривать как материализованные представления БД. Большие проблемы связаны с обновлением отношений через их мгновенные снимки, поскольку в момент обновления содержимое мгновенного снимка может расходиться с текущим содержимым базового отношения.

По отношению к мгновенным снимкам проблем поддержания согласованного состояния мгновенного снимка и базовых отношений не существует, поскольку автоматическое согласование не требуется. Что же касается разделенных отношений и копий отношений, то для них эта проблема общая и достаточно трудная. Вопервых, согласование разделов и копий вызывает существенные

вычислительные затраты и загрузку сети при выполнении операций модификации хранимых отношений. Для этого требуется выработка и соблюдение специальных протоколов модификации. Вовторых, введение копий отношений обычно производится не столько для увеличения эффективности системы, сколько для увеличения доступности данных при нарушении связности сети. В системах, в которых применяется этот подход, при нарушении связности сети работа с распределенной БД обычно продолжается только в одной из образовавшихся подсетей, при этом для выбора подсети используются алгоритмы, основанные на учете количества связных узлов сети.

Как уже было сказано, для повышения производительности в распределенных БД используется предварительная компиляция. Будем называть главным узлом тот узел сети, в котором инициирован процесс компиляции запроса, и дополнительными узлами те узлы. которые вовлекаются в этот процесс в ходе его выполнения. Тогда процесс компиляции можно разбить на ряд фаз.

В главном узле производится грамматический разбор запроса с построением внутреннего представления в виде дерева. На основе информации из локального каталога главного узла и удаленных каталогов дополнительных узлов производится замена имен объектов, фигурирующих в запросе, на их системные идентификаторы. В главном узле генерируется глобальный план выполнения запроса, в котором учитывается лишь порядок взаимодействий узлов при реальном выполнении запроса. Если в глобальном плане выполнения запроса участвуют дополнительные узлы, производится его декомпозиция на части, каждую из которых можно выполнить в одном узле, после чего соответствующие части запроса рассылаются в дополнительные узлы. В каждом узле, участвующем в выполнении запроса, выполняется завершающая стадия выполнения компиляции, которая включает оптимизацию и генерацию машинных кодов. Затем производится проверка прав пользователя, от имени которого производится компиляция, на выполнение соответствующих действий и только после этого происходит обработка представлений БД.

Выполнение транзакции в распределенной СУБД начинается в главном узле, однако выполнение одной транзакции, вообще говоря,

инициирует транзакции и в дополнительных узлах. Основной здесь является проблема согласованного завершения распределенной транзакции, чтобы результаты ее выполнения во всех затронутых ею узлах были либо отображены в состояние локальных БД, либо полностью отсутствовали.

Для достижения этой цели обычно используется специальный механизм завершения распределенной транзакции, заключающийся в следующем: ряд независимых транзакцийучастников распределенной транзакции выполняются под управлением транзакциикоординатора, которой принимается решение об окончании распределенной транзакции. После этого выполняется первая фаза завершения транзакции, когда координатор передает каждому из участников сообщение о подготовке к завершению. Получив такое сообщение, каждый участник переходит в состояние готовности к немедленному завершению транзакции или к ее откату. После этого каждый участник, успешно выполнивший подготовительные действия, посылает координатору сообщение о готовности к завершению. Если координатор получает такие сообщения ото всех участников, то он начинает вторую фазу завершения. рассылая всем участникам команду о завершении транзакции, и это считается завершением распределенной транзакции. Если не все участники успешно выполнили первую фазу, то координатор рассылает всем участникам команду об откате транзакции, и тогда эффект воздействия распределенной транзакции на состояние БД отсутствует.

Здесь может возникнуть проблема распределенных тупиков, которые могут возникнуть между несколькими распределенными транзакциями, выполняющимися параллельно. Для обнаружения распределенных синхронизационных тупиков обычно применяются методы, основная идея которых состоит в том, что в каждом узле периодически производится анализ на предмет существования тупика с использованием информации о связях транзакций по ожиданию ресурсов, локальной в данном узле и полученной от других узлов При проведении этого анализа обнаруживаются либо циклы ожиданий. что означает наличие тупика, либо потенциальные циклы, которые необходимо уточнить в других узлах. Если обнаруживается наличие

синхронизационного тупика он разрушается за счет отката одной из транзакций, входящей в цикл, причем в качестве последней выбирается транзакция, выполнившая к этому моменту наименьший объем работы.

Как уже было сказано выше. одним из требований к распределенной БД является независимость от СУБД, однако некоторые системы, с одной стороны, не являются полностью или частично совместимыми друг с другом, а с другой — достаточно часто возникает объективная потребность в объединении различных систем. В таком случае прибегают к помощи специальных надстроечных программ, называемых шлюзами. Работа шлюзов заключается в том, что за счет их использования одна СУБД видит работу другой в понятном для нее виде, для чего используются общие протоколы обмена информацией, типы данных (или осуществляется преобразование одних типов данных в другие) и обеспечение совместной реализации блокировки, выполнения транзакций и т.д.

Применительно к СУБД архитектура клиентсервер интересна и актуальна главным образом потому, что обеспечивает простое и относительно дешевое решение проблемы коллективного доступа к БД в локальной сети. В некотором роде системы, основанные на архитектуре клиентсервер, являются упрощенным приближением к распределенным системам, не требующим решения основного набора проблем действительно распределенных БД. Реальное распространение архитектуры клиентсервер стало возможным благодаря развитию и широкому внедрению в практику концепции открытых систем.

Основной идеей открытых систем является упрощение сопряжения вычислительных систем за счет стандартизации аппаратной и программной части. Главной причиной развития концепции открытых систем явился переход к использованию локальных вычислительных сетей и проблемы сопряжения аппаратнопрограммных средств, которые вызвал этот переход В связи с бурным развитием технологий глобальных коммуникаций открытые системы приобретают еще большее значение и масштабность.

Важнейшим моментом открытых систем, направленных в сторону пользователей, является независимость от конкретного поставщика.

Ориентируясь на продукцию компаний, придерживающихся стандартов открытых систем, потребитель, приобретающий любой продукт такой компании, не попадает в зависимость от нее. Он может продолжить наращивание мощности своей системы путем приобретения продуктов любой другой компании, соблюдающей стандарты, причем это касается как аппаратных, так и программных средств.

Практической опорой системных и прикладных программных средств открытых систем является стандартизованная операционная система: в настоящее время такой системой является UNIX. Фирмампоставщикам различных вариантов ОС UNIX в результате длительной работы удалось придти к соглашению об основных стандартах этой операционной системы, и сейчас все распространенные версии UNIX в основном совместимы по части интерфейсов, предоставляемых пользователям. Технологии и стандарты открытых систем обеспечивают реальную и проверенную практикой возможность производства системных и прикладных программных средств при простоте переноса программной системы независимо от аппаратнопрограммных средств, соответствующих стандартам. Прежде всего, открытые системы обеспечивают естественное решение проблемы поколений аппаратных и программных средств — производители таких средств не вынуждаются решать все проблемы заново; они могут по крайней мере временно продолжать концепцию совместимости системы, используя существующие компоненты.

Преимуществом для пользователей таких систем является и то, что они могут постепенно заменить компоненты системы на более совершенные, не утрачивая работоспособности системы. В частности, в этом кроется решение проблемы постепенного наращивания вычислительных, информационных и других мощностей СУБД.

В основе широкого распространения локальных сетей компьютеров лежит известная идея разделения ресурсов. Высокая пропускная способность локальных сетей обеспечивает эффективный доступ из одного узла локальной сети к ресурсам, находящимся в других узлах. Развитие этой идеи приводит к функциональному разделению

компонентов сети: пользователь должен иметь не только доступ к ресурсами удаленного компьютера, но также получать от этого компьютера некоторый сервис, который специфичен для ресурсов данного рода и для обеспечения которого нецелесообразно дублировать в нескольких узлах соответствующие программные средства. Таким образом, приходят к дифференциации рабочих станций и серверов локальной сети.

Рабочая станция предназначена для непосредственной работы пользователя или категории пользователей и обладает ресурсами. соответствующими локальным потребностям данного пользователя. Специфическими особенностями рабочей станции могут быть небольшой объем оперативной памяти, наличие и объем дисковой памяти, характеристики процессора и монитора и т. д. При необходимости можно использовать ресурсы и услуги, предоставляемые сервером.

Сервер локальной сети должен обладать ресурсами, соответствующими его функциональному назначению и потребностям сети. Заметим, что в связи с ориентацией на подход открытых систем, правильнее говорить о логических серверах, имея в виду набор ресурсов и программных средств, обеспечивающих услуги над этими ресурсами, которые располагаются не обязательно на разных компьютерах. Особенностью логического сервера в открытой системе является то, что если по соображениям эффективности сервер целесообразно разместить на отдельном компьютере. не требуется какаялибо доработка использующих его прикладных программ и аппаратного обеспечения.

Серверами могут служить:

- вычислительный сервер, предоставляющий возможность производить вычисления, которые невозможно выполнить на рабочих станциях;
- сервер телекоммуникаций, обеспечивающий услуги по связи данной локальной сети с внешним миром;
- дисковый сервер, обладающий мощными ресурсами внешней памяти и предоставляющий их в использование рабочим станциями или другим серверам;

– файловый сервер, поддерживающий общее хранилище файлов для всех рабочих станций:
– сервер БД, фактически обычная СУБД. принимающая запросы по локальной сети и возвращающая результаты.

Очевидно, что для обеспечения взаимодействия прикладной программы, выполняемой на рабочей станции, с сервером потребуется специальный программный продукт, выполняющий такого рода функции. Из этого, собственно, и вытекают основные принципы системной архитектуры клиентсервер.

При этом вся система разбивается на две части, которые могут выполняться в разных узлах сети — клиентскую и серверную части. Прикладная программа или конечный пользователь взаимодействуют с клиентской частью системы, которая при необходимости обращается по сети к серверной части. Заметим, что в развитых системах сетевое обращение к серверной части может и не понадобиться, если система может предугадывать потребности пользователя и в клиентской части содержатся данные, способные удовлетворить его следующий запрос. Интерфейс серверной части определен и фиксирован, поэтому возможно создание новых клиентских частей уже существующей системы.

Основной проблемой систем, основанных на архитектуре клиент/сервер, является то, что в соответствии с концепцией открытых систем от них требуется мобильность в как можно более широком классе аппаратнопрограммных решений открытых систем, однако попытки создания систем, поддерживающих все возможные протоколы, приводят к их перегрузке сетевыми деталями в ущерб функциональности. Еще более сложным моментом является возможность использования разных представлений данных в разных узлах неоднородной локальной сети, т. к. в разных компьютерах может существовать различная адресация, представление чисел, кодировка символов и т. д.

Наиболее общим решением проблемы мобильности систем. основанных на архитектуре клиент/сервер, является опора на программные пакеты, реализующие протоколы удаленного вызова процедур. Использование таких средств для обращения к сервису в

удаленном узле выглядит как обычный вызов процедуры. тем самым скрывая от пользователя специфику сетевой среды и протоколов. При вызове удаленной процедуры такие программы производят преобразование форматов данных клиента в промежуточные машиннонезависимые форматы и затем преобразование в форматы данных сервера, а при передаче ответных параметров производятся аналогичные преобразования.

Термин сервер БД традиционно используют для обозначения всей СУБД, основанной на архитектуре клиент/сервер, включая как серверную, так и клиентскую части. Такие системы предназначены для хранения и обеспечения доступа к БД. Обычно одна БД целиком хранится в одном узле сети и поддерживается одним сервером, а серверы БД представляют собой некоторое приближение к распределенным БД, поскольку общие данные доступны для всех пользователей локальной сети.

Доступ к БД от прикладной программы или пользователя производится путем обращения к клиентской части системы. В качестве основного интерфейса между клиентской и серверной частями выступает язык баз данных SQL. который по сути представляет собой стандарт интерфейса СУБД в открытых системах. Собирательное название SQLсервер относится ко всем серверам БД основанных на SQL. Серверы БД, интерфейс которых основан на языке SQL, обладают своими преимуществами и своими недостатками.

Важным преимуществом является стандартность интерфейса. Вообще говоря, клиентские части любой SQLориентированной СУБД могли бы работать с любым SQLсервером вне зависимости от того, кто его произвел. Недостаток заключается в следующем: при таком уровне интерфейса между клиентской и серверной частями системы на стороне клиента работает слишком мало программ СУБД Это нормально, если на стороне клиента используется маломощная рабочая станция, но если клиентский компьютер обладает достаточной мощностью, то часто возникает желание возложить на него больше функций управления БД, разгрузив сервер, который, в свою очередь, является узким местом всей системы.

Одним из перспективных направлений СУБД является гибкое конфигурирование системы, при котором распределение функций

между клиентской и пользовательской частями СУБД определяется при установке системы.

Упоминавшиеся выше протоколы удаленного вызова процедур особенно важны в СУБД, основанных на архитектуре клиент/сервер. Это связано с тем. что использование механизма удаленных процедур позволяет действительно перераспределять функции между клиентской и серверной частями системы, т. к. в тексте программы удаленный вызов процедуры ничем не отличается от удаленного вызова, и, следовательно, любой компонент системы может располагаться как на стороне сервера, так и на стороне клиента.

Далее, механизм удаленного вызова скрывает различия между' взаимодействующими компьютерами. Физически неоднородная локальная сеть компьютеров приводится к логически однородной сети взаимодействующих программных компонентов, в результате чего пользователи не обязаны серьезно заботиться о разовой закупке совместимых серверов и рабочих станций.

В типичном на сегодняшний день случае на стороне клиента СУБД работает только такое программное обеспечение, которое не имеет непосредственного доступа к БД, а обращается для этого к серверу с использованием языка SQL. В некоторых случаях хотелось бы включить в состав клиентской части системы некоторые функции для работы с локальной частью БД, т.е. с тем ее фрагментом, который интенсивно используется клиентской прикладной программой. В современной технологии это можно сделать только путем формального создания на стороне клиента локальной копии сервера БД и рассмотрения всей системы как набора взаимодействующих серверов.

С другой стороны, иногда возникает необходимость в переносе большей части прикладной системы на сторону сервера, если разница в мощности клиентских рабочих станций и сервера чересчур велика. Сделать это нетрудно, однако требуется, чтобы базовое программное обеспечение сервера действительно позволяло выполнить такое перераспределение.

Из всего этого следует, что требования к аппаратуре и программному обеспечению клиентских и серверных компьютеров различаются в зависимости от вида использования системы. Если разделение между клиентом и сервером достаточно жесткое, то пользователям, работающим на рабочих станциях, абсолютно все равно, какая аппаратура и операционная система установлены на сервере — лишь бы

он справлялся с потоком запросов. Если же возникают потребности в перераспределении функций между клиентом и сервером, то возникает проблема, связанная с выбором аппаратуры и операционной системы, используемой в клиентской части.

Таким образом, рассмотрев такое направление развития СУБД, как распределенных систем, можно назвать его одним из самых перспективных.

Лекция 14. ХРАНИЛИЩА ДАННЫХ

Предположим, что в 1970 году была основана риэлтерская фирма. В СУБД, которой она пользуется, хранятся сведения об объектах недвижимости, клиентах, продажах и т.д. С течением времени БД разрастается и с одной стороны, устаревшие данные только замедляют работу, а с другой, они могут понадобится при принятии какогонибудь решения. Оптимальным выходом представляется выгрузка данных в архив и организация интерфейса для удобства работы с ним.

В этот момент возникает концепция хранилища данных. Эволюционно они дополняются наборами обобщенных статистических данных, с тем чтобы не генерировать их по запросам. Это вполне оправдано, т.к. выгруженные данные уже не изменяются.

Затем стало понятно, что кроме ответа на вопросы что, когда, кому и за сколько было продано или сколько заработали за 1981 год? хотелось бы получать ответы на более широкий круг вопросов. Нр,

1. Как война в Ираке 2003 года повлияет на рынок жилья в нашей области на основе данных по периоду «Бури в пустыне»?
2. Как менялась годовая выручка отделений компании в зависимости от профессионального состава персонала?
3. Какие типа объектов недвижимости продаются лучше/хуже при улучшении/ухудшении демографической ситуации, климата, смене правительства, изменении уровня жизни населения и т.д.
4. Готовится законопроект о повышении пенсий. Какую ценовую политику повести, чтобы успешно конкурировать на рынке на основании данных при аналогичных ситуациях?

Ответы на подобного типа вопросы можно получить только если организовать хранение соответствующей информации.

Т.о. на сегодняшний момент хранилище данных можно определить как:

Хранилище данных – предметноориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для под¬держки принятия решений.

В приведенном выше определении Инмона (Inmon, 1993) указанные характеристики данных понимаются следующим образом.

Предметная ориентированность. Хранилище данных организовано вокруг основных предметов (или субъектов) организации (например, клиенты, това¬ры и продажи), а не вокруг прикладных областей деятельности (выписка счета клиенту, контроль товарных запасов и продажа товаров). Это свойство отражает необходимость хранения данных, предназначенных для поддержки принятия решений, а не обычных оперативноприкладных данных.

Интегрированность. Смысл этой характеристики состоит в том, что оперативноприкладные данные обычно поступают из разных источников, которые часто имеют несогласованное представление одних и тех же данных, например, используют разный формат. Для предоставления пользователю единого обобщенного представления данных необходимо создать интегрированный источник, обеспечивающий согласованность хранимой информации.

Привязка ко времени. Данные в хранилище точны и корректны только в том случае, когда они привязаны к некоторому моменту или промежутку времени. Привязанность хранилища данных ко времени следует из большой длительно¬сти того периода, за который была накоплена сохраняемая в нем информация, из явной или неявной связи временных отметок со всеми сохраняемыми данными, а также из того факта, что хранимая информация фактически представляет собой набор моментальных снимков состояния данных.

Неизменяемость. Это означает, что данные не обновляются в оперативном режиме, а лишь регулярно пополняются за счет информации из оперативных систем обработки. При этом новые данные никогда не заменяют прежние, а лишь дополняют их. Таким образом, база данных хранилища постоянно пополняется новыми данными, последовательно интегрируемыми с уже накопленной информацией.

Существует достаточно много определений хранилищ данных, причем наиболее ранние определения в основном отражают характеристики информации, содержащейся в хранилище. Более поздние версии расширяют диапазон определения хранилища данных,

включая в него описание типа обработки данных, связанной с доступом к данным из исходных источников и далее вплоть до доставки данных ли ответственным за принятие решений (Anahory and Murray, 1997).

Каким бы ни было определение, конечной целью создания хранилища данных является интеграция корпоративных данных в едином репозитории, обращаясь которому пользователи смогут составлять запросы, генерировать отчеты и выполнять анализ данных. Хранилище данных является рабочей средой для систем поддержки принятия решений, которая извлекает данные, хранимые в различных оперативных источниках, организует их и передает лицам, ответственным за принятие решений в данной организации. Подводя итог, можно сказать, что технология хранилищ данных — это технология управления данными и их анализа.

Преимущества технологии хранилищ данных

При успешной реализации хранилища данных в организации могут быть достигнуты определенные преимущества, которые обсуждаются ниже.

Потенциально высокая отдача от инвестиций

В случае применения данной технологии организации потребуется инвестировать значительные средства для того, чтобы гарантировать успешную реализацию проекта. В зависимости от используемых технических решений необходимая сумма инвестиций может варьироваться от 50 000 до 10 000 000 фунтов стерлингов. Однако данным фирмы International Data Corporation (IDC) в 1996 году усредненная за 3 года прибыль на инвестированный капитал (ROIприбыль — Return On Investme) в сфере хранилищ данных составила 401%, причем более 90% фирм, охваченных данным исследованием, имели ROIприбыль свыше 40%, половина фирм — свыше 160%, а четверть фирм — свыше 600% (IDC, 1996).

Повышение конкурентоспособности

Огромные прибыли на инвестированный капитал фирм, которые успешно применили технологию хранилищ данных, стали доказательством существенного повышения конкурентоспособности,

которое явилось прямым следствием применения данной технологии. Повышение конкурентоспособности достигается за счет того, лица, ответственные за принятие решений в данной организации, получают доступ к ранее недоступной, неизвестной и никогда не использовавшейся информации, например, о клиентах, тенденциях рынка и спросе.

Повышение эффективности труда лиц, ответственный за принятие решений

Технология хранилищ данных повышает эффективность труда лиц, ответственных за принятие решений в данной организации, — за счет создания, интегрированной базы данных, состоящей из непротиворечивой, предметноориентированной и охватывающей обширный временной интервал информации. В этой базе данные, выбранные из нескольких, как правило, несовместимых между собой оперативных систем, интегрированы в форме, позволяющей получить единое, развернутое во времени представление о деятельности организации. Преобразуя исходные данные в осмысленную информацию, хранилище данных позволяет руководящему звену выполнять более содержательный, точный и согласованный анализ деятельности предприятия.

Сравнение OLTPсистем и хранилищ данных

СУБД, созданная для поддержки оперативной обработки транзакций (OLTP), обычно рассматривается как непригодная для организации хранилищ данных, поскольку к этим двум типам систем предъявляются совершенно разные требования. Например, OLTPсистемы проектируются с целью обеспечения максимально интенсивной обработки фиксированных транзакций, тогда как хранилища данных — прежде всего для обработки единичных произвольных запросов (ad hoc query). В таблице 1 сравниваются основные характеристики типичных OLTPсистем и хранилищ данных (Singh, 1997).

OLTP-система	Хранилище данных
Содержит текущие данные	Содержит исторические данные
Хранит подробные сведения	Хранит подробные сведения, а также в разной степени обобщенные данные
Данные являются динамическими	Данные в основном являются статическими
Повторяющийся способ обработки данных	Нерегламентированный, неструктурированный и эвристический способ обработки данных
Высокая интенсивность обработки транзакций	Средняя и низкая интенсивность обработки транзакций
Предсказуемый способ использования данных	Непредсказуемый способ использования данных
Предназначена для обработки транзакций	Предназначена для проведения анализа
Ориентирована на прикладные области	Ориентирована на предметные области
Поддержка принятия повседневных решений	Поддержка принятия стратегических решений
Обслуживает большое количество работников исполнительного звена	Обслуживает относительно малое количество работников руководящего звена

Организация обычно имеет несколько различных OLTPсистем, предназначенных для поддержки таких бизнеспроцессов, как контроль товарных запасов, выписка счетов клиентам, продажа товаров. Эти системы генерируют оперативные данные, которые являются очень подробными, текущими и подверженными изменениям. OLTPсистемы оптимизированы для интенсивной обработки транзакций, которые проектируются заранее, многократно повторяются и связаны преимущественно с обновлением данных. В соответствии с этими особенностями, данные в OLTPсистемах организованы согласно требованиям конкретных бизнесприложений и позволяют принимать

повседневные решения большому количеству параллельно работающих пользователейисполнителей.

В противоположность сказанному выше, в организации обычно имеется только одно хранилище данных, которое содержит исторические, подробные, обобщенные до определенной степени и практически неизменяемые данные (т.е. новые данные могут только добавляться). Хранилища данных предназначены для обработки относительно небольшого количества транзакций, которые имеют непредсказуемую природу и требуют ответа на произвольные, неструктурированные и эвристические запросы. Информация в хранилище данных организована в соответствии с требованиями возможных запросов и предназначена для поддержки принятия долговременных стратегических решений относительно небольшим количеством руководящих работников.

Хотя OLTPсистемы и хранилища данных имеют совершенно разные характеристики и создаются для различных целей, все же они тесно связаны в том смысле, что OLTPсистемы являются источником информации для хранилища данных. Основная проблема при организации этой связи заключается в том, что поступающие из OLTPсистем данные могут быть несогласованными, фрагментированными, подверженными изменениям, содержащими дубликаты или пропуски. Поэтому до помещения в хранилище данные должны быть "очищены".

OLTPсистемы не предназначены для получения быстрого ответа на произвольные запросы. Они также не используются для хранения устаревших исторических данных, которые требуются для анализа тенденций. OLTPсистемы, в основном, поставляют огромное количество сырых данных, которые не такто легко поддаются анализу. С помощью хранилищ данных можно получить ответы на запросы, более сложные, чем запросы с простейшими обобщениями типа следующего: "Какова средняя цена объектов недвижимости в крупнейших городах Великобритании?" Для хранилищ данных характерны совсем другие запросы, примеры которых приведены ниже.

– Какие типы объектов недвижимости продаются по ценам выше средней цены объектов недвижимости в крупнейших городах Великобритании и как эти данные коррелируют с демографическими данными?
– Какие три района в обслуживаемых городах были наиболее популярны с точки Зрения аренды объектов недвижимости в 1997 году и как эти данные связаны с данными за предыдущих два года?
– Какова месячная выручка от продажи объектов недвижимости в каждом отделении компании в сравнении с аналогичными показателями годичной давности?
– Какая связь наблюдается между ежегодной выручкой в каждом отделении компании и общим количеством торговых агентов в каждом из этих отделений?

Проблемы хранилищ данных

Ниже перечислены потенциальные проблемы, связанные с разработкой и сопровождением хранилищ данных (Greenfield, 1996).

– Недооценка ресурсов, необходимых для загрузки данных.
– Скрытые проблемы источников данных.
– Отсутствие требуемых данных в имеющихся архивах.
– Повышение требований конечных пользователей.
– Гомогенизация данных.
– Высокие требования к ресурсам.
– Владение данными. .,
– Сложное сопровождение.
– Долговременный характер проектов.
– Сложности интеграции.

Недооценка ресурсов, необходимых для загрузки данных

Многие разработчики склонны недооценивать время, необходимое для извлечения, очистки и загрузки данных в хранилище. На выполнение этого процесса может потребоваться до 80% общего времени разработки (Inmon, 1990), хотя эту долю можно существенно сократить при использовании более совершенных инструментов очистки и сопровождения данных.

Скрытые проблемы источников данных

Скрытые проблемы, связанные с источниками данных, поставляющими информацию в хранилище, могут быть обнаружены только спустя несколько лет после начала их эксплуатации. При этом разработчику придется принять решение об устранении возникших проблем в хранилище данных и/или в источниках данных. Например, при вводе данных о новом объекте недвижимости некоторые поля могут остаться незаполненными (NULL) в результате того, что сотрудник в свое время ввел в базу данных неполные сведения об этом объекте, невзирая на то, что они имелись в наличии.

Отсутствие требуемых данных в имеющихся архивах

В хранилищах данных часто возникает потребность получить некоторые сведения, которые не учитывались в оперативных системах, служащих источниками данных. В таком случае организация должна решить, стоит ей модифицировать существующие OLTPсистемы или же лучше создать новую систему по сбору недостающих данных. Может возникнуть необходимость анализа характеристик некоторых событий, например, регистрации новых клиентов и объектов недвижимости в каждом отделении компании. Однако в настоящее время это невозможно, поскольку для выполнения такого анализа недостает некоторых нужных сведений, например, даты регистрации объектов или клиентов.

Повышение требований конечных пользователей

После того как конечные пользователи получат в свое распоряжение инструменты составления запросов и отчетов, их требования к помощи и консультациям сотрудников информационной службы организации скорее возрастут, чем сократятся. Это вызвано тем, что пользователи хранилища данных начинают в большей степени сознавать истинные возможности и значение этого инструмента. Данную проблему ясно частично разрешить, используя менее мощные, но простые и удобные инструменты либо уделяя большее внимание обучению пользователей. Еще одной причиной увеличения нагрузки на сотрудников информационной службы организации является то, что сразу после запуска хранилища данных возрастает количество пользователей и запросов, причем сложность запросов также существенно возрастает.

Гомогенизация данных

Создание крупномасштабного хранилища данных может быть связано с решением серьезной задачи гомогенизации данных, что в итоге способно уменьшить ценность собранной информации. Например, при создании консолидированного и интегрированного представления данных организации разработчик хранилища данных может поддаться искушению подчеркнуть сходство, а не различие между данными, которые используются таких разных прикладных областях, как продажа и аренда объектов недвижимости.

Высокие требования к ресурсам

Для хранилища данных может потребоваться огромный объем дисковой памяти. Для многих реляционных систем поддержки принятия решений используются схемы типа "звезда", "снежинка" или "звездаснежинка". Все эти варианты приводят к созданию очень больших таблиц с фактическими данными (или таблиц фактов). При наличии множества размерностей фактических данных для хранения таблиц фактов вместе с итоговыми данными и индексами может потребоваться гораздо больше места, чем для хранения исходных необработанных данных.

Владение данными

Создание хранилища данных может потребовать изменить статус конечных пользователей в отношении прав владения данными. Наиболее критичные данные, которые ранее были доступны для просмотра и использования только отдельным подразделениями организации, занятым в определенных бизнессферах (например, продажа или маркетинг), теперь потребуется сделать доступными и другим сотрудникам организации.

Сложное сопровождение

Хранилища данных обычно характеризуются сложностью сопровождения, поскольку любая реорганизация бизнеспроцессов или источников данных может повлиять на происходящие в них процессы. Для того чтобы хранилище данных всегда оставалось ценным ресурсом, необходимо, чтобы оно постоянно полностью соответствовало организации, работу которой оно поддерживает.

Долговременный характер проектов

Хранилище данных представляет собой единый информационный ресурс организации. Однако для его создания может потребоваться до трех лет, а потому многие организации строят также свои собственные магазины данных. Магазины данных (data marts) предназначены для поддержки работы только какогото одного подразделения организации или одной ее прикладной области, а потому создать их можно гораздо быстрее.

Сложности интеграции

Наиболее важной частью процесса сопровождения хранилища данных являются его интеграционные возможности. Это значит, что организация должна потратить значительное количество времени для того, чтобы определить, насколько хорошо могут интегрироваться различные инструменты хранилища данных для получения искомого общего решения. Это довольно трудная задача, потому что для выполнения различных операций с хранилищем данных могут использоваться самые разные инструменты, которые должны слаженно совместно работать на пользу всей организации в целом.

Архитектура хранилища данных

Исходные данные, помещаемые в хранилище, поступают из следующих источников.

- Оперативные данные мейнфреймов, содержащиеся в иерархических и сетевых базах данных первого поколения. По некоторым оценкам, большинство оперативных корпоративных данных хранится в системах этого типа.
- Данные различных подразделений, сохраняемые в специализированных файловых системах типа VSAM, RMS и базах данных таких реляционных СУБД, как Informix и Oracle.
- Закрытые данные, которые хранятся на рабочих станциях и закрытых серверах.
- Внешние системы, например, Internet, коммерчески доступные базы данных или базы данных, принадлежащие поставщикам или клиентами организации.

Менеджер загрузки

Менеджер загрузки (load manager), который часто называют внешним (frontend) компонентом, выполняет все операции, связанные с извлечением и загрузкой данных в хранилище. Эти операции включают простые преобразования данных, необходимые для их подготовки к вводу в хранилище. Размеры и сложность данного компонента могут варьироваться в значительной степени, поскольку в его состав обычно входят не только программы собственной разработки, но и инструменты, созданные сторонними поставщиками.

Менеджер хранилища

Менеджер хранилища (warehouse manager) выполняет все операции, связанные с правлением информацией, помещенной в хранилище данных. Этот компонент также может включать программы собственной разработки и инструменты, предоставленные сторонними фирмами. Менеджер хранилища выполняет такие операции, как:

- анализ непротиворечивости данных;
- преобразование и перемещение исходных данных из временного хранилища в основные таблицы хранилища данных;
- создание индексов и представлений для базовых таблиц;
- денормализация данных (в случае необходимости);
- обобщение данных (в случае необходимости);
- резервное копирование и архивирование данных.

В некоторых случаях менеджер хранилища также генерирует профили запросов ля определения необходимых индексов и требуемых предварительных обобщений данных. Профиль запроса может создаваться для отдельного пользователя, группы пользователей или хранилища данных в целом. Он строится на основе информации, описывающей такие характеристики запросов, как частота выполнения, набор используемых таблиц и размер результирующего набора данных.

Менеджер запросов

Менеджер запросов (query manager), который часто называют внутренним (backend) компонентом, выполняет все операции, связанные с управлением пользовательскими запросами. Этот компонент обычно создается на базе предоставляемых разработчиком СУБД инструментов доступа к данным, инструментов мониторинга хранилища и программ собственной разработки, использующих весь набор функциональных возможностей СУБД. Сложность менеджера запросов определяется функциональными возможностями, предоставляемыми инструментами доступа к данным и самой СУБД. К числу выполняемых этим компонентом операций относятся управление запросами к соответствующим таблицам и составление графиков выполнения этих запросов.

В некоторых случаях менеджер запросов также генерирует профили запросов, позволяющие менеджеру хранилища определить набор необходимых индексов и обобщенных данных.

Детальные данные

В этой части хранилища данных хранятся все детальные данные, описанные в схеме базы данных. В большинстве случаев детальные данные хранятся не на оперативном уровне, а в виде информации, обобщенной до следующего уровня детализации. Как правило, детальные данные периодически добавляются в хранилище с автоматическим выполнением обобщения исходной информации до необходимого уровня.

Частично и глубоко обобщенные данные

В этой области хранилища размещаются все данные, предварительно обработанные менеджером хранилища с целью их частичного или глубокого обобщения (aggregate). Эта часть хранилища данных является временной, поскольку она постоянно подвергается изменениям в ответ на изменения профилей запросов.

Назначение обобщенных данных состоит в повышении производительности запросов. Хотя предварительное обобщение информации связано с некоторым повышением расходов на обслуживание, однако эти дополнительные затраты компенсируются за счет исключения необходимости многократно выполнять

обобщающие операции (например, сортировку или группирование) при обработке каждого из запросов пользователей. Хранимые обобщенные данные обновляются по мере загрузки новых порций детальных данных в хранилище.

Архивные и резервные копии

Этот компонент хранилища данных отвечает за подготовку детальной и обобщенной информации к помещению в резервные и архивные копии. Хотя обобщенные данные генерируются на основе детальных, может потребоваться помещать в резервную копию заранее обобщенные данные, если предполагаемый период их хранения превышает срок хранения тех детальных данных, на основе которых они были созданы. Как правило, резервные и архивные копии размещаются на таких носителях, как магнитная лента или оптический диск.

Метаданные

В этой области хранилища данных хранятся все те метаданные (данные про данные), которые используются любыми процессами хранилища. Метаданные могут применяться для разных целей, включая перечисленные ниже.

- Извлечение и загрузка данных. Метаданные используются для отображения источников данных на общее представление информации внутри хранилища.
- Обслуживание хранилища. Метаданные применяются для автоматизации подготовки таблиц с обобщенными значениями.
- Часть процесса обслуживания запросов. Метаданные используются для направления запроса к наиболее подходящему источнику данных.

Структура метаданных может отличаться для разных процессов, в зависимости от их назначения. Это значит, что для одного и того же элемента данных в хранилище может храниться сразу несколько вариантов метаданных. Кроме того, большинство фирмразработчиков применяет собственные версии структуры метаданных в своих инструментах управления копированием данных и средствах доступа, предназначенных для конечных пользователей. В частности, инструменты управления копированием данных используют метаданные для определения правил отображения, которые

необходимо применить для преобразования исходных данных в общепринятую форму. Средства доступа конечных пользователей к данным используют метаданные для выбора способа построения запроса. Управление метаданными внутри хранилища данных является очень сложной задачей, которую не следует недооценивать.

Средства доступа к данным конечного пользователя

Основным назначением хранилища данных является предоставление конечным пользователям информации, необходимой им для принятия стратегических решений. Пользователи взаимодействуют с хранилищем с помощью специальных инструментов доступа к данным. Само хранилище данных должно обеспечивать эффективное выполнение произвольных запросов и предоставлять средства проведения анализа. Высокая производительность хранилища данных достигается за счет тщательного предварительного планирования операций соединения, обобщения и составления периодических отчетов, которые могут потребоваться конечным пользователям.

Хотя определения пользовательских инструментов доступа к данным могут перекрываться, но в данном контексте мы разобьем их на пять основных групп (Berson and Smith, 1997);

- инструменты создания отчетов и запросов;
- инструменты разработки приложений;
- инструменты информационной системы руководителя (Executive Information System — EIS);
- инструменты оперативной аналитической обработки (OLAPинструменты):
- инструменты разработки данных.

Инструменты создания отчетов и запросов

Инструменты создания отчетов подразделяются на инструменты создания итоговых отчетов и редакторы отчетов. Инструменты создания итоговых отчетов используются для создания регулярных оперативных отчетов и для подготовки таких объемных пакетных заданий, как выписка заказов/счетов для клиентов или составление платежных ведомостей для выдачи зарплаты сотрудникам. Редакторы

отчетов — это недорогие настольные инструменты, предназначенные для нужд конечных пользователей.

Инструменты создания запросов в реляционных СУБД служат для ввода или генерации SQLкоманд, используемых для извлечения данных из хранилища. Подобные инструменты обычно скрывают от конечных пользователей сложность операторов языка SQL и структур баз данных — за счет вставки между пользователем и базой данных промежуточного метауровня. Метауровень — это программное обеспечение, которое предоставляет пользователю некоторое предметноориентированное представление содержимого базы данных и позволяет генерировать SQLоператоры С помощью визуальных инструментов типа "выбери и щелкни". Примером подобного инструмента создания запросов является язык QueryByExample (QBE). Различные инструменты создания запросов весьма популярны среди пользователей бизнесприложений и могут применяться ими для любых целей, например, для выполнения демографического анализа или подготовки почтовых списков рассылки клиентов организации. Однако по мере усложнения запросов такие инструменты очень быстро становятся неэффективными.

Инструменты разработки приложений

Требования конечных пользователей могут быть такими, что встроенные средства создания отчетов и инструменты создания запросов окажутся неадекватными — либо изза того, что требуемый анализ не может быть ими выполнен в принципе, либо изза того, что пользователю потребуется иметь чрезвычайно высокую квалификацию и немалый опыт. В такой ситуации для обеспечения доступа пользователей к требуемой информации потребуется разработка собственных приложений с использованием графических сред доступа к данным, предназначенных для использования в системах архитектуры "клиент/сервер". Некоторые из этих платформ разработки приложений интегрируются с популярными OLAPинструментами. Они позволяют осуществлять доступ ко всем основным типам СУБД, включая Oracle, Sybase и Informix. Примерами подобных платформ разработки приложений являются PowerBuilder фирмы PowerSoft,

Visual Basic фирмы Microsoft, Forte фирмы Forte Software и Business Objects фирмы Business Objects.

Инструменты информационной системы руководителя

Информационные системы руководителя (Executive Information System — EIS), которые в последнее время стали называть "информационными системами для всех" (Everybody Information System — EIS), первоначально разрабатывались для поддержки принятия высокоуровневых стратегических решений. Однако впоследствии область применения этих систем была несколько расширена с целью предоставления поддержки управляющему персоналу всех уровней. На начальном этапе EISинструменты были связаны с мейнфреймами и позволяли их пользователям создавать собственные приложения с графическим интерфейсом, предназначенные для поддержки принятия решений за счет создания обобщающего представления о данных организации и предоставления доступа к внешним источникам данных.

На современном рынке граница между EISинструментами и другими инструментами поддержки принятия решений стала еще более размытой, так как разработчики EISинструментов добавляют в свои продукты средства создания запросов и предоставляют специализированные приложения для таких областей бизнеса, как торговля, маркетинг и финансы. Примерами EISинструментов являются продукты Lightship фирмы Pilot Software, Forest and Trees фирмы Platinum Technologies и Express Analyzer фирмы Oracle.

OLAP-инструменты

Инструменты оперативной аналитической обработки данных, или OLAPинструменты, создаются на основе концепции многомерной базы данных. Они позволяют квалифицированным пользователям анализировать данные с помощью сложных многомерных представлений. Типичные бизнесприложения для этих инструментов включают оценку эффективности маркетинговой кампании, предсказание объемов продаж и планирование товарных запасов. При использовании подобных инструментов предполагается, что данные организованы согласно многомерной модели, которая поддерживается специальной многомерной базой данных (MultiDimensional Database —

MDDB) или реляционной базой данных, предназначенной для работы с многомерными запросами.

Инструменты разработки данных

Разработка данных — это процесс открытия новых осмысленных корреляций, распределений и тенденций путем переработки огромного количества информации, извлеченной из хранилища или магазина данных, с использованием статистических и математических методов, а также методов искусственного интеллекта. Методы разработки данных обладают достаточным потенциалом, чтобы превзойти возможности OLAPинструментов, так как главным притягательным фактором использования технологии разработки данных является способность создавать предсказательные, а не ретроспективные модели.

Информационные потоки в хранилище данных

В технологии хранилищ данных основное внимание уделяется управлению пятью основными информационными потоками: входным, восходящим, нисходящим, выходным и метапотоком (Hackathorn, 1995).

С каждым из этих потоков связаны определенные процессы, которые представлены ниже.

Входной поток – извлечение, очистка и загрузка исходных данных.

Восходящий поток – повышение ценности сохраняемых в хранилище данных путем обобщения, упаковки и распределения исходных данных.

Нисходящий поток – архивирование и резервное копирование информации в хранилище.

Выходной поток – предоставление данных пользователям.

Метапоток – управлением метаданными.

Входной поток

Входной поток – процессы, связанные с извлечением, очисткой и загрузкой информации из источников данных в хранилище данных.

Входной поток связан с выборкой информации из источников данных с целью их последующей загрузки в хранилище данных.

Поскольку исходные данные генерируются преимущественно OLTPсистемами, эти данные должны быть перестроены в соответствии с требованиями хранилища данных. Перестройка данных включает такие операции, как:

- очистка данных;
- преобразование данных в соответствии с требованиями хранилища данных, включая добавление и/или удаление полей и денормализацию данных;
- проверка внутренней непротиворечивости данных и их непротиворечивости по отношению к данным, уже загруженным в хранилище.

Для эффективного управления входным потоком необходимо подобрать механизм определения момента начала извлечения данных с последующими выполнением требуемых преобразований и проверкой непротиворечивости данных. Для создания единого непротиворечивого представления корпоративных данных очень важно в процессе извлечения информации из источников убедиться в том, что она находится в согласованном состоянии. Сложность процесса извлечения информации зависит от степени взаимной согласованности между различными источниками данных.

Непосредственно после извлечения из источника данные обычно загружаются во временное хранилище с целью выполнения очистки и проверки непротиворечивости. Поскольку этот процесс достаточно сложен, важно, чтобы он был полностью автоматизирован и сопровождался выдачей сообщений о любых возникающих проблемах или сбоях. Для обслуживания входного потока на рынке предлагаются специальные коммерческие инструменты. Однако если требуемая процедура не является тривиальной, то в случае использования коммерческих инструментов потребуется выполнить их предварительную настройку.

Восходящий поток

Восходящий поток – процессы, связанные с повышением ценности сохраняемых в хранилище данных посредством обобщения, упаковки и распределения исходных данных.

Обслуживание восходящего потока включает выполнение приведенных ниже действий.

- Обобщение данных посредством операций выборки, проекции, соединения и группирования связанных данных, выполняемое для получения более удобных и полезных для пользователей представлений информации. Обобщение может включать выполнение не только простых реляционных операций, но и проведение сложного статистического анализа, включая вычисление трендов, кластеризацию и подбор типичных значений.
- Упаковка данных с преобразованием подробных исходных или обобщенных данных в более удобный формат представления, например, в виде электронных таблиц, текстовых документов, диаграмм, других графических представлений, закрытых баз данных и анимированных материалов.
- Распределение исходных данных на соответствующие группы для повышения их подготовленности к использованию и доступности.

Параллельно с планированием повышения ценности данных следует учитывать и требования увеличения общей производительности хранилища, а также снижения текущих расходов на его сопровождение. Все эти требования противоречат друг другу, что вынуждает разработчиков либо повышать производительность обработки запросов, либо сокращать расходы на сопровождение. Иначе говоря, администратор хранилища данных должен разработать проект базы данных, наиболее подходящий для удовлетворения всех существующих требований, для чего зачастую приходится идти на компромисс.

Нисходящий поток

Нисходящий поток – процессы, связанные с архивированием и резервным копированием информации в хранилище данных.

Архивирование устаревших данных играет важную роль при обеспечении высокой эффективности и производительности хранилища данных за счет переноса устаревших данных с ограниченной ценностью на архивный носитель, например, на магнитную ленту или оптические диски. Однако если схема секционирования базы данных выполнена правильно, то общее

количество оперативных данных не должно влиять на производительность хранилища данных.

Нисходящий поток информации включает процедуры, обеспечивающие возможность восстановления текущего состояния хранилища в случае потери данных изза Сбоев в программном или аппаратном обеспечении. Архивные данные следует хранить таким образом, чтобы в случае необходимости они снова могли быть восстановлены в хранилище данных.

Выходной поток

Выходной поток – процессы, связанные с предоставлением данных пользователям.

Именно благодаря выходному потоку информации у сотрудников организации создается представление об истинной ценности хранилища данных. Полученные данные могут потребовать перестройки всех бизнеспроцессов организации с целью повышения ее конкурентоспособности (Hackathorn, 1995).

В качестве основных действий, связанных с выходным потоком, следует упомянуть следующие.

- Доступ к данным означает удовлетворение запросов конечных пользователей в отношении нужных им данных. Главное здесь заключается в создании такой среды, в которой пользователи смогли бы эффективно использовать инструменты создания запросов для получения доступа к наиболее подходящему источнику данных. Частота выполнения отдельных запросов пользователей может варьироваться от однократно выполняемого произвольного запроса до регулярно выполняемых запросов или даже запросов, выполняемых по установленному графику. При разработке графика выполнения запросов пользователей очень важно обеспечить использование системных ресурсов максимально эффективным образом.
- Доставка означает предварительную доставку информации на рабочие станции конечных пользователей. Это относительно новая область обработки та формации в хранилищах данных, связанная с процессами типа публикации/подписки. Хранилище данных публикует различные бизнесобъекты, которые периодически подвергаются ревизии в соответствии с установленными шаблонами использования. Пользователи могут

подписаться на такой набор бизнесобъектов, который наилучшим образом отвечает их потребностям.

Важной особенностью управления выходным потоком является активная популяризация хранилища данных среди пользователей, что может оказать существенно влияние на функционирование всей организации. В числе дополнительных действий по обслуживанию выходного потока следует также назвать направление запросов к соответствующим целевым таблицам и обработку профилей запросов конкретных групп пользователей с целью выяснения, какие именно обобщения данных могу оказаться полезными.

Хранилища данных, содержащие обобщенные данные, потенциально предоставляют пользователям большее количество различных источников данных, способных дать ответ на их конкретные запросы. Сюда относятся как собственно исходные детальные сведения, так и произвольное количество выполненных на их основе обобщений, способных удовлетворить информационные потребности запросов пользователей. Однако производительность запроса может в значительной степени варьироваться — в зависимости от характеристик обрабатываемых данных. Наиболее очевидной такой характеристикой является объем считываемой информации. В состав действий по обслуживанию выходного потока входит и поиск системой наиболее эффективного способа выполнения запроса.

Метапоток

Метапоток – процессы, связанные с управлением метаданными.

Предыдущие потоки характеризуют управление хранилищем данных в отношении перемещения данных в хранилище и из него. Метапоток — это процесс, связанный с перемещением метаданных, т.е. данных о других потоках. Метаданные — это описание информационного содержания хранилища данных: что в нем содержится, откуда что поступает, какие операции выполнялись во время очистки, как осуществлялись интеграция и обобщение.

В ответ на изменение бизнеспотребностей постепенно менялись и продолжают изменяться любые традиционные унаследованные системы. Поэтому при управлении хранилищем данных необходимо учитывать эти продолжающиеся изменения, т.е. принимать во

внимание изменения традиционных источников данных и бизнессреды. Таким образом, метапоток (метаданные) должен постоянно обновляться в соответствии с происходящими изменениями.

Инструменты и технологии хранилищ данных

Создание полностью интегрированного хранилища данных является чрезвычайно сложной задачей, поскольку нет такого разработчика, который смог бы предложить весь необходимый набор инструментов, — от начала и до конца. Поэтому создавать хранилища данных обычно приходится с использованием нескольких продуктов разных поставщиков. Полная интеграция и организация совместной работы используемых инструментов является чрезвычайно важной и сложной задачей. В этом разделе рассматриваются инструменты и технологии, используемые при создании и сопровождении хранилищ данных, а также уделяется необходимое внимание вопросам интеграции этих инструментов. Более подробную информацию об используемых в хранилищах данных инструментах и технологиях можно найти в работе Берсона и Смита (Berson and Smith, 1997).

Инструменты извлечения, очистки и преобразования данных

Выбор оптимальных инструментов извлечения, очистки и преобразования данных очень важен для успешного создания хранилища данных. Количество фирмразработчиков, основное внимание которых обращено в сторону удовлетворения требований, предъявляемых к хранилищам данных, постоянно растет. Предлагаемые ими инструменты могут решать гораздо более сложные задачи, чем простое перемещения данных между разными аппаратными платформами. При этом задачи извлечения данных из источника, их очистки и преобразования с последующей загрузкой в конкретную систему могут быть выполнены либо с помощью нескольких разных программных продуктов, либо посредством применения единого интегрированного подхода. Подобные интегрированные решения делятся на следующие категории:

- генераторы кода;
- инструменты репликации информации базы данных;
- механизмы динамического преобразования.

Генераторы кода

Генераторы кода создают настраиваемые преобразующие программы на языках третьего или четвертого поколения исходя из предоставленных им определений форматов входных и выходных данных. Основной проблемой этого подхода является необходимость управления большим количеством программ, которые потребуются для поддержки сложного корпоративного хранилища данных. Фирмыразработчики признают наличие указанной проблемы, поэтому некоторые из них, применив такие методы, как конвейерная обработка и автоматическая разработка графиков, создали специальные управляющие компоненты.

Инструменты репликации информации базы данных

Данные инструменты используют триггеры баз данных или журналы регистрации транзакций для обнаружения изменений отдельных элементов данных в одной системе и переноса этих изменения в копию исходных данных, размещенную в другой системе. Большинство инструментов репликации не поддерживает обнаружения изменений в нереляционных файлах и базах данных и зачастую не имеет инструментов для значительного преобразования и очистки данных. Эти инструменты могут быть использованы для восстановления базы данных после сбоя или создания базы данных для магазина данных, но при условии, что количество источников данных невелико, а требуемое преобразование данных имеет относительно простой характер.

Механизмы динамического преобразования

Исходя из установленных правил, механизмы динамического преобразования извлекают данные из источника через определенные пользователем интервалы времени, преобразуют их, а затем отсылают и загружают обработанную информацию в указанное место назначения. На сегодняшний день большинство программных продуктов поддерживает только реляционные источники данных, однако в последнее время стали появляться продукты, способные работать и с нереляционными исходными файлами и базами данных.

Примерами инструментов преобразования данных являются пакеты Enterprise/ Access фирмы Apertus Corporation, Warehouse Manager

фирмы Prism Solutions, PASSPORT и Metacenter фирмы Carleton Corporation, ETIEXTRACT фирмы Evolutionary Technologies Inc., Powermart Suite фирмы Informatica.

СУБД для хранилища данных

СУБД для хранилищ данных очень редко бывает источником проблем интеграции. Благодаря относительной зрелости таких программных продуктов, большинство реляционных баз данных интегрируется с другими типами программного обеспечение вполне предсказуемым образом. Однако потенциальным источником проблем может послужить большой размер базы данных хранилища. При работе с подобной базой данных становится особенно важным обеспечение параллельности, а также таких традиционно важных параметров, как высокая производительность, масштабируемость, готовность и управляемость, что обязательно следует принимать во внимание при выборе СУБД.

Сначала мы рассмотрим основные требования, предъявляемые к СУБД для хранилища данных, а затем кратко обсудим, как можно организовать в хранилищах данных параллельное выполнение вычислений.

Требования к СУБД для хранилища данных

Специализированные требования к реляционной СУБД (РСУБД), предназначенной для хранилища данных, были опубликованы в документе White Paper (Red Brick Systems, 1996). Вот эти требования.

- Высокая производительность загрузки данных.
- Возможность обработки данных во время загрузки.
- Наличие средств управления качеством данных.
- Высокая производительность запросов.
- Широкая масштабируемость по размеру (до терабайт).
- Масштабируемость по количеству пользователей.
- Возможность организации сети хранилищ данных.
- Наличие средств администрирования хранилища.
- Поддержка интегрированного многомерного анализа.
- Расширенный набор функциональных средств запросов

Высокая производительность загрузки данных

В хранилищах данных требуется периодически выполнять загрузку порций новых данных, причем в ограниченных временных рамках. Производительность процесса загрузки в подобных случаях должна измеряться в сотнях миллионов строк или гигабайтах данных в час. Со стороны бизнесзадач не существует никаких ограничений в отношении максимально допустимого уровня производительности.

Возможность обработки данных во время загрузки

При загрузке в хранилище новых или обновленных данных обычно требуется выполнение нескольких последовательных этапов, включающих преобразование данных, фильтрование, переформатирование, проверку целостности, физическое сохранение, индексирование и обновление метаданных. На практике каждый такой этап может выполняться по отдельности, однако в общем процесс загрузки должен выглядеть как единая неразрывная процедура.

Наличие средств управления качеством данных

Для перехода к управлению на основе фактической информации требуются данные высочайшего качества. В хранилище данных должна гарантироваться локальная непротиворечивость данных, глобальная непротиворечивость данных, а также целостность данных на уровне ссылок, даже несмотря на использование "грязных" источников данных и громадные размеры базы данных. Хотя загрузка и подготовка данных — необходимые шаги, они все же не являются достаточными. Лишь способность дать ответы на запросы конечных пользователей является действительной мерой успешности создания хранилища данных. Анализ утверждает, что, чем больше ответов было успешно предоставлено пользователям, тем сложнее, и изощреннее становятся очередные вводимые ими запросы.

Высокая производительность запросов

Управление на основе фактической информации и произвольный анализ не должны замедляться или стопориться изза низкой производительности обработки запросов со стороны СУБД хранилища данных. Большие комплексные запросы для ключевых бизнесопераций должны завершаться за приемлемое время.

Широкая масштабируемость по размеру

Размеры хранилищ данных возрастают с огромной скоростью и достигают величин от сотен гигабайт до терабайт (1012 байт) и даже петабайт (1015 байт). Используемая РСУБД не должна иметь никаких архитектурных ограничений на размер базы данных и должна поддерживать модульное и параллельное управление. В случае сбоя РСУБД должна сохранять готовность к работе и предоставлять механизм восстановления до исходного состояния. РСУБД должна поддерживать работу с устройствами массовой памяти, например, оптические диски или иерархические устройства хранения. Наконец, производительность выполнения запросов должна зависеть не от размера базы данных, а прежде всего от сложности самого запроса.

Масштабируемость по количеству пользователей

В настоящее время считается, что доступ к хранилищу данных будет ограничен только относительно небольшим кругом менеджеров. Однако маловероятно, что такая тенденция сохранится и при возрастании значения хранилищ данных. По некоторым оценкам, в недалеком будущем РСУБД для хранилищ данных должны будут поддерживать работу сотен и даже тысяч параллельно работающих пользователей, обеспечивая при этом приемлемую производительность выполнения их запросов.

Возможность организации сети хранилищ данных

Хранилище данных должно обладать способностью работать в большой сети, состоящей из многих хранилищ данных. Хранилище данных должно включать инструменты, которые координировали бы перемещение подмножеств данных между отдельными хранилищами. На своей рабочей станции пользователи должны иметь возможность просматривать и работать с содержимым нескольких хранилищ данных.

Наличие средств администрирования хранилища

Исключительно большой размер и цикличная природа хранилищ данных требует наличия простых и в то же время гибких инструментов администрирования. РСУБД должна предоставлять средства управления для ограничения ресурсов, подсчета накладных расходов для всех пользователей, а также систему установки приоритетов

выполнения запросов для удовлетворения потребностей различных категорий пользователей и видов деятельности. РСУБД должна также иметь средства для отслеживания и настройки режимов рабочей нагрузки, необходимых для оптимизации производительности и пропускной способности системы. Наиболее очевидной и ощутимой ценностью реализации хранилища данных является беспрепятственный творческий доступ конечных пользователей к данным.

Поддержка многомерного интегрированного анализа

Ценность многомерных представлений — общепризнанный факт, а потому поддержка работы с ними непременно должна быть предусмотрена в РСУБД, используемой для организации хранилища данных, поскольку это является условием для обеспечения максимальной производительности реляционных OLAPинструментов. РСУБД должна поддерживать быстрое и простое создание предварительно подготовленных итоговых значений для больших хранилищ данных, а также предоставлять инструменты для автоматизации процесса создания этих предварительно вычисленных обобщений. Динамическое вычисление обобщенных значении должно быть согласовано с требованиями обеспечения необходимого уровня производительности интерактивной работы пользователей.

Расширенный набор функциональных средств запросов

Конечным пользователям необходимо иметь возможность выполнять аналитические вычисления, последовательный и сравнительный анализ, согласованный доступ к детальным и обобщенным данным. Использование языка SQL в клиент/серверной среде создания запросов по типу "выбери и щелкни" иногда может оказаться непрактичным или даже просто невозможным изза высокой сложности пользовательских запросов. В РСУБД должен быть предусмотрен полный набор всех необходимых аналитических инструментов.

Параллельные СУБД

При работе с хранилищем данных обычно требуется обработать огромное количество данных, а технология параллельной работы с базами данных предлагает эффективное решение для обеспечения необходимого роста производительности. Успешность работы параллельных СУБД зависит от эффективного управления многими ресурсами, например, таким, как процессоры, память, диски и сетевые подключения.

По мере возрастания популярности хранилищ данных фирмыразработчики создают большие СУБД, предназначенные для систем поддержки принятия решений и использующие технологию организации параллельных вычислений. Основная цель состоит в решении поставленных пользователем задач с использованием нескольких узлов, параллельно работающих над одной и той же проблемой. Важнейшими характеристиками параллельных СУБД являются масштабируемость, оперативность и готовность.

Параллельные СУБД могут одновременно выполнять сразу несколько операций с базой данных, разбивая отдельные задачи на малые части так, чтобы они могли быть распределены между несколькими процессорами. Параллельные СУБД должны "уметь выполнять параллельные запросы. Иначе говоря, они должны уметь разбивать большие сложные запросы на подзапросы, параллельно запускать отдельные подзапросы на выполнение, а затем собирать вместе полученные результаты. Такие СУБД должны уметь выполнять в параллельном режиме загрузку данных, сканирование таблицы, а также архивирование и резервное копирование данных.

Существуют две основные архитектуры аппаратного обеспечения для выполнения параллельных вычислений, которые могут использоваться в качестве платформы для сервера базы данных в хранилищах данных.

– Симметричная мультипроцессорная обработка (Symmetric Multiprocessing — SMP) — это группа тесно связанных процессоров, которые совместно используют оперативную и дисковую память.

– Массовая мультипроцессорная обработка (Massively MultiProcessing — MMP) — это группа слабо связанных процессоров, каждый из которых использует свою собственную оперативную и дисковую память.

Метаданные хранилища данных

Существует много других вопросов, связанных с интеграцией программного обеспечения хранилищ данных, однако в этом разделе мы рассмотрим только проблемы, касающиеся интеграции метаданных, т.е. данных о данных (Darling, 1996). Управление метаданными в хранилище данных представляет собой чрезвычайно сложную и многогранную задачу. Метаданные используются в самых разных целях, а управление метаданными является важнейшим вопросом при создании полностью интегрированного хранилища данных.

Основным назначением метаданных является сохранение информации о происхождении данных, с той целью, чтобы администраторы хранилища могли знать историю любого элемента данных, помещенного в хранилище. Однако проблема состоит в том, что метаданные в пределах хранилища данных обладают несколькими функциями, связанными с преобразованием и загрузкой данных, обслуживанием хранилища данных и генерацией запросов.

Метаданные, связанные с преобразованием и загрузкой данных, должны описывать источник данных и любые изменения, внесенные в эти данные. Например, для каждого исходного поля должны храниться такие сведения, как уникальный идентификатор, исходное имя поля, тип источника данных, исходное расположение, включая системное и объектное имя, а также тип преобразованных данных и имя таблицы назначения. Если поле подвергается какимто изменениям, начиная с простого изменения его типа и вплоть до выполнения над ним сложного набора процедур и функций, то все эти изменения также должны быть записаны.

Метаданные, связанные с управлением данными, описывают способ хранения данных в хранилище. Каждый объект базы данных должен быть описан, включая данные, содержащиеся во всех таблицах, индексах и представлениях. Кроме того, должны быть описаны и

любые связанные с этими объектами ограничения. Данная информация хранится в системном каталоге СУБД, но с учетом некоторых дополнительных ограничений, присущих хранилищам данных. Например, метаданные должны описывать любые поля, связанные с обобщенными данными (т.е. итоговыми значениями), включая способ обобщения (т.е. использованную для этой цели функцию). Кроме того, должно быть описано секционирование таблиц, в том числе ключ секционирования, а также диапазон данных, связанных с каждой секцией.

Описанные выше метаданные требуются также менеджеру запросов для генерации соответствующих запросов. Менеджер запросов генерирует дополнительные метаданные о выполняемых запросах, которые могут быть использованы для генерации исторических данных обо всех выполненных запросах и их профилях для каждого пользователя, группы пользователей или хранилища данных в целом. Существуют— также метаданные, связанные с пользователями запросов, которые включают, например, информацию, описывающую значение терминов "цена" и "клиент" для некоторой базы данных с учетом изменения их смысла с течением времени.

Синхронизация метаданных

Важнейшим вопросом интеграции является синхронизация метаданных разного типа, используемых в пределах всего хранилища данных. Поскольку разные инструменты хранилища данных создают и используют свои собственные метаданные, для достижения полной интеграции требуется сделать так, чтобы эти инструменты могли пользоваться метаданными совместно. Задача в этом случае заключается в синхронизации метаданных между разными программными продуктами различных фирмразработчиков, использующих разные хранилища метаданных. Например, необходимо идентифицировать нужный элемент метаданных на соответствующем уровне детализации одного программного продукта, потом установить его соответствие другому элементу метаданных на соответствующем уровне детализации другого программного продукта, а затем отрегулировать любые существующие между ними различия в способе кодирования. Эту операцию следует повторить для всех других

метаданных, общих для двух программных продуктов. Более того, любые изменения метаданных или даже "метаметаданных" в одном программном продукте должны быть переданы другому программному продукту. Задача синхронизации двух программных продуктов очень сложна, а потому повторение этого процесса для шести или более продуктов, которые образуют программное обеспечение хранилища данных, может потребовать очень большого расхода ресурсов. Тем не менее синхронизация метаданных обязательно должна быть выполнена. Решить данную проблему можно двумя способами:

- с помощью механизмов автоматической передачи метаданных между хранилищами метаданных разных инструментов;
- путем создания репозитория метаданных.

В настоящее время ведутся исследования возможных механизмов передачи метаданных. Ранние продукты, построенные на передаче метаданных, имели очень ограниченный успех изза отсутствия стандарта обмена метаданными. В 1995 году был образован комитет Metadata Coalition Committee, который разработал спецификацию Metadata Interchange Format (MIF), предназначенную для стандартизации процесса передачи метаданных. В настоящее время она используется фирмамиразработчиками для создания инструментов обмена метаданными. Несмотря на полезность этих продуктов, все же лучше для этой цели использовать репозиторий метаданных.

Репозиторий метаданных объединяет различные типы метаданных в едином хранилище и может синхронизировать и реплицировать метаданные, распределенные по всему хранилищу данных. Таким образом, репозиторий метаданных предоставляет собой источник метаданных для других инструментов хранилища данных. Среди примеров репозиториев метаданных следует назвать PLATINUM Repository фирмы Platinum Technologies, ROCHADE Repository фирмы R&O, Directory Manager фирмы Prisms Solution и Universal Directory фирмы Logic Works.

Инструменты управления и администрирования

Хранилище данных обязательно должно включать инструменты администрирования, предназначенные для управления столь сложной средой. Эти инструменты относительно ограничены в своих

возможностях, особенно те, которые хорошо интегрированы с различными типами метаданных и с типичными операциями в хранилищах данных. Инструменты управления и администрирования хранилищ данных должны позволять выполнять такие операции, как:

- мониторинг загрузки данных из нескольких источников;
- проверка качества и целостности данных;
- управление и обновление метаданных;
- мониторинг текущей производительности базы данных с целью обеспечения быстрой реакции на запрос и эффективного использования ресурсов;
- аудит процессов использования хранилища данных с целью сбора информации о стоимости выполненной каждым пользователем работы;
- репликация, разбиение и распределение данных;
- поддержка эффективного управления хранилищем данных;
- очистка данных;
- архивирование и резервное копирование данных;
- средства восстановления после сбоя;
- управление средствами защиты.

В качестве примеров инструментов управления и администрирования можно назвать пакеты HP Intelligent Warehouse фирмы HewlettPackard, FlowMark and Data Hub фирмы IBM, Site Analyzer фирмы Information Builders, Inspect/SQL фирмы Precise Software Solutions, Prism Warehouse Manager фирмы Prisms Solutions, Enterprise Control and Coordination фирмы Red Brick Systems, SAS/CPE фирмы SAS Institute и SourcePoint фирмы Software AG.

ОСНОВНАЯ ЛИТЕРАТУРА

1. Карпова, Т.С. Базы данных: модели, разработка, реализация: учебное пособие / Т.С. Карпова. - 2-е изд., исправ. - Москва: Национальный Открытый Университет «ИНТУИТ», 2016. - 241 с. : ил. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429003

2. Давыдова, Е.М. Базы данных [Электронный ресурс]: учеб. пособие / Е.М. Давыдова, Н.А. Новгородова. — Электрон. дан. — Москва: ТУСУР, 2007. — 166 с. — Режим доступа: https://e.lanbook.com/book/11636. — Загл. с экрана.

3. Харрингтон, Д. Проектирование объектно ориентированных баз данных [Электронный ресурс] — Электрон. дан. — Москва: ДМК Пресс, 2007. — 272 с. — Режим доступа: https://e.lanbook.com/book/1231. — Загл. с экрана.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

4. Голицина О.Л., Максимов Н.В., Попов И.И. Базы данных: учеб. пособие. – М.: Форум:Инфра-М, 2007.

5. Гагарин Ю.Е. Применение языка SQL в MS Access: учебно-методическое пособие. – М.: МГТУ им. Н.Э. Баумана, 2012.

Электронные ресурсы:

1. Научная электронная библиотека http://eLIBRARY.RU
2. Электронно-библиотечная система http://e.lanbook.com
3. Электронно-библиотечная система «Университетская библиотека онлайн» http://biblioclub.ru
4. Электронно-библиотечная система IPRBook http://www.iprbookshop.ru