Федеральное агентство по образованию
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ГОУВПО «АмГУ»

УТВЕРЖДАЮ
Зав. кафедрой ТиЭФ
_____Е.А. Ванина
«_____»_____2007

ОРГАНИЗАЦИЯ БАЗ ДАННЫХ
УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО ДИСЦИПЛИНЕ

для специальности 010700 – «физика»

Составитель: к.ф.-м.н,
Стукова Е.В.

Благовещенск
2007

Печатается по решению
редакционно-издательского
совета инженерно-физического
факультета Амурского
государственного
университета

Е.В. Стукова

Учебно-методический комплекс по дисциплине «Организация баз данных» для студентов очной формы обучения специальности 010700 – «Физика». – Благовещенск: Амурский гос. ун-т, 2007. – 91с.

Содержание

Министерство образования и науки РФ
Федеральное агентство по образованию
АМУРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
(ГОУВПО « АмГУ»)

УТВЕРЖДАЮ
Проректор по УНР
_____Е.С.Астапова

«____»_____2006 г

РАБОЧАЯ ПРОГРАММА

по дисциплине "Организация баз данных"
для специальности 010701 "Физика"

курс 4 семестр 8

Лекции 36 (час.) Экзамен 8

Лабораторные работы 18 (час.)

Практические занятия 18 (час.)

Курсовая работа 8 сем.

Самостоятельная работа _6__ (час.)

Всего часов _108___ час.

Составитель: доцент Чепак Л.В.

Факультет Математики и информатики

Кафедра Информационных и управляющих систем

2006 г.

Рабочая программа составлена на основании Государственного образовательного стандарта ВПО по специальности 010701 «Физика»

Рабочая программа обсуждена на заседании кафедры Информационных и управляющих систем

«___» _____2006 г., протокол №_____

Заведующий кафедрой А.В. Бушманов

Рабочая программа одобрена на заседании УМС 010701 «Физика»

«___» _____2006 г., протокол №

Председатель А.В. Бушманов

СОГЛАСОВАНО

Начальник УМУ
_____ Г.Н.Торопчина

«__» _____2006 г.

СОГЛАСОВАНО

Председатель УМС факультета
_____ Е.Л.Еремин

«__» _____2006 г.

СОГЛАСОВАНО
Заведующий выпускающей кафедрой
_____ А.В. Бушманов

«__» _____2006 г.

1. ЦЕЛИ И ЗАДАЧИ ДИСЦИПЛИНЫ, ЕЕ МЕСТО В УЧЕБНОМ ПРОЦЕССЕ

1.1. Цели и задачи дисциплины

Использование баз данных становится неотъемлемой составляющей деятельности любой организации. В связи с этим большую актуальность приобретает освоение принципов построения и эффективного использования соответствующих технологий и программных продуктов: систем управления базами данных, средств администрирования и защиты баз данных.

Курс "Организация баз данных" знакомит студентов с основами организации баз данных, методами их проектирования и использования. Лекционный курс сопровождается практическими занятиями и лабораторным практикумом.

Целью дисциплины является изучение и практическое освоение методов создания баз данных и их последующей эксплуатации. Рассматриваются теоретические и прикладные вопросы применения современных систем управления базами данных.

1.2. Требования к уровню освоения содержания дисциплины

В результате изучения курса студенты должны

знать:

–принципы организации и архитектуры банков данных;

–модели баз данных;

–современные методы и средства разработки и синтеза структур информационных моделей предметных областей автоматизированных систем обработки информации и управления;

–последовательность и содержание этапов проектирования баз данных;

–современные методики синтеза и оптимизации структур баз данных;

–основные конструкции языков манипулирования данными;

–методики оптимизации процессов обработки запросов;

–современные методы обеспечения целостности данных;

–методы организации баз данных на носителях информации;

уметь:

- применять современную методологию для исследования и синтеза информационных моделей предметных областей;
- применять современную методологию на стадии проектирования - обследование, выбор и системное обоснование проектных решений по структуре информационных моделей и базам данных, по архитектуре банка данных и его компонентам;
- применять методы проектирования баз данных и составления программ взаимодействия с базой данных;
- применять методы организации работы в коллективах разработчиков баз данных;

иметь представление:

–о тенденциях и перспективах развития современных систем управления базами данных.

1.3. **Перечень дисциплин с указанием разделов (тем), усвоение которых студентами необходимо при изучении данной дисциплины**

Изучение данной дисциплины требует от студентов предварительного усвоения таких дисциплин как «Информатика», «Дискретная математика» в объеме государственного образовательного стандарта высшего профессионального образования.

2. СОДЕРЖАНИЕ ДИСЦИПЛИНЫ

2.1. Федеральный компонент

Дисциплина «Организация баз данных» является дисциплиной, входящей в блок дисциплин специализаций для специальности 010701 «Физика». Государственный стандарт – Д.С.03.

2.2. Наименование тем, их содержание, объем в лекционных часах

ТЕМАТИЧЕСКИЙ ПЛАН ЛЕКЦИОННЫХ ЗАНЯТИЙ

Наименование темы	Кол-во часов
1. Введение в базы данных.	2
2 Модели данных	4
3. Реляционная алгебра и реляционное исчисление.	2
4. Архитектура системы баз данных.	2
5. Теория проектирования баз данных.	2
6. Инфологическое проектирование БД.	8
7. Логическое проектирование БД.	8
8. Физическое проектирование БД.	4
9. Обзор СУБД.	4
ИТОГО	36

Тема 1. Введение в базы данных.

Информация и данные. Базы и банки данных. Предметная область банка данных. Базы данных (БД) в составе автоматизированных систем. Компоненты систем баз данных. Функции приложения базы данных. Функции СУБД (систем управления базой данных). Преимущества и недостатки СУБД. Выбор СУБД.

Тема 2. Модели данных

Понятие модели данных. Структуры данных. Основные операции над данными. Ограничения целостности. Выбор модели данных. Иерархическая, сетевая и реляционная модели данных, их типы структур, основные операции и ограничения. Схема данных.

Тема 3. Реляционная алгебра и реляционное исчисление.

Формальное определения реляционной алгебры. Схема отношения и схема базы данных. Основные и дополнительные операции реляционной алгебры: объединение, выборка, разность, проекция, декартово произведение, селекция, соединение, пересечение, деление. Системы реляционного исчисления: исчисление с переменными кортежами, исчисление с переменными на доменах.

Тема 4. Архитектура системы баз данных.

Архитектура ANSI/SPARC. Внешний, концептуальный и внутренний уровни. Администратор базы данных. Функции администратора базы данных.

Тема 5. Теория проектирования баз данных

Методология проектирования БД. Основные этапы проектирования БД; анализ и определение требований к БД; инфологическое проектирование БД; датологическое проектирование БД. Задачи инфологического, логического и физического этапов проектирования.

Тема 6. Инфологическое проектирование БД

Модель "Сущность - связь". Типы связей. Моделирование локальных представлений. Объединение моделей локальных представлений: идентичность, агрегация, обобщение, выявление противоречий. Пример инфологической модели.

Тема 7. Логическое проектирование БД.

Общие положения. Проектирование реляционной логической модели базы данных. Установление дополнительных логических связей. Отображение инфологической модели на реляционную модель. Совокупность отношений

реляционной модели. Нормализация отношений: 1НФ, 2НФ, 3НФ, НФБК, 4НФ, 5НФ.

Тема 8. Физическое проектирование БД.

Компоненты этапа физического проектирования. Проектирование формата хранимой записи. Проектирование методов доступа. Статическое и динамическое хеширование. Жизненный цикл БД. Реорганизация БД.

Тема 9. Обзор СУБД.

Функциональные возможности СУБД. Производительность СУБД. Обеспечение целостности данных на уровне базы данных. Обеспечение безопасности. Доступ к данным посредством языка запросов SQL. Возможности запросов и инструментальные средства разработки прикладных программ. Схема обобщенной технологии работы в СУБД.

2.3. Практические занятия, их содержание и объем в часах.

ТЕМАТИЧЕСКИЙ ПЛАН ПРАКТИЧЕСКИХ ЗАНЯТИЙ

Наименование темы	Кол-во часов
1. Проектирование конкретной БД. (описание предметной области, определение границ предметной области, выявление информационных запросов пользователей).	4
2. Инфологическое проектирование (разработка спецификаций сущностей, атрибутов, связей, выбор ключей, создание справочника задач, построение концептуальной инфологической модели)	6
3. Логическое проектирование (проектирование реляционной логической модели базы данных, составление матрицы частоты совместного использования сущностей на основе справочника задач, оценка объема лишнего чтения, установление дополнительных логических связей, отображение инфологической модели на реляционную модель, получение совокупности отношений реляционной модели)	4
4. Нормализация отношений (приведение совокупности отношений к 1НФ, 2НФ, 3НФ)	2
5. Физическое проектирование	2
ИТОГО	18

2.4. Лабораторные занятия, их содержание и объем в часах.

ТЕМАТИЧЕСКИЙ ПЛАН ЛАБОРАТОРНЫХ ЗАНЯТИЙ

Наименование темы	Кол-во часов
1. Создание локальной базы данных, создание таблиц с помощью конструктора, целостность данных, создание ключей и индексов, определение типов данных. Маски ввода данных.	2
2. Схема базы данных, установление связей между таблицами. Обработка данных в таблицах. Сортировка и фильтрация данных. Обычные и расширенные фильтры.	2
3. Создание простых запросов, псевдонимы.	2
4.Создание запросов на основе нескольких таблиц. Выборка данных с условием. Использование выражений в запросах.	2
5. Соединение таблиц. Внутреннее, рекурсивное, внешнее левое и правое соединения, соединение по отношению	2
6. Перекрестные запросы. Использование функций в запросах.	2
7. Запросы на создание, на обновление, на удаление, каскадное удаление и каскадное обновление данных.	2
8. Создание форм. Элементы управления формы. Диаграммы. Составные и связанные формы. Оформление формы. Ввод данных через форму.	2
9. Создание отчетов. Вычисляемые поля в отчете. Сортировка и группировка данных.	2
ИТОГО	18

Лекция 1. Введение в базы данных

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

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

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

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

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

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

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

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

Более точно, к числу функций СУБД принято относить следующие:

Непосредственное управление данными во внешней памяти

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

Управление буферами оперативной памяти

СУБД обычно работают с БД значительного размера; по крайней мере этот размер обычно существенно больше доступного объема оперативной памяти. Понятно, что если при обращении к любому элементу данных будет

производиться обмен с внешней памятью, то вся система будет работать со скоростью устройства внешней памяти. Практически единственным способом реального увеличения этой скорости является буферизация данных в оперативной памяти. При этом, даже если операционная система производит общесистемную буферизацию (как в случае ОС UNIX), этого недостаточно для целей СУБД, которая располагает гораздо большей информацией о полезности буферизации той или иной части БД. Поэтому в развитых СУБД поддерживается собственный набор буферов оперативной памяти с собственной дисциплиной замены буферов.

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

Управление транзакциями

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

(если, конечно, такая система заслуживает названия СУБД). Но понятие транзакции гораздо более важно в многопользовательских СУБД.

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

С управлением транзакциями в многопользовательской СУБД связаны важные понятия *сериализации транзакций* и *сериального плана выполнения смеси транзакций*. Под сериализаций параллельно выполняющихся транзакций понимается такой порядок планирования их работы, при котором суммарный эффект смеси транзакций эквивалентен эффекту их некоторого последовательного выполнения. Сериальный план выполнения смеси транзакций - это такой план, который приводит к сериализации транзакций. Понятно, что если удается добиться действительно сериального выполнения смеси транзакций, то для каждого пользователя, по инициативе которого образована транзакция, присутствие других транзакций будет незаметно (если не считать некоторого замедления работы по сравнению с однопользовательским режимом).

Существует несколько базовых алгоритмов сериализации транзакций. В централизованных СУБД наиболее распространены алгоритмы, основанные на синхронизационных захватах объектов БД. При использовании любого алгоритма сериализации возможны ситуации конфликтов между двумя или

более транзакциями по доступу к объектам БД. В этом случае для поддержания сериализации необходимо выполнить откат (ликвидировать все изменения, произведенные в БД) одной или более транзакций. Это один из случаев, когда пользователь многопользовательской СУБД может реально (и достаточно неприятно) ощутить присутствие в системе транзакций других пользователей.

Журнализация

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

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

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

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

Самая простая ситуация восстановления - индивидуальный откат транзакции. Строго говоря, для этого не требуется общесистемный журнал изменений БД. Достаточно для каждой транзакции поддерживать локальный журнал операций модификации БД, выполненных в этой транзакции, и производить откат транзакции путем выполнения обратных операций, следуя от конца локального журнала. В некоторых СУБД так и делают, но в большинстве систем локальные журналы не поддерживают, а индивидуальный откат транзакции выполняют по общесистемному журналу, для чего все записи от одной транзакции связывают обратным списком (от конца к началу).

При мягком сбое во внешней памяти основной части БД могут находиться объекты, модифицированные транзакциями, не закончившимися к моменту

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

Для восстановления БД после жесткого сбоя используют журнал и архивную копию БД. Грубо говоря, архивная копия - это полная копия БД к моменту начала заполнения журнала (имеется много вариантов более гибкой трактовки смысла архивной копии). Конечно, для нормального восстановления БД после жесткого сбоя необходимо, чтобы журнал не пропал. Как уже отмечалось, к сохранности журнала во внешней памяти в СУБД предъявляются особо повышенные требования. Тогда восстановление БД состоит в том, что исходя из архивной копии по журналу воспроизводится работа всех транзакций, которые закончились к моменту сбоя. В принципе, можно даже воспроизвести работу незавершенных транзакций и продолжить их работу после завершения восстановления. Однако в реальных системах это обычно не делается, поскольку процесс восстановления после жесткого сбоя является достаточно длительным.

Поддержка языков БД

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

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

Прежде всего, язык SQL сочетает средства SDL и DML, т.е. позволяет определять схему реляционной БД и манипулировать данными. При этом именование объектов БД (для реляционной БД - именование таблиц и их столбцов) поддерживается на языковом уровне в том смысле, что компилятор языка SQL производит преобразование имен объектов в их внутренние идентификаторы на основании специально поддерживаемых служебных таблиц-

каталогов. Внутренняя часть СУБД (ядро) вообще не работает с именами таблиц и их столбцов.

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

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

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

Лекция 2. Модели данных

Модель данных, или **концептуальное описание предметной области** - самый абстрактный уровень проектирования баз данных.

Структуры данных

База данных, организованная с помощью инвертированных списков, похожа на реляционную БД, но с тем отличием, что хранимые таблицы и пути доступа к ним видны пользователям. При этом:

a. Строки таблиц упорядочены системой в некоторой физической последовательности.
b. Физическая упорядоченность строк всех таблиц может определяться и для всей БД (так делается, например, в Datacom/DB).
c. Для каждой таблицы можно определить произвольное число ключей поиска, для которых строятся индексы. Эти индексы автоматически поддерживаются системой, но явно видны пользователям.

Иерархические базы данных

В основе данной модели - иерархическая модель данных. В этой модели имеется один главный объект и остальные - подчиненные - объекты, находящиеся на разных уровнях иерархии. Взаимосвязи объектов образуют иерархическое дерево с одним корневым объектом.

Иерархическая БД состоит из упорядоченного набора нескольких экземпляров одного типа дерева. Автоматически поддерживается целостность ссылок между предками и потомками. Основное правило: никакой потомок не может существовать без своего родителя (рис. 1).

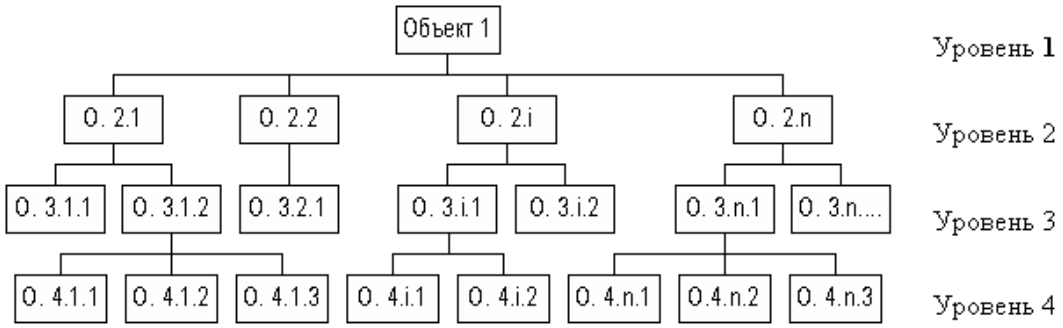


Рис. 1. Схема иерархической модели данных

Типичным представителем (наиболее известным и распространенным) является Information Management System (IMS) фирмы IBM. Первая версия появилась в 1968 г. До сих пор поддерживается много баз данных этой системы.

Сетевые базы 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.

В сетевой модели данных любой объект может быть одновременно и главным, и подчиненным, и может участвовать в образовании любого числа взаимосвязей с другими объектами. Сетевая БД состоит из набора записей и набора связей между этими записями, а если говорить более точно - из набора экземпляров каждого типа из заданного в схеме БД набора типов записи и набора экземпляров каждого типа из заданного набора типов связи (рис. 2.).

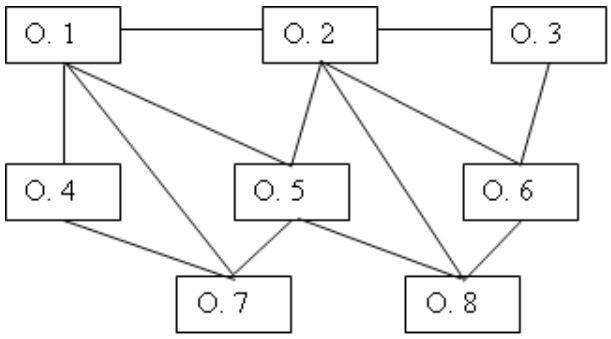


Рис. 2. Схема сетевой модели

Типичным представителем является **Integrated Database Management System (IDMS)** компании Cullinet Software, Inc., предназначенная для использования на машинах основного класса фирмы IBM под управлением большинства операционных систем. Архитектура системы основана на предложениях Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL) - организации, ответственной за определение языка программирования Кобол. Отчет DBTG был опубликован в 1971 г., а позже появилось несколько систем, среди которых IDMS.

Современные базы данных

Реляционные системы

Реляционные системы далеко не сразу получили широкое распространение. В то время как основные теоретические результаты в этой области были получены еще в 70-х годах и тогда же появились первые прототипы реляционных СУБД, долгое время считалось невозможным добиться эффективной реализации таких систем. Однако постепенное накопление методов и алгоритмов организации реляционных баз данных и управления ими привели к тому, что уже в середине 80-х годов реляционные системы практически вытеснили с мирового рынка ранние СУБД.

Реляционная модель данных основывается на математических принципах, вытекающих непосредственно из теории множеств и логики предикатов. Эти принципы впервые были применены в области моделирования данных в конце 1960-х гг. доктором Е.Ф. Коддом, в то время работавшим в IBM, а впервые опубликованы - в 1970 г. [1].

Техническая статья "Реляционная модель данных для больших разделяемых банков данных" доктора Е.Ф. Кодда, опубликованная в 1970 г., является родоначальницей современной теории реляционных БД. Доктор Кодд определил 13 правил реляционной модели (которые называют 12 правилами Кодда).

Атрибуты сущности бывают:

1. **Идентифицирующие и описательные**. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются потенциальными ключами. Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один **первичный ключ** (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными.
2. **Простые и составные.** Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от особенностей процессов его использования и может быть связано с обеспечением высокой скорости работы с большими базами данных.

3. **Однозначные и многозначные** - могут иметь соответственно одно или много значений для каждого экземпляра сущности.
4. **Основные и производные**. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст человека вычисляется на основе даты его рождения и текущей даты

Спецификация атрибута состоит из его названия, указания типа данных и описания ограничений целостности - множества значений (или домена), которые может принимать данный атрибут.

Домен - это набор всех допустимых значений, которые может содержать атрибут. Понятие "домен" часто путают с понятием "тип данных". Необходимо различать эти два понятия. Тип данных - это физическая концепция, а домен - логическая. Например, "целое число" - это тип данных, а "возраст" - это домен.

Связи - на концептуальном уровне представляют собой простые ассоциации между сущностями. Например, утверждение "Покупатели приобретают продукты" указывает, что между сущностями "Покупатели" и "Продукты" существует связь, и такие сущности называются **участниками** этой связи.

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

Каждая связь в реляционной модели характеризуется именем, обязательностью, типом и степенью. Различают **факультативные** и **обязательные** связи. Если сущность одного типа оказывается по необходимости связанной с сущностью другого типа, то между этими типами объектов существует обязательная связь (обозначается двойной линией). Иначе связь является факультативной.

Степень связи определяется количеством сущностей, которые охвачены данной связью. Пример бинарной связи - связь между отделом и сотрудниками, которые в нем работают.

Схемой реляционной базы данных называется набор заголовков отношений, входящих в базу данных.

Хотя любое отношение можно изобразить в виде таблицы, нужно четко понимать, что *отношения не являются таблицами*. Это близкие, но не совпадающие понятия.

Лекция 3. Реляционная алгебра и реляционное исчисление

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

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

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

Специальные реляционные операции включают:

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

Кроме того, в состав алгебры включается операция присваивания, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения. Если не вдаваться в некоторые тонкости, которые мы рассмотрим в следующих подразделах, то почти все операции предложенного выше набора обладают очевидной и простой интерпретацией.

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

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

Заголовок отношения представляет собой множество пар <имя-атрибута, имя-домена>. Если посмотреть на общий обзор реляционных операций, приведенный в предыдущем подразделе, то видно, что домены атрибутов

результирующего отношения однозначно определяются доменами отношений-операндов. Однако с именами атрибутов результата не всегда все так просто.

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

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

Другие проблемы связаны с операцией взятия прямого произведения двух отношений. В теории множеств прямое произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств. Поскольку отношения являются множествами, то и для любых двух отношений возможно

получение прямого произведения. Но результат не будет отношением! Элементами результата будут являться не кортежи, а пары кортежей.

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

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

Эти соображения приводят к появлению понятия *совместимости по 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 отношений.

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

Реляционное исчисление

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

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

- выполнить соединение отношений СОТРУДНИКИ и ОТДЕЛЫ по условию СОТР_НОМ = ОТД_НАЧ;
- ограничить полученное отношение по условию ОТД_КОЛ > 50;
- спроецировать результат предыдущей операции на атрибут СОТР_ИМЯ, СОТР_НОМ.

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

Во второй формулировке мы указали лишь характеристики результирующего отношения, но ничего не сказали о способе его формирования. В этом случае система должна сама решить, какие операции и в каком порядке нужно выполнить над отношениями СОТРУДНИКИ и ОТДЕЛЫ. Обычно говорят, что алгебраическая формулировка является процедурной, т.е.

задающей правила выполнения запроса, а логическая - описательной (или декларативной), поскольку она всего лишь описывает свойства желаемого результата. Как мы указывали в начале лекции, на самом деле эти два механизма эквивалентны и существуют не очень сложные правила преобразования одного формализма в другой.

Кортежные переменные

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

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

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

Реляционное исчисление доменов

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

Основным формальным отличием исчисления доменов от исчисления кортежей является наличие дополнительного набора предикатов, позволяющих выражать так называемые условия членства. Если R - это n-арное отношение с атрибутами a_1, a_2, ..., a_n, то условие членства имеет вид

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

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

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

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

Лекция 4. Архитектура системы баз данных

Архитектура системы баз данных ANSI/SPARC состоит из 3 уровней

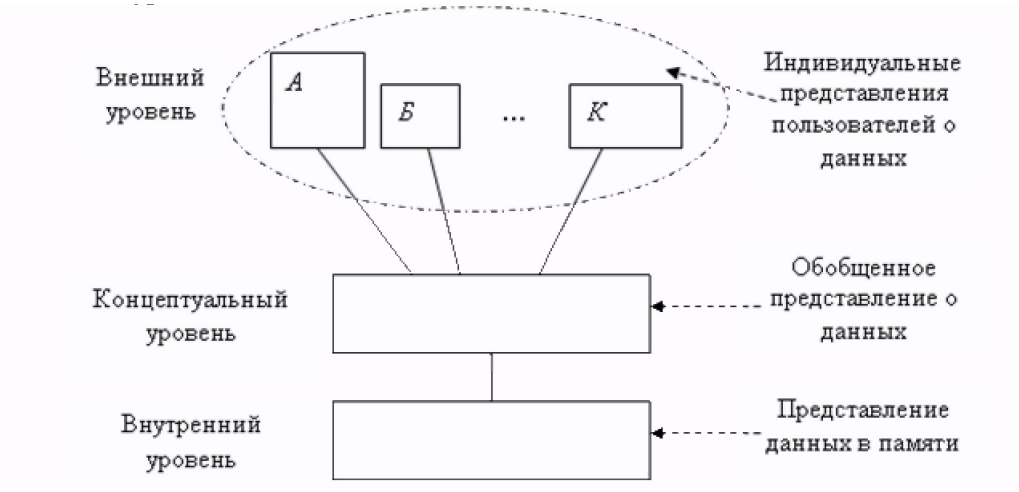


Рис. 1. Принципиальная архитектура системы баз данных

Как видим, в этой архитектуре учитывается, что с базой данных может работать несколько пользователей (в нашем случае – А, Б, … К, …) с разными информационными потребностями.

При этом исходят из того, что любой Банк Данных должен поддерживать разнообразные представления пользователей о Предметной Области.

Предметная область – часть реального мира, представляющая интерес для данного исследования (использования).

Рассмотрим каждый уровень архитектуры подробнее.

Внешний уровень

Отдельного пользователя интересует, как правило, только некоторая часть всей базы данных. Представление отдельного пользователя о предметной области называется **внешним представлением**.

Таким образом, внешний уровень состоит из внешних представлений (которые в английской терминологии называются views) ≈

Внешнее представление – это содержимое базы данных, каким его видит определенный пользователь.

Состоит из множества типов **внешних записей**.

Под **записью** понимается группа взаимосвязанных элементов данных, рассматриваемых как единое целое.

Пример 1.

Пользователь из отдела кадров может рассматривать базу данных как набор записей с информацией об отделах и набор записей с информацией о служащих, и может ничего не знать о записях с информацией о деталях и поставщиках, с которыми работают пользователи из отдела поставок и сбыта.

Для использования компьютера при обработке информации о предметной области эту информацию нужно представлять в специальном виде, строго, формализовано. Формализация - неотъемлемая часть разработки любой программной системы.

Способ формального описания баз данных заключается в использовании **схем**.

Схема - описание структуры БД в формализованном виде.

Схемы используются для строгого, формального описания каждого уровня архитектуры.

На внешнем уровне каждое представление пользователя описывается с помощью **внешней схемы**. Для внешних схем в общем случае используется собственный язык.

Концептуальный уровень

Концептуальное представление формируется на основе интеграции внешних представлений пользователей.

Концептуальное представление—представление **всего** содержимого БД.

Как правило, концептуальное представление существенно отличается от внешних представлений отдельных пользователей (поскольку суммирует их разрозненные представления в одно обобщенное), и состоит из множества типов **концептуальных записей**.

Концептуальное представление определяется с помощью **концептуальной схемы**.

Концептуальная схема – описание полной общей логической структуры базы данных

Концептуальная схема использует (в общем случае) другой язык описания данных. Определения концептуального языка должны относиться **только** к содержанию данных, не касаясь физических подробностей их хранения.

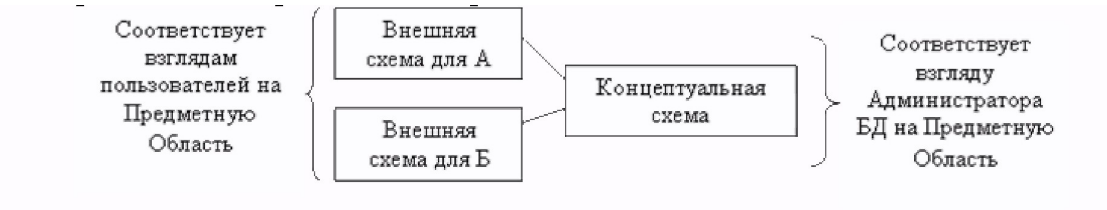


Рис. 2. Соответствие между взглядами пользователей и взглядом администратора БД

В концептуальной схеме **не рассматриваются** способы организации хранения или методы доступа к хранимым данным.

Определения в концептуальной схеме, помимо описания типов записей, могут включать такие средства, как безопасность, правила поддержания целостности.

Записи концептуального уровня не обязаны совпадать с записями внешних уровней.

Внутренний уровень

Внутреннее представление БД — представление структуры хранения записей, состоит из множества типов **хранимых записей**.

Внутреннее представление так же не связано с физическим уровнем, т.е. не рассматриваются физические записи, физические устройства хранения (например, цилиндры и дорожки цилиндры и дорожки), способы доступа к данным, расположенным удаленно.

Внутреннее представление описывается с помощью **внутренней схемы**, которая определяет не только различные типы хранимых записей, но и существующие индексы, способы представления хранимых полей, и т.д.

Внутренняя схема использует внутренний язык определения данных. Записи внутреннего уровня чаще всего не совпадают с записями внешних и концептуального уровней.

Детализованная архитектура системы БД

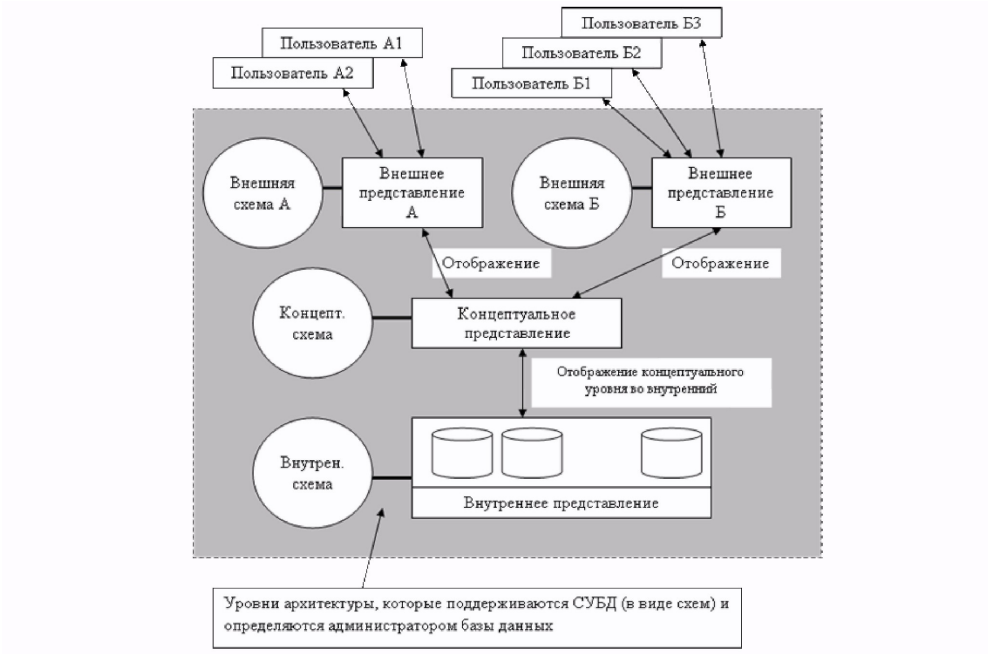


Рис. 3. Детализованная архитектура системы баз данных

Как видно, выбор СУБД определяет способ представления внешних, концептуальной и внутренней схем. Ответственность за корректное представление этих схем, а также за управление данными и схемами несет специальная категория пользователей – администраторы баз данных.

Группа администратора базы данных (АБД)

Поскольку система баз данных может быть весьма большой и может иметь много пользователей, должно существовать лицо или группа лиц, управляющих этой системой. Такое лицо называется **администратором базы данных** (АБД).

В любой базе данных должен быть хотя бы один человек, выполняющий административные обязанности; если база данных большая, эти обязанности могут быть распределены между несколькими администраторами.

Обязанности администратора базы данных

В обязанности администратора могут входить:

1• инсталляция и обновление версий СУБД и прикладных инструментов

2• распределение дисковой памяти и планирование будущих требований системы к памяти

3• создание первичных структур памяти в базе данных (табличных пространств) по мере проектирования приложений разработчиками приложений

4• создание первичных объектов (таблиц, представлений, индексов) по мере проектирования приложений разработчиками

5• модификация структуры базы данных в соответствии с потребностями приложений

6• зачисление пользователей и поддержание защиты системы

7• соблюдение лицензионного соглашения по СУБД

8• управление и отслеживание доступа пользователей к базе данных

9• отслеживание и оптимизация производительности базы данных

10• планирование резервного копирования и восстановления

11• поддержание архивных данных на устройствах хранения информации

12• осуществление резервного копирования и восстановления

13• обращение за техническим сопровождением к разработчикам СУБД

Обязанности сотрудника службы безопасности

В некоторых случаях база данных должна также иметь одного или нескольких сотрудников службы безопасности. Сотрудник службы безопасности главным образом отвечает за регистрацию новых пользователей, управление и отслеживание доступа пользователей к базе данных, и защиту базы данных.

Обязанности разработчика приложений

В обязанности разработчика приложений входит:

1• проектирование и разработка приложений базы данных

2• проектирование структуры базы данных в соответствии с требованиями приложений

3• оценка требований памяти для приложения

4• формулирование модификаций структуры базы данных для приложения

5• передача вышеупомянутой информации администратору базы данных

6• настройка приложения в процессе его разработки

7• установка мер по защите приложения в процессе его разработки

Основные логические объекты базы данных в современной СУБД.

Несмотря на достаточно простую архитектуру СБД в целом, на практике СУБД поддерживает более детальную классификацию объектов СБД, чем внешние, концептуальная и внутренняя схемы.

СУБД считает объектами СБД все доступные пользователю непосредственно компоненты структуры данных. Набор объектов обычно отличается от СУБД к СУБД, и даже от версии к версии одной СУБД. Чаще всего различают два набора объектов. Один набор объектов ведет начало от СУБД ORACLE, второй – предложен Microsoft.

Эти наборы объектов отличаются способом введения значений, за приращением которых следит СУБД – т.н. счетчики, или последовательности.

Рассмотрим набор объектов СБД, который поддерживается MS SQL Server 2000.

Таблица (table) – единственный объект СБД, в котором реально хранятся данные; являются двумерными матрицами.

Хранимая процедура (stored procedure) – набор команд SQL и Transact-SQL, сохраненный под определенным именем. Пользователи работают с таким набором, как с единым целым, обращаясь к нему по имени.

Триггер (trigger) – специальный вид хранимых процедур, который автоматически запускается сервером при удалении, изменении или добавлении записи в таблицу.

Представление (view) – виртуальная таблица, отражает результат запроса на выборку. Пользователь может работать с представлением как с обычной таблицей. С помощью представлений обычно описывается внешняя схема для некоторой группы пользователей.

Индекс (index) – файл специального вида, предназначен для повышения скорости работы с данными (например, для ускорения поиска данных) за счет представления этих данных в упорядоченном виде

Пользовательский тип данных (user-defined datatype) – тип данных, созданный пользователем на основе встроенных типов данных СУБД

Функция пользователя (user-defined function) – функция, создаваемая пользователем. Как и хранимая процедура, функция – тоже набор команд. В отличие от хранимой процедуры, функция может быть использована в выражениях (как, например, функция вычисления квадратного корня из значения поля).

Ограничение целостности (constraint) – объект базы данных, контролирует логическую целостность данных

Умолчание (default) – объект базы данных, который также как и ограничение целостности, может привязываться к столбцу таблицы или к пользовательскому типу данных.

Набор объектов, который поддерживается ORACLE, отличается от набора MS SQL Server в следующем:

Появляется объект «последовательность»:

Последовательность (sequence) - генерирует уникальные порядковые номера, которые могут использоваться как значения числовых столбцов таблиц базы данных.

Последовательности упрощают прикладное программирование, автоматически генерируя уникальные числовые значения для строк одной или нескольких таблиц. Номера, генерируемые последовательностью, независимы от таблиц, так что одну и ту же последовательность можно использовать для нескольких таблиц. После ее создания, к последовательности могут обращаться различные пользователи, чтобы получать действительные порядковые номера.

Под названием «программная единица» объединяют процедуры, функции и пакеты.

Процедура (procedure) – набор команд SQL и PL/SQL (языка программирования для ORACLE), который исполняется как единое целое и не возвращает результатов.

Функция (function) – набор команд SQL и PL/SQL (языка программирования для ORACLE), который исполняется как единое целое и возвращает результаты.

Пакет (package) – набор процедур и функций, сохраненный под некоторым именем.

Синоним (synonym) – это алиас (дополнительное имя) для таблицы, представлений, последовательности или программной единицы. Синоним не есть объект, но он является прямой ссылкой на объект.

Синонимы используются для:

11. маскировки действительного имени и владельца объекта
22. обеспечения общего доступа к объекту
33. достижения прозрачности местоположения для таблиц, представлений или программных единиц удаленной базы данных
44. упрощения кодирования предложений SQL для пользователей базы данных

Синоним может быть общим или личным. Индивидуальный пользователь может создать **личный синоним**, который доступен только этому пользователю. Администраторы баз данных чаще всего создают **общие синонимы**, благодаря которым объекты базовых схем становятся доступными для общего пользования всем пользователям базы данных.

Кластер (cluster) – это группа из одной или нескольких таблиц, физически хранящихся вместе. В кластеризованных таблицах связанные данные хранятся вместе, более эффективно. В некластеризованных таблицах связанные данные хранятся отдельно, занимая больше места.

Связь баз данных (database link) – именованный объект, который описывает "путь" от одной базы данных к другой. Связи баз данных неявно используются при обращении к **глобальному имени объекта** в распределенной базе данных.

Для того, чтобы управлять отдельными объектами базы данных, СУБД группирует всю информацию об этих объектах в **схеме** (schema). Схема содержит детальное описание свойств всех созданных в БД объектов.

Основные физические объекты базы данных в современной СУБД.

Все объекты и данные одной базы данных хранятся в одном или нескольких файлах данных.

При планировании базы данных планируется и размещение данных по файлам. Каждая база данных состоит минимум из двух файлов – **файла данных** и **файла для хранения журнала транзакций** (действий с базой данных).

Каждый **файл данных** рассматривается СУБД как **набор страниц**. **Страница** – это блок фиксированного размера, который считывается с диска за одну операцию чтения. Размер страницы составляет несколько килобайт, например, в MS SQL Server 2000 размер страницы составляет 8 Кб.

Различают страницы типов **data** (только для хранения данных), **index** (для данных индекса), **text/image** (для хранения данных типа текста, изображения), **Global Allocation Map** (GAM, для хранения информации об использовании

групп страниц – какому файлу какая группа страниц принадлежит), **Page Free Space** (для свободных страниц), **Index Allocation Map** (IAM, для хранения информации о самих группах страниц).

Для более эффективного управления страницами СУБД обычно объединяет группу страниц в **экстент** (extent). Размер экстента, например, в MS SQL Server 2000 – 8 страниц.

Лекция 5. Теория проектирования баз данных

При проектировании базы данных решаются две основных проблемы:

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

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

Более того, мы не будем касаться очень важного аспекта проектирования - определения ограничений целостности (за исключением ограничения

первичного ключа). Дело в том, что при использовании СУБД с развитыми механизмами ограничений целостности (например, SQL-ориентированных систем) трудно предложить какой-либо общий подход к определению ограничений целостности. Эти ограничения могут иметь очень общий вид, и их формулировка пока относится скорее к области искусства, чем инженерного мастерства. Самое большее, что предлагается по этому поводу в литературе, это автоматическая проверка непротиворечивости набора ограничений целостности.

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

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

Понимание принципов разработки, организации и функционирования подобных систем, способов хранения и обработки информации необходимо каждому современному специалисту.

При описании информационной системы предполагается, что она содержит два типа сущностей: операционные сущности, которые выполняют какую-либо обработку (некоторый аналог программы), и пассивные сущности, которые хранят информацию, доступную для пополнения, изменения, поиска, чтения (база данных).

При проектировании сложных информационных систем используется метод декомпозиции - система разбивается на составные части, которые связаны, взаимодействуют друг с другом и образуют иерархическую структуру. Иерархический характер сложных систем хорошо согласуется с принципом групповой разработки. В этом случае деятельность каждого участника проекта ограничивается соответствующим иерархическим уровнем.

Классический подход к разработке сложных систем представляет собой структурное проектирование, при котором осуществляется алгоритмическая

декомпозиция системы по методу "**сверху вниз**". Именно в этом случае можно построить хорошо функционирующую систему с общей базой данных, согласованными форматами использования и обработки информации на всех участках, с оптимальным взаимодействием всех подсистем.

Исторически сложилось так, что некоторые системы разрабатывались по методу "**снизу вверх**": вначале создавались отдельные автоматизированные рабочие места (АРМы), затем предпринимались попытки объединения их в единую информационную систему. Подобные разработки для крупных систем не могут быть успешны.

При создании проекта информационной системы для проектирования ее базы данных следует определить:

1. объекты информационной системы (сущности в концептуальной модели);
2. их свойства (атрибуты);
3. взаимодействие объектов (связи) и информационные потоки внутри и между ними.

При этом очень важен анализ существующей практики реализации информационных процессов и нормативной информации (законов, постановлений правительства, отраслевых стандартов), определяющих необходимый объем и формат хранения и передачи информации. Если радикальной перестройки сложившегося информационного процесса не предвидится, следует учитывать имеющиеся формы хранения и обработки информации в виде журналов, ведомостей, таблиц и т.п. бумажных носителей.

Однако предварительно необходимо выполнить анализ возможности перехода на новые системы учета, хранения и обработки информации, возможно, исходя из имеющихся на рынке программных продуктов-аналогов, разработанных крупными информационными компаниями и частично или полностью соответствующими поставленной задаче.

Схема формирования информационной модели представлена на рис.1.

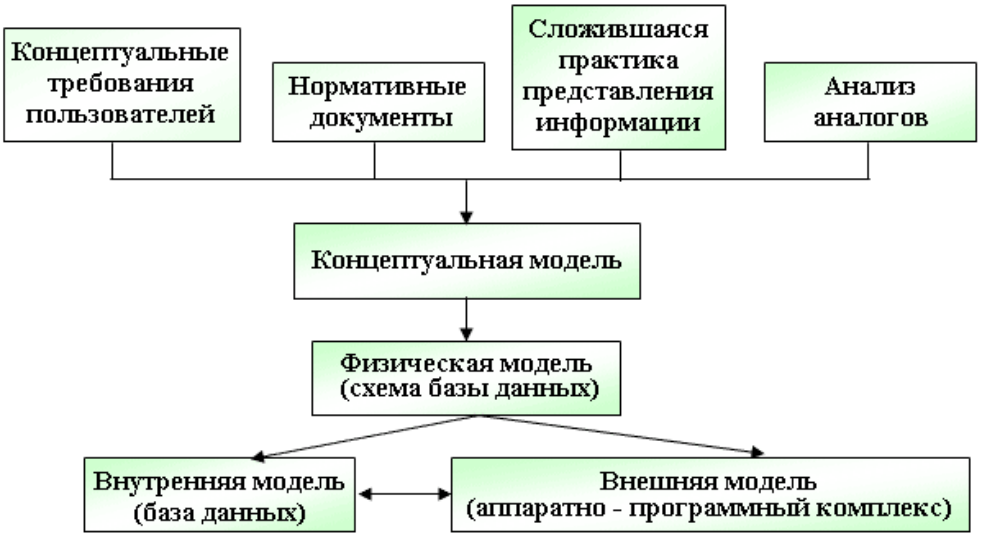


Рис. 1. Схема формирования информационной модели

Концептуальная модель - отображает информационные объекты, их свойства и связи между ними без указания способов физического хранения информации (модель предметной области, иногда ее также называют информационно-логической или инфологической моделью). Информационными объектами обычно являются **сущности** - обособленные объекты или события, информацию о которых необходимо сохранять, имеющие определенные наборы свойств - **атрибутов**.

Физическая модель - отражает все свойства (атрибуты) информационных объектов базы и связи между ними с учетом способа их хранения - используемой СУБД.

Лекция 6. Инфологическое проектирование БД

Модель "Сущность-Связи" (часто ее называют кратко ER-моделью).

На использовании разновидностей ER-модели основано большинство современных подходов к проектированию баз данных (главным образом,

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

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

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

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

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

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

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

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

Каждая связь в реляционной модели характеризуется именем, обязательностью, типом и степенью. Различают **факультативные** и **обязательные** связи. Если сущность одного типа оказывается по необходимости связанной с сущностью другого типа, то между этими типами объектов существует обязательная связь (обозначается двойной линией). Иначе связь является факультативной.

Степень связи определяется количеством сущностей, которые охвачены данной связью. Пример бинарной связи - связь между отделом и сотрудниками, которые в нем работают.

Важнейшая цель проектирования информационной модели - выработка непротиворечивой структурированной интерпретации реально существующей информации изучаемой предметной области и взаимодействия между ее структурными компонентами

Понятие концептуальной модели данных связано с методологией семантического моделирования данных, т.е. с представлением данных в контексте их взаимосвязей с другими данными.

Методология IDEF1X - один из подходов к семантическому моделированию данных, основанный на концепции "сущность-связь" (Entity-Relationship). Это инструмент для анализа информационной структуры систем различной природы. Информационная модель, построенная с помощью IDEF1X-методологии, отображает логическую структуру информации об объектах системы

Таким образом, концептуальная модель, представленная в соответствии со стандартом IDEF1X, является логической схемой базы данных для проектируемой системы

Основными объектами концептуальной модели являются сущности и связи.

Сущность - некоторый обособленный объект или событие моделируемой системы, имеющий определенный набор свойств - атрибутов. Отдельный элемент этого множества называется "экземпляром сущности". Сущность может обладать одним или несколькими атрибутами, которые однозначно идентифицируют каждый образец сущности, и может обладать любым количеством связей с другими сущностями.

Правила для атрибутов сущности:

1. Каждый атрибут должен иметь уникальное имя.
2. Сущность может обладать любым количеством атрибутов.
3. Сущность может обладать любым количеством наследуемых атрибутов, но наследуемый атрибут должен быть частью первичного ключа сущности-родителя.
4. Для каждого экземпляра сущности должно существовать значение каждого его атрибута (правило необращения в нуль - Not Null).
5. Ни один из экземпляров сущности не может обладать более чем одним значением для ее атрибута.

Сущность изображается на ER-диаграмме в виде прямоугольника, в верхней части которого приводится ее название; далее следует список атрибутов. Ключевые атрибуты могут быть выделены подчеркиванием или иным способом.

Стандарт IDEF1X описывает способы изображения двух типов сущностей - независимой и зависимой, и связей - идентифицирующих и неидентифицирующих Каждая сущность может обладать любым количеством связей с другими сущностями.

Сущность является независимой, если каждый ее экземпляр может быть однозначно идентифицирован без определения его связей с другими сущностями.

Сущность называется зависимой, если однозначная идентификация ее экземпляра зависит от его связей с другими сущностями.

Сущность может обладать атрибутами, которые наследуются через связь с родительской сущностью. Последние обычно являются **внешними ключами** и служат для организации связей между сущностями. Если **внешний ключ** сущности используется в качестве ее первичного ключа (PK) или как часть составного первичного ключа, то сущность является зависимой от родительской сущности. Если внешний ключ не является первичным и не входит в составной первичный ключ, то сущность является независимой от родительской сущности.

Если сущность является зависимой, то связь ее с родительской сущностью называется идентифицирующей, в противном случае - неидентифицирующей.

Связь изображается на ER-диаграмме линией, проводимой между сущностью-родителем и сущностью-потомком с точкой на конце линии у сущности-потомка. идентифицирующая связь изображается сплошной линией, неидентифицирующая - пунктирной.

Связи дается имя, выражаемое грамматической формой глагола. Для связи дополнительно может присутствовать указание мощности: какое количество экземпляров сущности-потомка может существовать для сущности-родителя. Имя связи всегда формируется с точки зрения родителя, так что может быть образовано предложение, если соединить имя сущности родителя, имя связи, выражение мощности и имя сущности-потомка (например "много СТУДЕНТов - сдают - ЭКЗАМЕН").

Принципы изображения концептуальных моделей баз данных стандарта IDEF1 и IDEF1X используют CASE Studio и другие CASE-средства. Подобные системы позволяют на основе концептуальной модели генерировать физическую модель и программный код создания базы данных для большинства наиболее распространенных СУБД и серверов баз данных.

Семантическое моделирование данных, ER-диаграммы

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

При этом проявляется ограниченность реляционной модели данных в следующих аспектах:

- Модель не предоставляет достаточных средств для представления смысла данных. Семантика реальной предметной области должна независимым от модели способом представляться в голове проектировщика. В частности, это относится к упоминавшейся нами проблеме представления ограничений целостности.
- Для многих приложений трудно моделировать предметную область на основе плоских таблиц. В ряде случаев на самой начальной стадии проектирования проектировщику приходится производить насилие над

собой, чтобы описать предметную область в виде одной (возможно, даже ненормализованной) таблицы.

- Хотя весь процесс проектирования происходит на основе учета зависимостей, реляционная модель не предоставляет каких-либо средств для представления этих зависимостей.
- Несмотря на то, что процесс проектирования начинается с выделения некоторых существенных для приложения объектов предметной области ("сущностей") и выявления связей между этими сущностями, реляционная модель данных не предлагает какого-либо аппарата для разделения сущностей и связей.

Семантические модели данных

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

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

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

Менее часто реализуется автоматизированная компиляция концептуальной схемы в реляционную. При этом известны два подхода: на основе явного представления концептуальной схемы как исходной информации

для компилятора и построения интегрированных систем проектирования с автоматизированным созданием концептуальной схемы на основе интервью с экспертами предметной области. И в том, и в другом случае в результате производится реляционная схема базы данных в третьей нормальной форме (более точно следовало бы сказать, что автору неизвестны системы, обеспечивающие более высокий уровень нормализации).

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

Основные понятия модели Entity-Relationship (Сущность-Связи)

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

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

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

Получение реляционной схемы из ER-схемы

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

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

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

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

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

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

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

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

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

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

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

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

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

Лекция 7. Логическое проектирование БД

Проектирование реляционных баз данных с использованием нормализации

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

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

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

- первая нормальная форма (1NF);
- вторая нормальная форма (2NF);
- третья нормальная форма (3NF);
- нормальная форма Бойса-Кодда (BCNF);
- четвертая нормальная форма (4NF);

- пятая нормальная форма, или нормальная форма проекции-соединения (5NF или PJ/NF).

Основные свойства нормальных форм:

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

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

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

Функциональная зависимость

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

Полная функциональная зависимость

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

Транзитивная функциональная зависимость

Функциональная зависимость R.X −> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X −> R.Z и R.Z −> R.Y и отсутствует функциональная зависимость R.Z --> R.X.

(При отсутствии последнего требования мы имели бы "неинтересные" транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

Неключевой атрибут

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

Взаимно независимые атрибуты

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

Вторая нормальная форма

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

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

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

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

Третья нормальная форма

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

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

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

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

Нормальная форма Бойса-Кодда

Детерминант

Детерминант - любой атрибут, от которого полностью функционально зависит некоторый другой атрибут.

Нормальная форма Бойса-Кодда

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

Четвертая нормальная форма

Многозначные зависимости

В отношении R (A, B, C) существует многозначная зависимость R.A −> −> R.B в том и только в том случае, если множество значений B, соответствующее паре значений A и C, зависит только от A и не зависит от C.

Теорема Фейджина

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

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

Четвертая нормальная форма

Отношение R находится в четвертой нормальной форме (4NF) в том и только в том случае, если в случае существования многозначной зависимости A −> −> B все остальные атрибуты R функционально зависят от A.

Пятая нормальная форма

Зависимость соединения

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

Пятая нормальная форма

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

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

Лекция 8. Физическое проектирование БД.

При физическом проектировании, решаются вопросы конкретного использования выбранной СУБД для наиболее эффективного выполнения запросов. Здесь выбирается способ организации файлов, методы доступа, способы организации и размеры буферов и блоков, способы индексирования, конкретная функция хэширования и прочее. Обычно СУБД решает эти вопросы автоматически, по умолчанию, но эти решения могут быть изменены с помощью настроек и специальных процедур.

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

- Наличие двух уровней системы: уровня непосредственного управления данными во внешней памяти (а также обычно управления буферами оперативной памяти, управления транзакциями и журнализацией изменений БД) и языкового уровня (например, уровня, реализующего язык SQL). При такой организации подсистема нижнего уровня должна поддерживать во внешней памяти набор базовых структур, конкретная интерпретация которых входит в число функций подсистемы верхнего уровня.
- Поддержание отношений-каталогов. Информация, связанная с именованием объектов базы данных и их конкретными свойствами (например, структура ключа индекса), поддерживается подсистемой языкового уровня. С точки зрения структур внешней памяти отношение-каталог ничем не отличается от обычного отношения базы данных.
- Регулярность структур данных. Поскольку основным объектом реляционной модели данных является плоская таблица, главный набор объектов внешней памяти может иметь очень простую регулярную структуру.

- При этом необходимо обеспечить возможность эффективного выполнения операторов языкового уровня как над одним отношением (простые селекция и проекция), так и над несколькими отношениями (наиболее распространено и трудоемко соединение нескольких отношений). Для этого во внешней памяти должны поддерживаться дополнительные "управляющие" структуры - индексы.
- Наконец, для выполнения требования надежного хранения баз данных необходимо поддерживать избыточность хранения данных, что обычно реализуется в виде журнала изменений базы данных.

Соответственно возникают следующие разновидности объектов во внешней памяти базы данных:

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

Подход System R позволяет использовать более эффективные методы за счет совместного решения проблем физической и логической синхронизации, использовании общих протоколов при управлении буферами и журнализации и т.д. Но при этом в некотором смысле подсистема нижнего уровня становится монолитом; при самой удачной ее структуризации компоненты остаются

связанными общими протоколами взаимодействия. Непродуманные локальные изменения одного компонента могут привести к фатальным последствиям для всей системы. Подход Ingres позволяет упростить структуру системы и сделать ее более гибкой, но это возможно только за счет огрубления алгоритмов: применения более грубых методов управления транзакциями; жестких протоколов журнализации и т.д.

Существуют два принципиальных подхода к физическому хранению отношений. Наиболее распространенным является покортежное хранение отношений (кортеж является единицей физического хранения). Естественно, это обеспечивает быстрый доступ к целому кортежу, но при этом во внешней памяти дублируются общие значения разных кортежей одного отношения и, вообще говоря, могут потребоваться лишние обмены с внешней памятью, если нужна часть кортежа.

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

Поскольку гораздо более распространено хранение по строкам, мы рассмотрим немного более подробно этот способ хранения отношений. Типовой, унаследованной от System R, структурой страницы данных является следующая:

К основным характеристикам этой организации можно отнести следующие:

- Каждый кортеж обладает уникальным идентификатором (tid), не изменяемым во все время существования кортежа. Структура tid следует из приведенного выше рисунка.
- Обычно каждый кортеж хранится целиком в одной странице. Из этого следует, что максимальная длина кортежа любого отношения ограничена размерами страницы. Возникает вопрос: как быть с "длинными" данными, которые в принципе не помещаются в одной странице? Применяются несколько методов. Наиболее простым решением является хранение таких данных в отдельных (вне базы данных) файлах с заменой "длинного" данного в кортеже на имя соответствующего файла. В некоторых системах (например, в предпоследней версии СУБД Informix) такие данные хранились в отдельном наборе страниц внешней памяти, связанном физическими ссылками. Оба эти решения сильно ограничивают возможность работы с длинными данными (как, например, удалить несколько байтов из середины 2-мегабайтной строки?). В настоящее время все чаще используется метод, предложенный несколько лет тому назад в проекте Exodus, когда "длинные" данные организуются в виде B-деревьев последовательностей байтов.
- Как правило, в одной странице данных хранятся кортежи только одного отношения. Существуют, однако, варианты с возможностью хранения в

одной странице кортежей нескольких отношений. Это вызывает некоторые дополнительные расходы по части служебной информации (при каждом кортеже нужно хранить информацию о соответствующем отношении), но зато иногда позволяет резко сократить число обменов с внешней памятью при выполнении соединений.

- Изменение схемы хранимого отношения с добавлением нового столбца не вызывает потребности в физической реорганизации отношения. Достаточно лишь изменить информацию в описателе отношения и расширять кортежи только при занесении информации в новый столбец.
- Поскольку отношения могут содержать неопределенные значения, необходима соответствующая поддержка на уровне хранения. Обычно это достигается путем хранения соответствующей шкалы при каждом кортеже, который в принципе может содержать неопределенные значения.
- Проблема распределения памяти в страницах данных связана с проблемами синхронизации и журнализации и не всегда тривиальна. Например, если в ходе выполнения транзакции некоторая страница данных опустошается, то ее нельзя перевести в статус свободных страниц до конца транзакции, поскольку при откате транзакции удаленные при прямом выполнении транзакции и восстановленные при ее откате кортежи должны получить те же самые идентификаторы.
- Распространенным способом повышения эффективности СУБД является кластеризация отношения по значениям одного или нескольких столбцов. Полезным для оптимизации соединений является совместная кластеризация нескольких отношений.
- С целью использования возможностей распараллеливания обменов с внешней памятью иногда применяют схему декластеризованного хранения отношений: кортежи с общим значением столбца декластеризации размещают на разных дисковых устройствах, обмены с которыми можно выполнять в параллель.

Что же касается хранения отношения по столбцам, то основная идея состоит в совместном хранении всех значений одного (или нескольких) столбцов. Для каждого кортежа отношения хранится кортеж той же степени, состоящий из ссылок на места расположения соответствующих значений столбцов. В последней лекции мы будем рассматривать особенности организации распределенных реляционных СУБД. Одним из приемов является так называемое вертикальное разделение отношений, когда в разных узлах сети хранятся разные проекции данного отношения. Хранение отношения по столбцам в некотором смысле является предельным случаем вертикального разделения отношений.

Индексы

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

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

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

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

B-деревья

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

Поиск в B-дереве - это прохождение от корня к листу в соответствии с заданным значением ключа. Заметим, что поскольку деревья сильно ветвистые и сбалансированные, то для выполнения поиска по любому значению ключа потребуется одно и то же (и обычно небольшое) число обменов с внешней памятью. Более точно, в сбалансированном дереве, где длины всех путей от

корня к листу одни и те же, если во внутренней странице помещается n ключей, то при хранении m записей требуется дерево глубиной $\log_n(m)$, где $\log_n$ вычисляет логарифм по основанию n. Если n достаточно велико (обычный случай), то глубина дерева невелика, и производится быстрый поиск.

Основной "изюминкой" B-деревьев является автоматическое поддержание свойства сбалансированности. Рассмотрим, как это делается при выполнении операций занесения и удаления записей.

При занесение новой записи выполняется:

- Поиск листовой страницы. Фактически, производится обычный поиск по ключу. Если в B-дереве не содержится ключ с заданным значением, то будет получен номер страницы, в которой ему надлежит содержаться, и соответствующие координаты внутри страницы.
- Помещение записи на место. Естественно, что вся работа производится в буферах оперативной памяти. Листовая страница, в которую требуется занести запись, считывается в буфер, и в нем выполняется операция вставки. Размер буфера должен превышать размер страницы внешней памяти.
- Если после выполнения вставки новой записи размер используемой части буфера не превосходит размера страницы, то на этом выполнение операции занесения записи заканчивается. Буфер может быть немедленно вытолкнут во внешнюю память, или временно сохранен в оперативной памяти в зависимости от политики управления буферами.
- Если же возникло переполнение буфера (т.е. размер его используемой части превосходит размер страницы), то выполняется расщепление страницы. Для этого запрашивается новая страница внешней памяти, используемая часть буфера разбивается грубо говоря пополам (так, чтобы вторая половина также начиналась с ключа), и вторая половина записывается во вновь выделенную страницу, а в старой странице модифицируется значение размера свободной памяти. Естественно, модифицируются ссылки по списку листовых страниц.

- Чтобы обеспечить доступ от корня дерева к заново заведенной страницы, необходимо соответствующим образом модифицировать внутреннюю страницу, являющуюся предком ранее существовавшей листовой страницы, т.е. вставить в нее соответствующее значение ключа и ссылку на новую страницу. При выполнении этого действия может снова произойти переполнение теперь уже внутренней страницы, и она будет расщеплена на две. В результате потребуется вставить значение ключа и ссылку на новую страницу во внутреннюю страницу-предка выше по иерархии и т.д.
- Предельным случаем является переполнение корневой страницы B-дерева. В этом случае она тоже расщепляется на две, и заводится новая корневая страница дерева, т.е. его глубина увеличивается на единицу.

При удалении записи выполняются следующие действия:

- Поиск записи по ключу. Если запись не найдена, то значит удалять ничего не нужно.
- Реальное удаление записи в буфере, в который прочитана соответствующая листовая страница.
- Если после выполнения этой подоперации размер занятой в буфере области оказывается таковым, что его сумма с размером занятой области в листовых страницах, являющихся левым или правым братом данной страницы, больше, чем размер страницы, операция завершается.
- Иначе производится слияние с правым или левым братом, т.е. в буфере производится новый образ страницы, содержащей общую информацию из данной страницы и ее левого или правого брата. Ставшая ненужной листовая страница заносится в список свободных страниц. Соответствующим образом корректируется список листовых страниц.
- Чтобы устранить возможность доступа от корня к освобожденной странице, нужно удалить соответствующее значение ключа и ссылку на освобожденную страницу из внутренней страницы - ее предка. При этом может возникнуть потребность в слиянии этой страницы с ее левым или правыми братьями и т.д.

- Предельным случаем является полное опустошение корневой страницы дерева, которое возможно после слияния последних двух потомков корня. В этом случае корневая страница освобождается, а глубина дерева уменьшается на единицу.

Как видно, при выполнении операций вставки и удаления свойство сбалансированности B-дерева сохраняется, а внешняя память расходуется достаточно экономно.

Проблемой является то, что при выполнении операций модификации слишком часто могут возникать расщепления и слияния. Чтобы добиться эффективного использования внешней памяти с минимизацией числа расщеплений и слияний, применяются более сложные приемы, в том числе:

- упреждающие расщепления, т.е. расщепления страницы не при ее переполнении, а несколько раньше, когда степень заполненности страницы достигает некоторого уровня;
- переливания, т.е. поддержание равновесного заполнения соседних страниц;
- слияния 3-в-2, т.е. порождение двух листовых страниц на основе содержимого трех соседних.

Следует заметить, что при организации мультидоступа к B-деревьям, характерного при их использовании в СУБД, приходится решать ряд нетривиальных проблем. Конечно, грубые решения очевидны, например монопольный захват B-дерева на все выполнение операции модификации. Но существуют и более тонкие решения, рассмотрение которых выходит за пределы нашего курса.

И последнее замечание относительно B-деревьев. В литературе вид рассмотренных нами деревьев принято называть B* или B+-деревьями.

Альтернативным и все более популярным подходом к организации индексов является использование техники хэширования. Это очень обширная тема, которая заслуживает отдельного рассмотрения. Мы ограничимся лишь несколькими замечаниями. Общей идеей методов хэширования является применение к значению ключа некоторой функции свертки (хэш-функции),

вырабатывающей значение меньшего размера. Свертка значения ключа затем используется для доступа к записи.

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

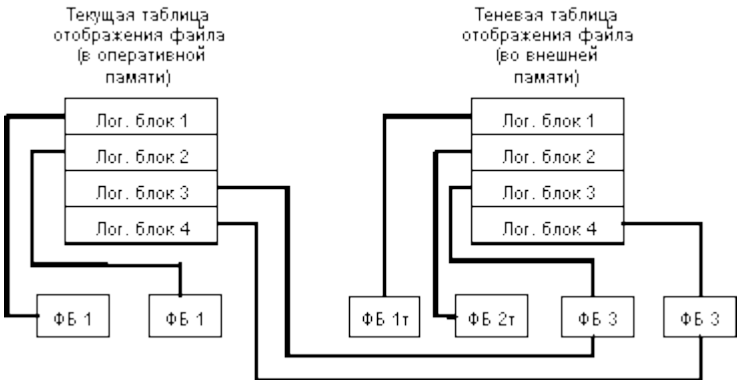
В случае баз данных такие действия являются абсолютно неприемлемыми. Поэтому обычно вводят промежуточные таблицы-справочники, содержащие значения ключей и адреса записей, а сами записи хранятся отдельно. Тогда при переполнении справочника требуется только его переделка, что вызывает меньше накладных расходов.

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

Каким же образом можно обеспечить наличие точек физической согласованности базы данных, т.е. как восстановить состояние базы данных в момент tpc? Для этого используются два основных подхода: подход, основанный на использовании теневого механизма, и подход, в котором применяется журнализация постраничных изменений базы данных.

При открытии файла таблица отображения номеров его логических блоков в адреса физических блоков внешней памяти считывается в оперативную память. При модификации любого блока файла во внешней памяти выделяется новый блок. При этом текущая таблица отображения (в оперативной памяти) изменяется, а теневая - сохраняется неизменной. Если во время работы с открытым файлом происходит сбой, во внешней памяти автоматически сохраняется состояние файла до его открытия. Для явного восстановления файла достаточно повторно считать в оперативную память теневую таблицу отображения.

Общая идея теневого механизма показана на следующем рисунке:



В контексте базы данных теневой механизм используется следующим образом. Периодически выполняются операции установления точки физической согласованности базы данных (checkpoints в System R). Для этого все логические операции завершаются, все буфера оперативной памяти, содержимое которых не соответствует содержимому соответствующих страниц внешней памяти, выталкиваются. Теневая таблица отображения файлов базы данных заменяется на текущую (правильнее сказать, текущая таблица отображения записывается на место теневой).

Восстановление к tlpc происходит мгновенно: текущая таблица отображения заменяется на теневую (при восстановлении просто считывается теневая таблица отображения). Все проблемы восстановления решаются, но за счет слишком большого перерасхода внешней памяти. В пределе может потребоваться вдвое больше внешней памяти, чем реально нужно для хранения базы данных. Теневой механизм - это надежное, но слишком грубое средство.

Обеспечивается согласованное состояние внешней памяти в один общий для всех объектов момент времени. На самом деле, достаточно иметь набор согласованных наборов страниц, каждому из которых может соответствовать свой набор времени.

Для достижения такого более слабого требования наряду с логической журнализацией операций изменения базы данных производится журнализация постраничных изменений. Первый этап восстановления после мягкого сбоя состоит в постраничном откате незакончившихся логических операций. Подобно тому, как это делается с логическими записями по отношению к транзакциям, последней записью о постраничных изменениях от одной логической операции является запись о конце операции. Для того, чтобы распознать, нуждается ли страница внешней памяти базы данных в восстановлении, при выталкивании любой страницы из буфера оперативную память в нее помещается идентификатор последней записи о постраничном изменении этой страницы. Имеются и другие технические нюансы.

В этом подходе имеются два поднаправления. В первом поднаправлении поддерживается общий журнал логических и страничных операций. Естественно, наличие двух видов записей, интерпретируемых абсолютно по-разному, усложняет структуру журнала. Кроме того, записи о постраничных изменениях, актуальность которых носит локальный характер, существенно (и не очень осмысленно) увеличивают журнал.

Поэтому все более популярным становится поддержание отдельного (короткого) журнала постраничных изменений. Такая техника применяется, например, в известном продукте Informix Online.

Лекция 9

По технологии обработки данных базы данных подразделяются на централизованные и распределенные.

Централизованная база данных хранится в памяти одной вычислительной системы. Эта вычислительная система может быть мэйнфреймом - тогда доступ к ней организуется с использованием терминалов - или файловым сервером локальной сети ПК.

Распределенная база данных состоит из нескольких, возможно, пересекающихся или даже дублирующих друг друга частей, которые хранятся в различных ЭВМ вычислительной сети. Работа с такой базой осуществляется с помощью системы управления распределенной базой данных (СУРБД).

По способу доступа к данным базы данных разделяются на **базы данных с локальным доступом** и **базы данных с сетевым доступом**.

Для всех современных баз данных можно организовать сетевой доступ с многопользовательским режимом работы.

Централизованные базы данных с сетевым доступом могут иметь следующую архитектуру:

- файл-сервер;
- клиент-сервер базы данных;
- "тонкий клиент" - сервер приложений - сервер базы данных (трехуровневая архитектура).

Рис. 1. Схема работы с БД в локальной сети с выделенным файловым сервером

Файл-сервер. Архитектура систем БД с сетевым доступом предполагает выделение одной из машин сети в качестве центральной (файловый сервер). На

этот компьютер устанавливается операционная система (ОС) для выделенного сервера (например, Microsoft Windows Server 2003). На нем же хранится совместно используемая централизованная БД в виде одного или группы файлов. Все другие компьютеры сети выполняют функции рабочих станций (могут работать в ОС Microsoft Windows 2000 Professional или Microsoft Windows 98). Файлы базы данных в соответствии с пользовательскими запросами передаются на рабочие станции, где и производится обработка информации (рис. 2.). При большой интенсивности доступа к одним и тем же данным производительность информационной системы падает. Пользователи могут создавать также локальные БД на рабочих станциях.

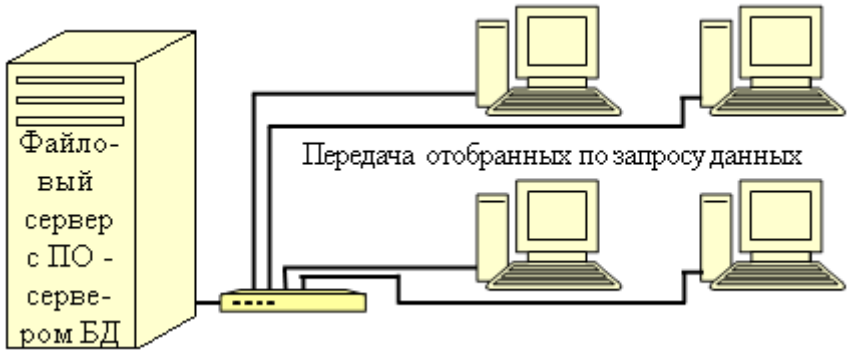


Рис. 2. Схема работы с БД в архитектуре "Клиент-сервер"

Клиент-сервер. В этой архитектуре на выделенном сервере, работающем под управлением серверной операционной системы, устанавливается специальное программное обеспечение (ПО) - сервер БД, например, Microsoft®SQL Server™или Oracle. СУБД подразделяется на две части: клиентскую и серверную. Основа работы сервера БД - использование языка запросов (SQL). Запрос на языке SQL, передаваемый клиентом (рабочей станцией) серверу БД, порождает поиск и извлечение данных на сервере. Извлеченные данные транспортируются по сети от сервера к клиенту (рис. 2). Тем самым, количество передаваемой по сети информации уменьшается во много раз.

Трехуровневая архитектура функционирует в Интранет- и Интернет-сетях. Клиентская часть ("тонкий клиент"), взаимодействующая с

пользователем, представляет собой HTML-страницу в Web-браузере либо Windows-приложение, взаимодействующее с Web-сервисами. Вся программная логика вынесена на сервер приложений, который обеспечивает формирование запросов к базе данных, передаваемых на выполнение серверу баз данных. Сервер приложений может быть Web-сервером или специализированной программой (например, Oracle Forms Server) (рис. 3).

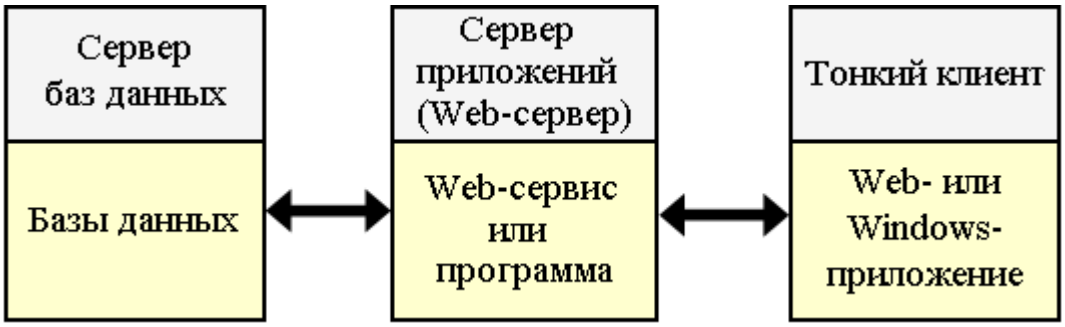


Рис. 3. Схема работы с БД в трехуровневой архитектуре

Язык для взаимодействия с БД SQL появился в середине 70-х и был разработан в рамках проекта экспериментальной реляционной СУБД System R. Исходное название языка SEQUEL (Structered English Query Language) только частично отражает суть этого языка. Конечно, язык был ориентирован главным образом на удобную и понятную пользователям формулировку запросов к реляционной БД, но на самом деле уже являлся полным языком БД, содержащим помимо операторов формулирования запросов и манипулирования БД средства определения и манипулирования схемой БД; определения ограничений целостности и триггеров; представлений БД; возможности определения структур физического уровня, поддерживающих эффективное выполнение запросов; авторизации доступа к отношениям и их полям; точек сохранения транзакции и откатов. В языке отсутствовали средства синхронизации доступа к объектам БД со стороны параллельно выполняемых транзакций: с самого начала предполагалось, что необходимую синхронизацию неявно выполняет СУБД.

Рассмотрим эти свойства языка немного более подробно.

Запросы и операторы манипулирования данными

Как известно, двумя фундаментальными языками запросов к реляционным БД являются языки реляционной алгебры и реляционного исчисления. При всей своей строгости и теоретической обоснованности эти языки редко используются в современных реляционных СУБД в качестве средств пользовательского интерфейса. Запросы на этих языках трудно формулировать и понимать. SQL представляет собой некоторую комбинацию реляционного исчисления кортежей и реляционной алгебры, причем до сих пор нет общего согласия, к какому из классических языков он ближе. При этом возможности SQL шире, чем у этих базовых реляционных языков, в частности, в общем случае невозможна трансляция запроса, сформулированного на SQL, в выражение реляционной алгебры, требуется некоторое ее расширение.

Существенными свойствами подъязыка запросов SQL являются возможность простого формулирования запросов с соединениями нескольких отношений и использование вложенных подзапросов в предикатах выборки. Вообще говоря, одновременное наличие обоих средств избыточно, но это дает пользователю при формулировании запроса возможность выбора более понятного ему варианта.

В предикатах со вложенными подзапросами в SQL System R можно употреблять теретико-множественные операторы сравнения, что позволяет формулировать квантифицированные запросы (эти возможности обычно труднее всего понимаются пользователями и поэтому в дальнейшем в SQL появились явно квантифицируемые предикаты).

Существенной особенностью SQL является возможность указания в запросе потребности группирования отношения-результата по указанным полям с поддержкой условий выборки на всю группу целиком. Такие условия выборки могут содержать агрегатные функции, вычисляемые на группе. Эта возможность SQL главным образом отличает этот язык от языков реляционной алгебры и реляционного исчисления, не содержащих аналогичных средств.

Еще одним отличием SQL является необязательное удаление кортежей-дубликатов в окончательном или промежуточных отношениях-результатах. Строго говоря, результатом оператора выборки в языке SQL является не отношение, а мультимножество кортежей. В тех случаях, когда семантика запроса требует наличия отношения, уничтожение дубликатов производится неявно.

Самый общий вид запроса на языке SQL представляет теоретико-множественное алгебраическое выражение, составленное из элементарных запросов. В SQL System R допускались все базовые теретико-множественные операции (UNION, INTERSECT и MINUS).

Работа с неопределенными значениями в SQL System R до конца продумана не была, хотя неявно предполагалось использование трехзначной логики при вычислении логических выражений.

Операторы манипулирования данными UPDATE и DELETE построены на тех же принципах, что и оператор выборки данных SELECT. Набор кортежей указанного отношения, подлежащих модификации или удалению, определяется входящим в соответствующий оператор логическим выражением, которое может включать сложные предикаты, в том числе и с вложенными подзапросами.

В операторе вставки кортежа(ей) в указанное отношение заносимый кортеж может задаваться как в литеральной форме, так и с помощью внутреннего подоператора выборки.

Операторы определения и манипулирования схемой БД

В число операторов определения схемы БД SQL System R входили операторы создания и уничтожения постоянных и временных хранимых отношений (CREATE TABLE и DROP TABLE) и создания и уничтожения представляемых отношений (CREATE VIEW и DROP VIEW). В языке и в реализации System R не запрещалось использовать операторы определения

схемы в пределах транзакции, содержащей операторы выборки и манипулирования данными. Допускалось, например, использование операторов выборки и манипулирования данными, в которых указываются отношения, не существующие в БД к моменту компиляции оператора. Конечно, эта возможность существенно усложняла реализацию и требовалась по существу очень редко.

Оператор манипулирования схемой БД ALTER TABLE позволял добавлять указываемые поля к существующим отношениям. В описании языка определялось, что выполнение этого оператора не должно приводить к недействительности ранее откомпилированных операторов над отношением, схема которого изменяется, и что значения вновь определенных полей в существующих кортежах отношения становятся неопределенными.

Определения ограничений целостности и триггеров

Язык SQL System R включал очень мощные средства контроля и поддержания целостности БД. Средства контроля базировались на аппарате ограничений целостности (ASSERTIONS). Фактически, ограничение целостности - это логическое выражение, вычисляемое над текущим состоянием БД, ложность которого соответствует нецелостному состоянию БД. Логическое выражение ограничения целостности могло содержать любой допустимый в языке предикат.

Более точно, ограничения целостности делились на два класса: проверяемые после выполнения оператора манипулирования данными и проверяемые при завершении транзакции или при выполнении специального оператора INFORCE INTEGRITY. Типы предикатов, которые можно использовать в операторах определения ограничений целостности разных классов, различаются. В операторах первого класса проверяется, фактически, текущий кортеж, с которым производится манипулирование. Во втором случае проверяются указанные в ограничении целостности отношения, т.е. все их кортежи. Различается и определяемая в языке реакция системы на нарушения

ограничений целостности разных классов. В первом случае нарушение ограничения целостности приводит к откату транзакции в точку, непосредственно предшествующую операции манипулирования данными, выполнение которого вызвало нарушение ограничения целостности. Во втором случае ограничение приводит к полному откату транзакции к ее началу.

Очень важным механизмом, определенным в языке SQL System R, является механизм триггеров. В контексте System R этот механизм рассматривался главным образом как средство автоматического поддержания целостности БД. При определении триггера указывалось условие проверки его применимости (имя отношения и тип операции манипулирования данными), условие применимости триггера (логическое выражение, построенное по правилам, близким к правилам для ограничений целостности первого класса) и действие, которое должно быть выполнено над БД в случае истинности условия применимости. Такое действие могло быть выражено с помощью произвольного оператора манипулирования данными. Во время выполнения действия могли срабатывать другие триггеры и т.д.

Механизмы ограничений целостности и триггеров System R являлись очень мощными и общими, но реализация их очень трудна и накладна (как уже отмечалось, триггеры так и не были реализованы в System R). Дополнительную сложность в реализации создавал тот факт, что допускалось (по крайней мере не запрещалось языком) определение ограничений целостности и триггеров в пределах той же транзакции, в которой выполняются операторы манипулирования данными. При наиболее полной реализации требовалось бы большое число дополнительных действий во время выполнения транзакции. Кроме того, в ряде случаев отсутствие зафиксированной семантики соответствующих конструкций языка приводило к неоднозначному пониманию выполнения транзакций.

Представления базы данных

В языке допускалось использование хранимых отношений БД и представляемых отношений. Наиболее удачным решением было использование для определения представлений общего аппарата операторов выборки. Любой оператор выборки может быть использован для определения представления.

В языке отсутствуют какие-либо ограничения по поводу использования представлений: в любом операторе SQL, в котором допускается использование имени хранимого отношения, допускается и использование имени представления. В SQL System R ничего не говорится о рекомендуемом способе реализации доступа к представлениям, но при любом способе эффект должен быть таким, как если бы выполнить полную материализацию представления до выполнения оператора.

Массу проблем, исследований и предложений породила потенциальная возможность выполнения операторов манипулирования данными над представлениями. Понятно, что эта возможность легко реализуема для простых представлений, но в более сложных случаях не только реализация, но и семантика операций становится нетривиальной. Кстати, в System R операторы манипулирования данными допускались только над простыми представлениями.

Определение управляющих структур

Внесение в реляционный язык, каким является SQL, явных операторов порождения и уничтожения структур физического уровня, поддерживающих эффективное выполнение запросов к БД, явилось в SQL System R чисто прагматическим решением, обеспечивающим возможность всех видов работ с БД с помощью одного языка.

В SQL System R упоминаются два вида таких структур: индексы и связи (links). Индекс в его абстрактном языковом представлении - это инвертированный файл, обеспечивающий доступ к кортежам

соответствующего отношения на основе заданных значений одного или нескольких столбцов, составляющих ключ индекса. Операторы языка позволяли создавать и уничтожать индексы, но никаким образом не давали возможности явно указать на необходимость использования существующего индекса при выполнении оператора выборки, решение об этом возлагалось на реализацию.

С помощью оператора определения индекса можно было выразить два дополнительных утверждения, касающихся логической схемы отношения и физической структуры его хранения. Использование при определении индекса ключевого слова UNIQUE означало, что ключ этого индекса является возможным ключом соответствующего отношения. Фактически это означает наличие дополнительного механизма определения ограничения целостности отношения. Один из индексов для данного отношения мог быть определен с ключевым словом CLUSTERING. Это означает требование физической кластеризации во внешней памяти кортежей отношения с равными или близкими значениями ключа индекса.

Операторы определения связи позволяли в стиле сетевой модели данных организовать во внешней памяти списки кортежей указанного отношения. Как и в случае индексов, операторы позволяли создавать и уничтожать такие списки, но не давали возможности явно указать на необходимость использования существующих списков при выполнении операторов выборки. Большая трудоемкость поддержания списков при выполнении операторов манипулирования данными и трудность выполнения оценок стоимости их использования при выполнении операторов выборки привели к тому, что механизм связей исчез из языка уже на поздней стадии проекта System R. С тех пор этот механизм, насколько нам известно, не появлялся ни в одном варианте SQL.

Авторизация доступа к отношениям и их полям

Существенной особенностью языка SQL, появившейся в нем с самого начала, является обеспечение защиты доступа к данным средствами самого языка. Основная идея такого подхода состоит в том, что по отношению к любому отношению БД и любому столбцу отношения вводится предопределенный набор привилегий. С каждой транзакцией неявно связывается идентификатор пользователя, от имени которого она выполняется (способы связи и идентификации пользователей не фиксируются в языке и определяются в реализации).

После создания нового отношения все привилегии, связанные с этим отношением и всеми его столбцами, принадлежат только пользователю-создателю отношения. В число привилегий входит привилегия передачи всех или части привилегий другому пользователю, включая привилегию на передачу привилегий. Технически передача привилегий осуществляется при выполнении оператора SQL GRANT. Существует также привилегия изъятия всех или части привилегий у пользователя, которому они ранее были переданы. Эта привилегия также может передаваться. Технически изъятие привилегий происходит при выполнении оператора SQL REVOKE.

Проверка полномочности доступа к данным происходит на основе информации о полномочиях, существующих во время компиляции соответствующего оператора SQL. Подобно тому, что мы отмечали в связи с ограничениями целостности и триггерами, в SQL System R отсутствовали какие-либо ограничения по поводу использования операторов GRANT и REVOKE. Это приводило к существенным техническим затруднениям в реализации, а иногда к неоднозначному пониманию поведения.

Долгое время подход к защите данных от несанкционированного доступа принимался практически без критики, однако в связи с распространяющимся использованием реляционных СУБД в нетрадиционных приложениях все чаще раздается критика. Если, например, в системе БД должна поддерживаться

многоуровневая защита данных, соответствующую систему полномочий весьма трудно, а иногда и невозможно построить на основе средств SQL.

Точки сохранения и откаты транзакции

В SQL System R существовали два специальных оператора для установки так называемых точек сохранения транзакции и для отката транзакции к ранее установленной точке сохранения. В литературе, относящейся к System R, обсуждение этих возможностей практически не содержится, из чего неявно следует, что они не были реализованы.

Прямолинейная реализация этого механизма не вызывает особых технических затруднений, но и не очень полезна, потому что после выполнения частичного отката транзакции для успешного продолжения работы прикладной программы потребовалось бы и восстановить ее состояние в соответствующей точке, а это никак не поддерживается. Понятно, что при более тщательной проработке должны быть увязаны механизмы точек сохранения и контроля целостности. Например, было бы естественно, чтобы при выполнении оператора ENFORCE INTEGRITY, если какие-либо ограничения целостности нарушаются, происходил автоматический откат транзакции к ближайшей точки сохранения, в которой нарушения целостности БД не было. Это значительно усложнило бы реализацию, но было бы очень полезно. Аналогично, можно было бы использовать механизм точек сохранения при автоматических откатах транзакций по причине возникновения синхронизационных тупиков.

Отметим еще два важных свойства языка SQL System R, которые в разных видах присутствуют во всех развитых последующих вариантах языка.

Встроенный SQL

В SQL System R присутствуют специальные операторы, поддерживающие встраивание операторов SQL в традиционные языки программирования (в System R основным таким языком был PL/1).

Основная проблема встраивания SQL в язык программирования состояла в том, что SQL - реляционный язык, т.е. его операторы большей частью работают со множествами, в то время как в языках программирования основными являются скалярные операции. Решение SQL состоит в том, что в язык дополнительно включаются операторы, обеспечивающие покортежный доступ к результату запроса к БД.

Для этого в язык вводится понятие курсора, с которым связывается оператор выборки. Над определенным курсором можно выполнять оператор OPEN, означающий материализацию отношения-результата запроса, оператор FETCH, позволяющий выбрать очередной кортеж результирующего отношения в память программы, и оператор CLOSE, означающий конец работы с данным курсором.

Дополнительную гибкость при создании прикладных программ со встроенным SQL обеспечивает возможность параметризации операторов SQL значениями переменных включающей программы.

Динамический SQL

Для упрощения создания интерактивных SQL-ориентированных систем в SQL System R были включены операторы, позволяющие во время выполнения транзакции откомпилировать и выполнить любой оператор SQL.

Оператор PREPARE вызывает динамическую компиляцию оператора SQL, текст которого содержится в указанной переменной символьной строке включающей программы. Текст может быть помещен в переменную при выполнении программы любым допустимым способом, например, введен с терминала.

Оператор DESCRIBE служит для получения информации об указанном операторе SQL, ранее подготовленном с помощью оператора PREPARE. С помощью этого оператора можно узнать, во-первых, является ли подготовленный оператор оператором выборки, и во-вторых, если это оператор

выборки, получить полную информацию о числе и типах столбцов результирующего отношения.

Для выполнения ранее подготовленного оператора SQL, не являющегося оператором выборки, служит оператор EXECUTE. Для выполнения динамически подготовленного оператора выборки используется аппарат курсоров с некоторыми отличиями по части задания адресов переменных включающей программы, в которые должны быть помещены значения столбцов текущего кортежа результата.

Подводя итог приведенному краткому описанию основных черт SQL System R, отметим, что несмотря на недостаточную техническую проработку, в идейном отношении язык содержал все необходимые средства, позволяющие использовать его как базовый язык СУБД.

Курсовая работа (30 часов).

В качестве курсовой работы по дисциплине «Организация баз данных» студенты разрабатывают и реализовывают с помощью современной СУБД базу данных в предложенной им предметной области. Основные разделы, которые должны быть отражены в курсовой работе:

–описание предметной области;

–исследование информационных потребностей пользователей;

–инфологическое проектирование;

–логическое проектирование;

–физическое проектирование (реализация в СУБД по выбору);

–программный продукт;

–руководство пользователя;

–аппаратные и программные требования.

Темы курсовых работ:

1. Паспортный стол
2. Автомобильный магазин
3. Библиотека
4. Отдел сбыта и маркетинга ОАО Кондитерская фабрика "Зея"
5. Учет преступников
6. Деканат
7. Отдел налоговой полиции
8. Школа
9. Отдел кадров
10. Учет административных нарушений
11. Факультет дистанционного обучения
12. Музыкальный магазин
13. Регистрация транспортных средств
14. Детский сад
15. Отдел управления фирмы "Фармация"

16. Сведения об абитуриентах
17. Складской учет
18. Фирма по продаже компьютерного оборудования
19. Отдел аспирантуры и докторантуры
20. Поликлиника
21. Страхование
22. Станция технического обслуживания "Амур-Лада"
23. Гостиница
24. Ресторан

Самостоятельная работа студентов (6 часов).

В течение семестра студентами должны быть самостоятельно изучены следующие вопросы и подготовлен реферат по заданной теме:

1. Функции, архитектура распределенных БД.
2. Преимущества и недостатки распределенных БД.
3. Фундаментальный принцип, свойства распределенных БД.
4. Технология клиент-сервер.
5. Связь объектно-ориентированных СУБД с общими понятиями объектно-ориентированного подхода.
6. Объектно-ориентированные модели данных.
7. Характеристики, достоинства и недостатки объектно-ориентированных СУБД.
8. Языки программирования объектно-ориентированных СУБД.
9. Языки запросов объектно-ориентированных СУБД.
10. Манифесты БД.
11. Характеристики объектно-реляционных СУБД.
12. Достоинства и недостатки объектно-реляционных СУБД.
13. Сравнительная характеристика объектно-ориентированных и объектно-реляционных СУБД.
14. Требования, предъявляемые к интеграции СУБД в среду Web.

15. Архитектура Web-СУБД.
16. Преимущества и недостатки интеграции СУБД в Web.
17. Основные методы интеграции СУБД в среду Web.
18. Безопасность Web-СУБД.

Вопросы к экзамену

1. Информация и данные. Базы и банки данных. Предметная область банка данных.
2. Базы данных в составе автоматизированных систем.
3. Компоненты системы БД
4. Функции приложения БД
5. Функции СУБД
6. Преимущества и недостатки СУБД
7. Архитектура системы БД
8. Администратор БД. Функции администратора БД.
9. Этапы проектирования БД
10. Инфологический подход к проектированию систем БД
11. Модель «сущность-связь»
12. Моделирование локальных представлений.
13. Объединение моделей локальных представлений.
14. Иерархическая модель системы.
15. Сетевая модель системы.
16. Реляционная модель данных.
17. Реляционная алгебра.
18. Системы реляционного исчисления.
19. Логическое проектирование БД
20. Установление дополнительных логических связей.
21. Отображение концептуальной инфологической модели на реляционную модель.
22. Нормализация отношений.
23. Физическое проектирование БД

24.Статическое и динамическое хеширование.

Виды контроля.

Текущий контроль за аудиторной и самостоятельной работой обучаемых осуществляется во время проведения аудиторных занятий посредством устного опроса, проведения контрольных работ или осуществления лекции в форме диалога. Промежуточный контроль осуществляется два раза в семестр в виде анализа разработанных схем по инфологическому и датологическому проектированию. Итоговый контроль осуществляется после успешного прохождения студентами текущего и промежуточного контроля в виде устного или письменного экзамена при ответах экзаменуемого на два вопроса в билете и дополнительные вопросы по желанию экзаменатора.

Критерии оценки

Студент, сдающий экзамен по данному предмету, должен показать знания по архитектуре и функциям БД, знать последовательность и содержание этапов проектирования БД, основные модели данных и конструкции языков манипулирования данными, разбираться в современном программном обеспечении систем управления базами данных.

Знания студента оцениваются как отличные при полном изложении теоретического материала экзаменационного билета и ответах на дополнительные вопросы со свободной ориентацией в материале и других литературных источниках.

Оценка “хорошо” ставится при твердых знаниях студентом всех разделов курса, но в пределах конспекта лекций и обязательных заданий по самостоятельной работе с литературой.

Оценку «удовлетворительно» студент получает, если дает неполные ответы на теоретические вопросы билета, показывая поверхностное знание учебного материала, владение основными понятиями и терминологией; при неверном ответе на билет ответы на наводящие вопрос.

Оценка «неудовлетворительно» выставляется за незнание студентом одного из разделов курса. Студент не дает полные ответы на теоретические

вопросы билета, показывая лишь фрагментарное знание учебного материала, незнание основных понятий и терминологии; наводящие вопросы остаются без ответа.

Для допуска к экзамену студент должен сдать все лабораторные работы и реферат по самостоятельной работе.

Рекомендуемая литература

Перечень обязательной (основной) литературы

1.Дейт К. Введение в системы баз данных: Учебное пособие. – М.: Вильямс, 2001.- 1072 с.

2. Коннолли Т., Бегг К., Страчан А. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. – М.: Вильямс, 2000.- 1120 с.

3.Кренке Д.М. Теория и практика построения баз данных: Учебное пособие. – СПб.: Питер, 2005. – 786 с.

4. Харитонова И. Самоучитель: Office Access 2003. – СПб., Питер, 2004. – 464 с.

5. Харитонова И., Вольман Н. Программирование в Access 2002: учебный курс. – СПб., Питер, 2002. – 480 с.

6.Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для высших учебных заведений - СПб.: Корона принт. -2004. - 736 с.

7. Ульман Дж., Уидом Дж. Введение в системы баз данных. - М.: Издательство "Лори". - 2000, 374 с.

8.Федоров А.П. Базы данных для всех. - М.: Компьютер пресс, 2001.- 256 с.

Перечень дополнительной литературы

1.Вейскас Д. Эффективная работа с Microsoft Access 2003: учебный курс. – СПб., Питер, 2004. – 398 с.

2.Робинсон С. Microsoft Access 2000: учебный курс. – СПб., Питер, 2001. – 476 с.

3.Хансен Г., Хансен Дж. Базы данных: разработка и управление. М.: Бином, 1999, 704 с.

4.Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н. Базы и банки данных и знаний: Учебник для вузов по специальности "АСУ" - М.: Высшая школа, 1992 г. - 367 с.