В.И. Швецов

БАЗЫ ДАННЫХ

2008

УДК 681.3
ББК 32.97

ОБЩАЯ ИНФОРМАЦИЯ О КУРСЕ

Курс «Базы данных»

Краткая аннотация: Университетский курс, формирующий концептуальные представления о принципах построения БД и СУБД, представляющий фундаментальные понятия и математические модели, лежащие в основе БД и СУБД, принципы проектирования БД, а также технологии реализации БД и иллюстрирующий вышеуказанные понятия на примере ACCES и MS SQL-Server.

Подробное описание: Учебное пособие посвящено важнейшей составляющей широко разрабатываемых и используемых информационных систем организационного управления – базам данных (БД), создаваемым и функционирующим на основе систем управления базами данных (СУБД).

Главной целью пособия является формирование концептуальных представлений об основных принципах построения БД и СУБД, принципах проектирования БД, а также анализ основных технологий реализации БД. Особое внимание уделяется представлению фундаментальных понятий и математических моделей, лежащих в основе баз данных и систем управления базами данных.

Изучение курса включает усвоение ряда фундаментальных понятий и теоретических основ организации баз данных и систем управления базами данных:

- тенденции развития основных понятий представления данных и интегрирования данных;
- программный интерфейс между пользователями и базой данных – СУБД;
- модели организации работы пользователей с базой данных;
- моделирование базы данных (моделирование внешних представлений, концептуальное моделирование, моделирование структур хранения);
- особенности реляционного моделирования;
- реализация языка запросов к базам данных (SQL).

В **задачи** курса входит изучение процесса проектирования базы данных, включающего:

- составление формализованного описания предметной области (внешней модели);
- разработку концептуальной модели и ее специфицирование к конкретной модели данных СУБД;
- анализ моделей физического представления данных.

Рассмотрение указанных вопросов иллюстрируется на примерах конкретных систем управления базами данных – ACCES и MS SQL-Server.

Предисловие

Последние десятилетия в области программирования характеризуются резким ростом количества создаваемых информационных систем организационного управления. Практически в каждой организации функционирует (или создается) такая система (или её элементы). Важнейшей структурной частью информационных систем являются базы данных, создаваемые и функционирующие на основе использования специализированных программных систем – систем управления базами данных. Все это обусловливает большую потребность в квалифицированных кадрах, способных как создавать информационные системы на основе систем управления базами данных, так и обслуживать соответствующие информационные системы и базы данных.

Цель данного учебного пособия состоит в формировании концептуальных представлений об основных принципах построения баз данных, систем управления базами данных; о математических моделях, описывающих базу данных; о принципах проектирования баз данных; а также анализе основных технологий реализации баз данных.

Тематика, связанная с базами данных, чрезвычайно широка. Можно указать, в качестве примера, целый ряд возможных семестровых или годовых курсов по соответствующей тематике: введение в базы данных, проектирование баз данных, реляционные базы данных, язык запросов SQL, клиент-серверные системы, работа в среде конкретной СУБД и т. п. В связи с эти невозможно в одном курсе детально раскрыть все стороны этой тематики. В то же время очень важно дать читателю достаточно полное представление об общей структуре тематики баз данных и важнейших понятиях в этой области.

Главной задачей настоящей книги является представление читателю фундаментальных понятий, лежащих в основе баз данных и систем управления базами данных, и иллюстрация способов реализации соответствующих понятий в конкретных программных системах.

Рассмотрение указанных вопросов иллюстрируется на примерах конкретных систем управления базами данных – ACCES и MS SQL-Server..

Пособие разработано с учетом международных рекомендаций по стандартизации обучения информатике в университетах Computing Curricula 2001 (совместная раработка Компьютерного общества Института инженеров по электротехнике и электронике (IEEE-CS) и Ассоциации по вычислительной технике (ACM)) и включает, в соответствии с этими рекомендациями, основную совокупность знаний по Управлению информацией (разделы IM2 – IM9). Структура пособия соответствует структуре курса CS270T. «Базы данных» из вышеуказанных рекомендаций.

Цель курса: Цель данного курса состоит в формировании концептуальных представлений об основных принципах построения баз данных, систем управления базами данных; о математических моделях, описывающих базу данных; о принципах проектирования баз данных; а также анализе основных технологий реализации баз данных.

Главной задачей учебного курса является представление слушателю фундаментальных понятий, лежащих в основе баз данных и систем управления базами данных, и иллюстрация способов реализации соответствующих понятий в конкретных программных системах.

В задачи курса входит изучение процесса проектирования базы данных, включающее формализацию описания предметной области, (разработку концептуальной модели и ее специфицирование к конкретной модели данных СУБД.

Рассмотрение указанных вопросов иллюстрируется на примерах конкретных систем управления базами данных – ACCES и MS SQL-Server.

Предварительные знания

Курс «Базы данных» опирается на материалы следующих курсов:

Основы построения ЭВМ;

ЭВМ и программирование;

Дискретная математика.

Автор: Швецов Владимир Иванович, доктор технических наук, профессор, проректор по информатизации ГОУ ВПО «Нижегородский государственный университет им. Н.И.Лобачевского».

Профессиональные интересы: проектирование и создание баз данных, разработка программных систем обработки данных для конкретных классов задач.

Лекция 1. Введение в базы данных. Общая характеристика основных понятий

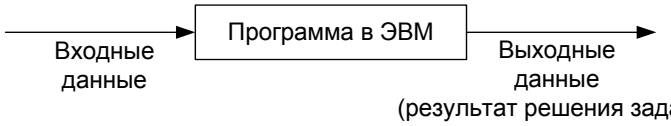
Лекция посвящена рассмотрению развития основных понятий обработки данных, связанного с постоянным расширением классов решаемых на ЭВМ задач. Показывается необходимость интеграции данных при решении несколькими пользователями задач, использующих общие данные. Вводится понятие базы данных.

Ключевые термины: представление данных, элемент данных, логическая запись, экземпляр записи, логический файл, интеграция данных, база данных, система управления базами данных, СУБД, функции СУБД.

Цель лекции: показать, что с изменением вида решаемых на ЭВМ задач в программировании возникают новые виды представления данных, в том числе такой вид, как базы данных.

1.1. Развитие основных понятий представления данных

Любой вычислительный процесс представляет собой отображение (по определенному алгоритму) некоторых входных данных в выходные.



Соотношение сложности **представления обрабатываемых данных** и алгоритма вычислений определяет два класса задач:

- вычислительные задачи – достаточно простое представление данных и сложный, многооперационный процесс вычислений;
- задачи обработки данных (невычислительные задачи) – простой алгоритм обработки данных и сложное **представление обрабатываемых данных**.

На начальной стадии обучения программированию основное внимание уделяется разработке алгоритма решения задачи. Однако часто оказывается, что возможность (или невозможность) решения конкретной задачи зависит не только от выбранного алгоритма, но и от того, какие понятия используются для представления обрабатываемых данных.

Рассмотрим простейший пример вычисления по формуле:

$$Y = X^2 + 5X,$$

где X и Y – определенные числа, которые являются здесь элементарными единицами данных (элементами данных).

При программировании алгоритма решения этой задачи (программирование формулы) используется простейший вид данных – простая переменная (X и Y представляются в программе простыми переменными). Заметим, что простая переменная в системах программирования характеризуется определенным типом ее значений, которые должны выбираться при программировании. Даже в этом простейшем случае необходимо правильно выбрать тип переменной, причем от этого выбора может зависеть возможность или невозможность решения

конкретной прикладной задачи (например, для представления конкретных данных не хватит отведенных разрядов).

Рассмотрим другой пример:

$$S = a_1 + a_2 + ... + a_N .$$

Решение этой задачи в общем случае невозможно получить используя только простые переменные. Здесь обрабатывается не отдельное число, а последовательность чисел. В этом случае при программировании используется такой вид данных, как массив – совокупность элементов, с каждым из которых связан упорядоченный набор целых чисел, называемых индексами. Все элементы должны иметь одинаковый тип их значений, который и будет типом массива. В этом случае числа a_1, a_2,…, a_N представляются в программе массивом $A(1)$, $A(2)$,..., $A(N)$.

Приведенные примеры показывают, что изменение вида задач обусловливает необходимость использования других видов данных.

Ранние языки программирования (ФОРТРАН, АЛГОЛ-60) были предназначены для решения научно-технических вычислительных задач. В этих языках использовались только вышеуказанные виды данных (простые переменные и массивы) что было вполне достаточно.

Начиная с конца 60-х годов компьютеры начинают интенсивно использоваться для решения так называемых невычислительных задач, связанных с обработкой различного рода документов. Рассмотрим появление новых видов данных на примере упрощенных задач обработки данных.

Задача 1. Начисление заработной платы.

Рассматриваем задачу при двух упрощающих предположениях:

- сотруднику начисляется заработная плана на основе его оклада;
- никакие налоги и вычеты не учитываются.

Необходимые для решения этой задачи сведения о сотруднике представлены в следующей карточке НАЧИСЛЕНИЕ:

Фамилия, имя, отчество *FIO*	Оклад *O*	Количество отработанных дней в месяце *K*o	Начисленная сумма *S*

Для каждого работника начисленная сумма за определенный месяц рассчитывается по следующей формуле:

$$S = K_o O / K_r ,$$

где K_r – количество рабочих дней в данном месяце.

Для каждого сотрудника соответствующие данные имеют конкретное значение, например:

Иванов Иван Иванович	1800	24	1800

Эти значения имеют смысл только во взаимосвязи друг с другом. Отдельно выбранное число 1800 теряет свой содержательный смысл, поэтому использовать такой вид данных, как простая переменная, здесь нельзя. В то же время набор соответствующих значений, характеризующих конкретного сотрудника, имеет разные типы (символьный и числовой), т.е. использовать для его представления такой вид данных, как массив, также нельзя. Таким образом, понятий «простая переменная» и «массив» недостаточно, чтобы представить соответствующую карточку.

Для описания аналогичных представлений данных в предметной области невычислительных задач вводится ряд новых понятий [1].

Элемент данных (поле) – наименьшая единица поименованных данных.

Для данного примера элементами данных являются *FIO*, *O*, *K*о, *S*.

Для описания карточки сотрудника используется понятие «Логическая запись».

Логическая запись – поименованная совокупность элементов данных (полей).

Экземпляр логической записи – текущее значение элементов записи.

Для представления всего набора карточек сотрудников используется понятие **«Логический файл»**

Логический файл - поименованная совокупность всех экземпляров записей заданного типа.

Пример логического файла НАЧИСЛЕНИЕ:

Иванов Иван Иванович	1800	24	1800

Петров Петр Петрович	2200	20	1830

– –

Сидоров Сидор Сидорович	2500	24	2500

Таким образом, с помощью введенных понятий можно описывать соответствующие данные. Для отображения этих понятий в современных языках программирования, предназначенных как для вычислительных задач, так и для задач обработки данных, введены новые виды данных.

В алгоритмическом языке Паскаль вводится такой вид данных, как запись (RECORD) – сложная переменная с несколькими компонентами, которые могут иметь разные типы. Кроме того, доступ к компонентам записи (полям) осуществляется не по индексу, а по имени. При программировании задачи 1 на языке Паскаль логическая запись НАЧИСЛЕНИЕ представляется видом данных RECORD, набор экземпляров логических записей сотрудников представляется (логический файл) представляется «физическим» файлом, формируемым средствами языка Паскаль и операционной системы..

```
Salary = RECORD
      FIO:string;
      O:  real;
      Ko: real;
      S:  real;
END;
```

Отметим важную специфику таких невычислительных задач. Для этих задач характерны большие объемы данных (большое количество сотрудников, большое количество производимых изделий и т. п.). Указанные данные, как правило, используются для решения задачи многократно (зарплата начисляется постоянно каждый месяц), поэтому данные должны достаточно долго храниться в памяти ЭВМ. Для длительного хранения всегда используется внешняя память.

В связи с этим решение задачи 1 состоит из двух этапов.

1. Ввод исходных данных и занесение их во внешнюю память.

```
type
Salary = RECORD
FIO:   string;
O:     real;
Ko:    real;
S:     real;
END;
FSalary = File of Salary;
var
  F: FSalary;
…
{ Ввод исходных данных }
repeat
   write('Введите количество сотрудников (не более',
         MaxN,' ): ');
   readln(N);
until (N>0) AND (N<=MaxN);
For I := 1 to N do
Begin
  Write('Введите фамилию сотрудника с номером ',I,': ');
  ReadLn(Sotr[i].FIO);
  Write('Введите оклад сотрудника с номером ', I, ': ');
  ReadLn(Sotr[i].O);
  Write('Введите кол-во отработанных дней сотрудника с
        номером ', I, ': ');
  ReadLn(Sotr[i].Ko);
End;

{ Занесение данных во внешнюю память }

Assign(F, 'MyFile.fsf');
Rewrite(F);
For I := 1 to N do
  Write(F, Sotr[i]);
Close(F);
…
```

2. Чтение исходных данных из внешней памяти, расчет начисленных сумм и вывод на печать.

```
…
{ Чтение данных из внешней памяти }
Assign(F, 'MyFile.fsf');
Reset(F);
For I := 1 to N do
  Read(F, Sotr[i]);
Close(F);
{ Расчет и печать начисленных сумм }
For I := 1 to N do
Begin
  Sotr[i].S := Sotr[i].O * Sotr[i].Ko / Kr;
  WriteLn(Sotr[i].FIO, ': ', Sotr[i].S);
End;
…
```

Представленные программы решают поставленную задачу при сделанных предположениях. Необходимые для этого данные хранятся в файле MyFile.fsf, предназначенном только для решения этой задачи. Отметим, что в этом случае описание данных включено в прикладную программу. При изменении формата записей файла необходимо изменение прикладной программы. Таким образом, программная система, решающая поставленную задачу, определяет свои собственные данные и управляет ими. Такие программные системы называются файловыми системами [2], [3].

Задача 2. Учет кадрового состава.

Здесь обрабатываются сведения о сотруднике, представленные в карточке СОТРУДНИК:

Фамилия, имя, отчество *FIO*	Должность *D*	Год рождения *G*	Оклад *O*	Место жительства *M*

Решение задачи состоит из следующих этапов:

Ввод исходных данных и занесение их во внешнюю память.

Чтение исходных данных из внешней памяти с целью удаления, корректировки или добавления записи.

```
…
{ Чтение данных из внешней памяти }
Assign(F, 'MyFile.fsf');
Reset(F);
IsFound := False;
For I := 1 to N do
Begin
       Read(F, Sotr);
       If Sotr.FIO = KeyFio Then
       Begin
            IsFound := True;
            Sotr.D := 'Начальник отдела';
```

```
            Seek(F, FilePos(F)-1);
            Write(F, Sotr);
            Break;
        End;
If IsFound Then
WriteLn('Корректировка успешно произведена')
Else WriteLn('Сотрудника ', KeyFio, 'не обнаружено');
Close(F);
…
```

В рассматриваемом случае задача 2 решается независимо от задачи 1.

Задача 3. Учет экономии фонда оплаты труда (ФОТ) в связи с болезнью сотрудников.

Обрабатываются сведения, представленные записями ЭКОНОМИЯ ФОТ:

Фамилия, имя, отчество *FIO*	Оклад *O*	Количество дней на больничном листе *K*дв	Невыплаченная сумма *SN*

$$SN = K_{\text{дв}} O / K_r .$$

Программа решения задачи 3 аналогична программе решения задачи 1.

Рассмотрим типичный случай, когда все три вышеуказанные программные системы функционируют в одной организации. Отметим следующие принципиальные эксплуатационные недостатки:

Информация дублируется. В трех файлах присутствуют поля *FIO*, *O*, что приводит к существенному перерасходу памяти. При внесении изменений (например, изменении фамилии) приходится вносить одно и то же значение несколько раз в разные файлы, что приводит к увеличению затрат машинного времени. Существует потенциальная возможность противоречивости данных (в один файл изменения внесены, в другой – нет).

Устранить перечисленные недостатки можно, объединив соответствующие записи и создав единую информационную базу для всех вышеназванных задач. На первый взгляд наиболее естественно объединить все записи в одну, убрав дублирующие поля. Получаем возможный вариант объединения:

FIO	*D*	*O*	*G*	K_o	*M*	$K_{дв}$	*S*	*SN*

Дублирование информации полностью убрано. Расход памяти минимален. Недостатки устранены. Рассмотрим, как в этом случае изменится время решения задач 1–3. Время решения задачи прямо пропорционально объему считываемых из внешней памяти данных.

Обозначим T_i, l_i, N_i соответственно время решения, длину записи, число записей i-й задачи (i = 1, 2, 3) при использовании отдельных файлов для каждой задачи:

$$T_i \approx C^* \ l_i \ ^* N_i ,$$

где C – некоторый коэффициент пропорциональности.

Обозначим R_i, d, N соответственно время решения i-й задачи (i = 1, 2, 3) при использовании файла объединенных записей, длину записи, число записей:

$$R_i \approx C^* d^* N.$$

Заметим, что $N1 = N2 = N$, $N3 \ll N$.

Тогда время решения i-й задачи ($i = 1, 2$) при использовании объединенного файла увеличится в $R_i / T_i \approx d / l_i$ раз. Для нашего примера время решения задач в зависимости от выбранной длины полей может изменяться в 2–3 раза. Таким образом, платой за исключение дублирования информации является увеличение времени решаемых задач. Заметим, что такое увеличение, как правило, допустимо.

Время решения задачи 3 увеличится в $R3/T3 \approx d^*N / l3 \ ^*N3$ раз. Так как для данного примера $N3 \ll N$, то $R3 \gg T3$. Время решения задачи 3 может увеличиться на несколько порядков, что совершенно недопустимо.

Рассмотрим другой вариант построения единой информационной базы. Объединим записи задач 1 и 2, запись задачи 3 оставим отдельно. Получим два типа записей:

FIO	*D*	*O*	*G*	K_o	*S*	*M*

FIO	*O*	$K_{дв}$	*SN*

В этом случае дублирование остается (дублируются поля *FIO*, *O*). Но так как $N_3 \ll N$, то общий объем дублирования незначителен. Время решения задачи 1 и 2 в этом случае незначительно возрастет по сравнению с вариантом отдельных файловых систем, время решения задачи 3 такое же, как и в начальном варианте отдельного файла. Такое объединение позволяет значительно уменьшить влияние недостатков и в то же время существенно увеличивает время решения всех задач. Все три задачи можно решать, используя общую информационную базу из двух типов записей. Отметим, что два приведенных типа записей связаны друг с другом по полю *FIO* (находятся в некотором отношении). Отметим, что приведенные варианты интеграции не исчерпывают все возможные способы интеграции данных для приведенных задач и к вопросу выбора наилучшего варианта вернемся в последующих лекциях.

Здесь очень важно, что в этом случае для решения вышеуказанных задач используется некоторый новый вид данных, формируемый на основе интеграции записей.

Для описания этого вида данных вводится новое понятие **«База данных»** [1].

База данных – совокупность экземпляров различных типов записей и отношений между записями и элементами.

Базу данных можно определить как совокупность взаимосвязанных хранящихся вместе данных при наличии такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений.

Таким образом, появление понятие **«Базы даннах»** обусловлено возникновением нового класса невычислительных задач, при решении которых используются общие данные. В качестве основного критерия оптимальности функционирования базы данных,как правило, используются временные характеристики реализации запросов пользователей прикладными программами.

Краткие итоги. Рассмотрено развитие основных понятий представления данных. Описаны классические понятия программирования, связанные с данными (переменная, массив) и появление новых понятий программирования (поле, запись, файл) как следствие расширения круга решаемых задач и их отражения в системах программирования. Поставлена задача интегрирования данных при использовании несколькими задачами общих данных. Определено понятие базы данных.

Контрольные тесты

Задача 1. С чем связано появление новых понятий обработки данных?

Вариант 1.
С чем связано появление новых понятий обработки данных?

☐ с развитием вычислительной техники
☐ с развитием операционных систем
☐ с повышением квалификации программистов
⊞ с расширением круга решаемых на ЭВМ задач

Вариант 2.
Какие задачи относятся к задачам обработки данных?

☐ задачи с большим объемом сложных вычислений
⊞ задачи учета кадрового состава организации
⊞ задачи бухгалтерского учета
☐ решение систем линейных уравнений

Вариант 3.
Какие новые понятия в представлении данных появились с появлением задач обработки данных?

☐ простая переменная
☐ массив
⊞ запись
⊞ поле

Задача 2. Что такое логическая запись?

Вариант 1.
Что является элементом логической записи?

☐ простые переменные
☐ элементы массива
☐ файлы
⊞ поля

Вариант 2.
Что не является элементом логической записи?

⊞ простые переменные
⊞ элементы массива
⊞ файлы
☐ поля

Вариант 3.
Из чего состоит логическая запись?

☐ из простых переменных и полей
☐ из элементов массива и переменных
⊞ из полей
☐ из простых переменных

Задача 3. Что такое логический файл?

Вариант 1.
Что такое логический файл?

☐ совокупность полей
☐ совокупность логических записей
⊞ совокупность экземпляров логических записей
☐ набор данных во внешней памяти ЭВМ

Вариант 2.
Из каких составляющих элементов состоит логический файл?

☐ из полей
☐ из элементов массива
⊞ из экземпляров логических записей
☐ из переменных

Вариант 3.
Какие понятия не используются при описании логического файла?

☐ экземпляр записи
☐ поле
☐ логическая запись
⊞ массив

Задача 4. Какие основные этапы решения задачи обработки организационных документов (обработки данных)?

Вариант 1. Из каких основных этапов состоит решение задачи обработки организационных документов (обработки данных)?

☐ проведение сложных математических вычислений
⊞ занесение данных во внешнюю память
⊞ чтение данных из внешней памяти
⊞ поиск неооходимых данных

Вариант 2. Какие из перечисленных действий не входят в решение задач обработки организационных документов (обработки данных)?

⊞ проведение сложных математических вычислений
☐ занесение данных во внешнюю память
☐ чтение данных из внешней памяти
☐ поиск неооходимых данных

Вариант 3. Какие основные операции с данными производятся в задачах обработки организационных документов (обработки данных)?

⊞ поиск необходимых данных
⊞ модификация данных
⊞ удаление данных
⊞ добавление данных

Задача 5. Основные свойства программных систем с отдельными файлами для каждой задачи (файловых систем).

Вариант 1. Какие из перечисленных свойств характерны для комплекса программных систем с отдельными файлами для каждой задачи (файловых систем)?

⊞ дублирование данных
☐ большое время решения каждой задачи
☐ высокая достоверность всей совокупности данных
⊞ потенциальная противоречивость данных

Вариант 2. Какие из перечисленных свойств не характерны для комплекса программных систем с отдельными файлами для каждой задачи (файловых систем)?

☐ дублирование данных
⊞ большое время решения каждой задачи
⊞ высокая достоверность всей совокупности данных
☐ потенциальная противоречивость данных

Вариант 3. Какие из перечисленных свойств комплекса программных систем с отдельными файлами для каждой задачи (файловых систем) можно устранить объединением (интеграцией) данных?

⊞ дублирование данных
☐ большое время решения каждой задачи
☐ высокая достоверность всей совокупности данных
⊞ потенциальная противоречивость данных

Задача 6. Как представляются интегрированные данные?

Вариант 1. В каком виде представляются интегрированные данные?

☐ отдельный файл
☐ набор отдельных файлов
☐ набор экземпляров записей одного типа
⊞ набор экземпляров записей разных типов и связей между ними

Вариант 2. В каком виде не представляются интегрированные данные?

⊞ отдельный файл
☐ набор связанных файлов
⊞ набор экземпляров записей одного типа
☐ набор экземпляров записей разных типов и связей между ними

Вариант 3. Какое понятие из нижеперечисленных является важнейшим при интеграции данных?

☐ файл
☐ запись
☐ экземпляр записи
⊞ связь между записями (файлами)

Задача 7. Определить понятие базы данных.

Вариант 1. Что такое база данных?

☐ совокупность экземпляров записи одного типа
☐ совокупность экземпляров записей разных типов
⊞ совокупность экземпляров записей разных типов и связей (отношений) между ними
☐ поименованная совокупность логических записей

Вариант 2. Какие понятия соответствуют содержанию понятия базы данных?

☐ набор данных для решения отдельной задачи
☐ набор отдельных файлов
⊞ набор связанных файлов
☐ файловая система

Вариант 3. Какие понятия не соответствуют содержанию понятия базы данных?

⊞ набор данных для решения отдельной задачи
⊞ набор отдельных файлов
☐ набор связанных файлов
⊞ файловая система

Задача 8. Отметить основные свойства базы данных.

Вариант 1. Отметить основные свойства базы данных.

☐ отсутствие дублирования
⊞ минимальная избыточность
☐ минимальное время решения всех задач
⊞ используется для решения ряда задач

Вариант 2. Какие из перечисленных свойств характерны для базы данных?

⊞ минимальное дублирование данных
⊞ интеграция данных
☐ каждая задача решается за минимально возможное время
☐ отсутствие дублирования

Вариант 3. Какие из перечисленных свойств не характерны для базы данных?

☐ минимальное дублирование данных
☐ интеграция данных
⊞ каждая задача решается за минимально возможное время
⊞ отсутствие дублирования

Литература

1. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Щерса. – М.: Мир, 1980. – 664 с.
2. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
3. Крёнке Д. Теория и практика построения баз данных. 8-е изд. – СПб.: Питер, 2003. – 800 с.

Лекция 2. Системы управления базами данных

Вводится понятие системы управления базами данных (СУБД).Дается характеристика основных функций системы управления базами данных.

Ключевые термины: Система управления базами данных, СУБД, банк данных, функции СУБД, транзакции, блокировки

Цель лекции: показать необходимость создания программного интерфейса между прикладными программами и базой данных, определить понятие системы управления базами данных и сформулировать основные функции СУБД, вытекающие из задачи взаимодействия многих пользователей с базой данных.

В прикладной программе, использующей при решении задачи один или несколько отдельных файлов, за сохранность и достоверность данных отвечал программист, работающий с этой задачей. Использование базы данных предполагает работу с ней нескольких прикладных программ, решающих задачи разных пользователей.

Естественно, что за сохранность и достоверность интегрированных данных программист, решающий одну из прикладных задач, отвечать уже не может. Кроме того, расширение круга решаемых с использованием базы данных задач может приводить к появлению новых типов записей и отношений между ними. Такое изменение структуры базы данных не должно вести к изменению множества ранее разработанных и успешно функционирующих прикладных программных систем, работающих с базой данных. С другой стороны, возможное изменение любой из прикладных программ, в свою очередь, не должно приводить к изменению структуры данных. Все вышесказанное обусловливает необходимость отделения данных от прикладных программ.

Роль интерфейса между прикладными программами и базой данных, обеспечивающего их независимость, играет программный комплекс – система управления базами данных (**СУБД**) (рис. 2.1).

СУБД – программный комплекс поддержки интегрированной совокупности данных, предназначенный для создания, ведения и использования базы данных многими пользователями (прикладными программами).

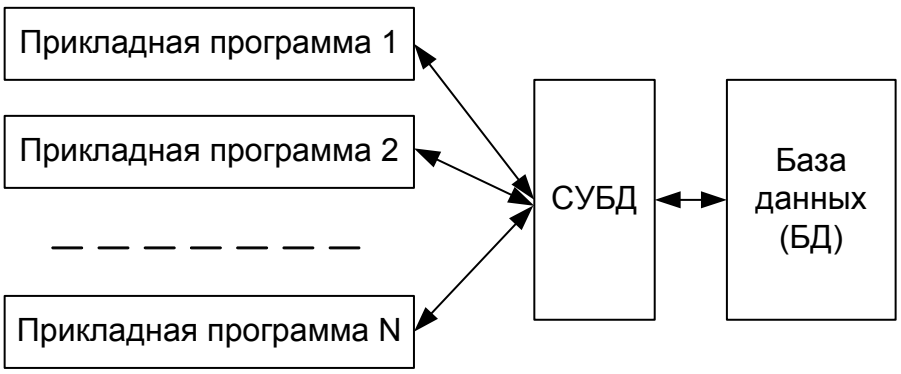


Рис.2.1. Обеспечение независимости прикладных программ и базы данных

Определим еще одно понятие.

Банк данных – система языковых, алгоритмических, программных, технических и организационных средств поддержки интегрированной совокупности данных, а также сами эти данные, представленные в виде баз данных.

Перечислим основные **функции системы управления базами данных**.

1. Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки.

Как правило, создание структуры **базы данных** происходит в режиме диалога. **СУБД** последовательно запрашивает у пользователя необходимые данные. В большинстве современных **СУБД база данных** представляется в виде совокупности таблиц. Рассматриваемая функция позволяет описать и создать в памяти структуру таблицы, провести начальную загрузку данных в таблицы. Примеры таких действий для **СУБД** MS Access приведены на рисунке 2.2.

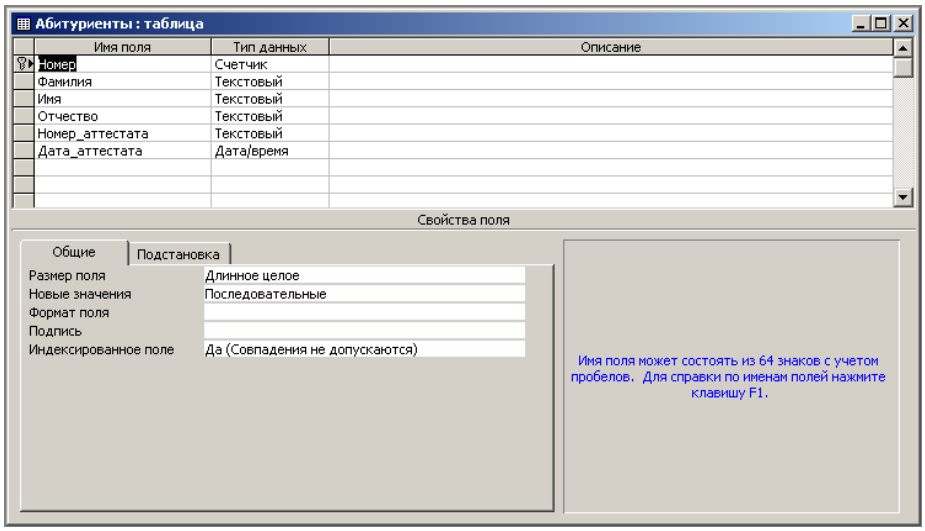


Рис. 2.2. Формирование структуры базы данных в СУБД Access

2. Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация).

Такие возможности в **СУБД** представляются либо на основе использования специального языка программирования, входящего в состав **СУБД,** либо с помощью графического интерфейса.

В MS Access реализация данной функции может быть реализована созданием запросов и форм ввода с помощью графического интерфейса (рис.2.3.).

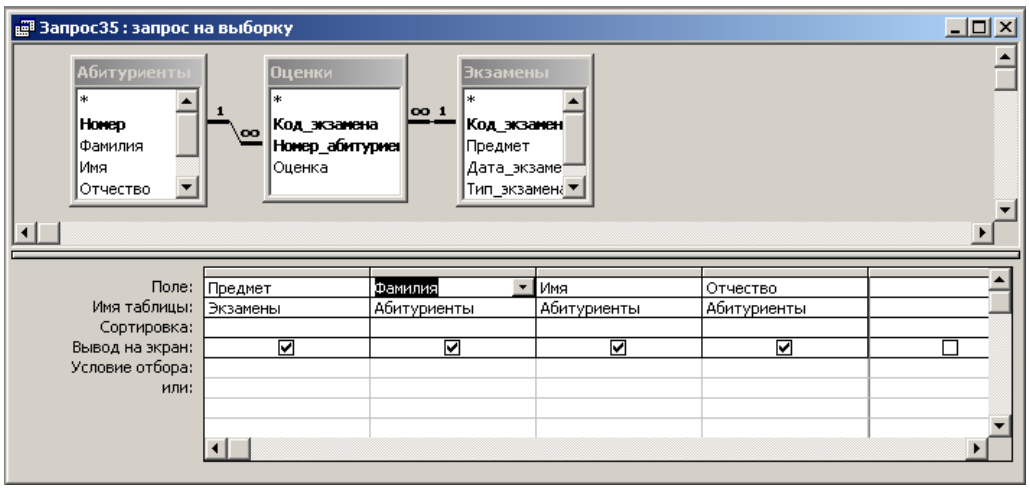


Рис. 2.3. Формирование запроса на выборку в СУБД Access

Для клиент-серверных **СУБД** существуют средства, позволяющие выполнять запросы, и программные средства, позволяющие создавать графический интерфейс пользователя.

3. Обеспечение независимости прикладных программ и данных (логической и физической независимости).

Важнейшим свойством **СУБД** является возможность поддерживать два независимых взгляда на базу данных – «взгляд пользователя», воплощаемый в логическом представлении данных, и его отражения в прикладных программах; и «взгляд системы» – физическое представление данных в памяти ЭВМ. Обеспечение логической независимости данных предоставляет возможность изменения (в определенных пределах) логического представления базы данных без необходимости изменения физических структур хранения данных. Таким образом, изменение логического представления данных в прикладных программах не приводит к изменению структур хранения данных. Обеспечение физической независимости данных предоставляет возможность изменять (в определенных пределах) способы организации базы данных в памяти ЭВМ не вызывая необходимости изменения «логического» представления данных. Таким образом, изменение способов организации базы данных не приводит к изменению прикладных программ.

4. Защита логической целостности **базы данных**.

Основной целью реализации этой функции является повышение достоверности данных в базе данных. Достоверность данных может быть нарушена при их вводе в БД или при неправомерных действиях процедур обработки данных, получающих и заносящих в БД неправильные данные. Для повышения достоверности данных в системе объявляются так называемые ограничения целостности, которые в определенных случаях «отлавливают» неверные данные. Так, во всех современных **СУБД** проверяется соответствие вводимых данных их типу, описанному при создании структуры. Система не позволит ввести символ в поле числового типа, не позволит ввести недопустимую дату и т.п. В развитых системах ограничения целостности описывает программист, исходя из содержательного смысла задачи, и их проверка осуществляется при каждом обновлении данных. Более подробно разные аспекты логической целостности базы данных будут рассматриваться в последующих разделах.

5. Защита физической целостности.

При работе ЭВМ возможны сбои в работе (например, из-за отключения электропитания), повреждение машинных носителей данных. При этом могут быть нарушены связи между данными, что приводит к невозможности дальнейшей работы. Развитые СУБД имеют средства восстановления базы данных. Важнешим используемым понятием является понятие **«транзакции»**. ***Транзакция – это единица действий, производимых с базой данных.*** В состав транзакции может входить несколько операторов изменения базы данных, но либо выполняются все эти операторы, либо не выполняется ни один. СУБД, кроме ведения собственно базы данных, ведет также журнал **транзакций**.

Необходимость использования **транзакций** в базах данных проиллюстрируем на упрощенном примере. Предположим, что база данных используется в некотором банке и один из клиентов желает перевести деньги на счет другого клиента банка. В базе данных хранится информация о количестве денег у каждого из клиентов. Нам нужно сделать два изменения в базе данных – уменьшить сумму денег на счете одного из клиентов и, соответственно, увеличить сумму денег на другом счете. Конечно, реальный перевод денег в банке представляет собой гораздо более сложный процесс, затрагивающий много таблиц, а возможно, и много баз данных. Однако суть остается та же – нужно либо совершить все действия (увеличить счет одного клиента и уменьшить счет другого), либо не выполнить ни одно из этих действий. Нельзя уменьшить сумму денег на одном счете, но не увеличить сумму денег на другом. Предположим также, что после выполнения первого из действий (уменьшения суммы денег на счете первого клиента) произошел сбой. Например, могла прерваться связь клиентского компьютера с базой данных или на клиентском компьютере мог произойти системный сбой, что привело к перезагрузке операционной системы. Что в этом случае стало с базой данных? Команда на уменьшение денег на счете первого клиента была выполнена, а вторая команда – на увеличение денег на другом счете – нет,что привнло бы к противоречивому, неактуальному состоянию базы данных.

Использование механизма **транзакций** позволяет находить решение в этом и подобных случаях. Перед выполнением первого действия выдается команда начала транзакции. В транзакцию включается операция снятия денег на одном счете и увеличения суммы на другом счете. Оператор завершения транзакций обычно называется COMMIT. Поскольку после выполнения первого действия транзакция не была завершена, изменения не будут внесены в базу данных. Изменения вносятся (фиксируются) только после завершения транзакции. До выдачи данного оператора сохранения данных в базе не произойдет.

В нашем примере, поскольку оператор фиксации транзакции не был выдан, база данных «откатится» в первоначальное состояние – иными словами, суммы на счетах клиентов останутся те же, что и были до начала транзакции. Администратор базы данных может отслеживать состояние транзакций и в необходимых случаях вручную «откатывать» транзакции. Кроме того, в очевидных случаях СУБД самостоятельно принимает решение об «откате» транзакции.

Транзакции не обязательно могут быть короткими. Бывают транзакции, которые длятся несколько часов или даже несколько дней. Увеличение количества действий в рамках одной транзакции требует увеличения занимаемых системных ресурсов. Поэтому желательно делать транзакции по возможности короткими. В журнал транзакций заносятся все транзакции – и зафиксированные, и завершившиеся «откатом». Ведение журнала транзакций совместно с созданием резервных копий базы данных позволяет достичь высокой надежности базы данных.

Предположим, что база данных была испорчена в результате аппаратного сбоя компьютера, на котором был установлен сервер СУБД. В этом случае нужно использовать последнюю сделанную резервную копию базы данных и журнал транзакций. Причем применить к базе данных нужно только те транзакции, которые были зафиксированы после создания резервной копии. Большинство современных СУБД позволяют администратору воссоздать базу данных исходя из резервной копии и журнала транзакций. В таких системах в определенный момент БД копируется на резервные носители. Все обращения к БД записываются программно в журнал изменений. Если база данных разрушена, запускается процедура восстановления, в процессе которой в резервную копию из журнала изменений вносятся все произведенные изменения.

6. Управление полномочиями пользователей на доступ к базе данных.

Разные пользователи могут иметь разные полномочия по работе с данными (некоторые данные должны быть недоступны; определенным пользователям не разрешается обновлять данные и т.п.). В СУБД предусматриваются механизмы разграничения полномочий доступа, основанные либо на принципах паролей, либо на описании полномочий.

7. Синхронизация работы нескольких пользователей.

Достаточно часто может иметь место ситуация, когда несколько пользователей одновременно выполняют операцию обновления одних и тех же данных. Такие коллизии могут привести к нарушению логической целостности данных, поэтому система должна предусматривать меры, не допускающие обновление данных другим пользователям, пока работающий с этими данными пользователь полностью не закончит с ними работать. Основным используемым здесь понятием является понятие «**блокировка**». **Блокировки** необходимы для того, чтобы запретить различным пользователям возможность одновременно работать с базой данных, поскольку это может привести к ошибкам.

Для реализации этого запрета СУБД устанавливает блокировку на объекты, которые использует транзакция. Существуют разные типы блокировок – табличные, страничные, строчные и другие, которые отличаются друг от друга количеством заблокированных записей. Чаще других используется строчная блокировка – при обращении транзакции к одной строке блокируется только эта строка, остальные строки остаются доступными для изменения.

Таким образом, процесс внесения изменений в базу данных состоит из следующей последовательности действий: выдается оператор начала транзакции, выдается оператор изменения данных, СУБД анализирует оператор и пытается установить блокировки, необходимые для его выполнения, в случае успешной блокировки оператор выполняется, затем процесс

повторяется для следующего оператора транзакции. После успешного выполнения всех операторов внутри транзакции выполняется оператор фиксации транзакции. СУБД фиксирует изменения, сделанные транзакцией, и снимает блокировки. В случае неуспеха выполнения какого-либо из операторов транзакция «откатывается», данные получают прежние значения, блокировки снимаются.

8. Управление ресурсами среды хранения.

БД располагается во внешней памяти ЭВМ. При работе в БД заносятся новые данные (занимается память) и удаляются данные (освобождается память). СУБД выделяет ресурсы памяти для новых данных, перераспределяет освободившуюся память, организует ведение очереди запросов к внешней памяти и т.п.

9. Поддержка деятельности системного персонала.

При эксплуатации базы данных может возникать необходимость изменения параметров СУБД, выбора новых методов доступа, изменения (в определенных пределах) структуры хранимых данных, а также выполнения ряда других общесистемных действий. СУБД предоставляет возможность выполнения этих и других действий для поддержки деятельности БД обслуживающему БД системному персоналу, называемому администратором БД.

Краткие итоги. Рассмотрено понятие системы управления базами данных как интерфейса между прикладными программами и базами данных. Введено понятие банка данных. Дана характеристика основных функций систем управления базами данных, вытекающие из задачи взаимодействия многих пользователей с базой данных:

- Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки
- Предоставление пользователям возможности манипулирования данными (выборка необходимых данных, выполнение вычислений, разработка интерфейса ввода/вывода, визуализация).
- Обеспечение независимости прикладных программ и (логической и физической независимости).
- Защита логической целостности базы данных.
- Защита физической целостности.
- Управление полномочиями пользователей на доступ к базе данных.
- Синхронизация работы нескольких пользователей.
- Управление ресурсами среды хранения.
- Поддержка деятельности системного персонала.

Информация по содержанию данной лекции содержится в [1-3]

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Основные причины появления систем управления базами данных.

Вариант 1.
Что обусловило появление систем управления базами данных?

☐ необходимость повышения эффективности работы прикладных программ
☐ появление современных операционных систем
⊞ совместное использование данных разными прикладными программами
☐ большой объем данных в прикладной программе

Вариант 2.
Основные требования, побуждающие пользователя к использованию СУБД:

☐ необходимость представления средств организации данных прикладной программе
☐ большой объем данных в прикладной программе
☐ большой объем сложных математических вычислений
⊞ необходимость решения ряда задач с использованием общих данных

Вариант 3.
Требования, из которых не следует необходимость в использовании СУБД:

⊞ необходимость представления средств организации данных прикладной программе
⊞ большой объем данных в прикладной программе
☐ большой объем сложных математических вычислений
⊞ необходимость решения ряда задач с использованием общих данных

Задача 2. Основная роль СУБД

Вариант 1
Основное назначение СУБД:

☐ обеспечение независимости прикладных программ и данных
☐ представление средств организации данных одной прикладной программе
☐ поддержка сложных математических вычислений
⊞ поддержка интегрированной совокупности данных

Вариант 2.
Что не входит в назначение СУБД?

☐ обеспечение независимости прикладных программ и данных
⊞ представление средств организации данных одной прикладной программе
⊞ поддержка сложных математических вычислений
☐ поддержка интегрированной совокупности данных

Вариант 3.
Для чего предназначена СУБД?

⊞ для создания базы данных
⊞ для ведения базы данных
⊞ для использования базы данных
☐ для разработки прикладных программ

Задача 3. Основные функции СУБД

Вариант 1.
Что входит в функции СУБД?

⊞ создание структуры базы данных
⊞ загрузка данных в базу данных
⊞ предоставление возможности манипулирования данными
☐ проверка корректности прикладных программ, работающих с базой данных
⊞ обеспечение логической и физической независимости данных
⊞ защита логической и физической целостности базы данных
⊞ управление полномочиями пользователей на доступ к базе данных

Вариант 2.
Что не входит в функции СУБД?

☐ создание структуры базы данных
☐ загрузка данных в базу данных
☐ предоставление возможности манипулирования данными
⊞ проверка корректности прикладных программ, работающих с базой данных
☐ обеспечение логической и физической независимости данных
☐ защита логической и физической целостности базы данных
☐ управление полномочиями пользователей на доступ к базе данных

Вариант 3.
Основные средства СУБД для работы пользователя с базой данных:

⊞ язык запросов
⊞ графический интерфейс
☐ алгоритмический языкПаскаль
☐ разрабатываемые пользователем программы

Задача 4. Определение понятия банка данных.

Вариант 1.

Что входит в понятие банка данных?

⊞ база данных
☐ прикладные программы работы с базой данных
⊞ СУБД
⊞ компьютеры с базой данных
⊞ администраторы базы данных

Вариант 2.

Как соотносятся понятия база данных и банк данных?

☐ одно и то же
☐ база данных включает банк данных
⊞ банк данных включает базу данных
☐ не связанные понятия

Вариант 3.

Что не входит в понятие банк данных?

☐ база данных
☐ технология обработки данных
☐ алгоритмы обработки данных
⊞ помещение, где обрабатываются данные
☐ администраторы базы данных

Задача 5. Логическая и физическая независимость данных.

Вариант 1.
Что дает логическая и физическая независимость данных?

⊞ изменение прикладных программ не приводит к изменению физического представления базы данных
☐ изменение программ СУБД не приводит к изменению физического представления данных
⊞ изменение физического представления данных не приводят к изменению прикладных программ
⊞ изменение программ СУБД не приводит к изменению прикладных программ

Вариант 2.
К чему приведет отсутствие логической и физической независимости данных?

⊞ к необходимости изменения прикладных программ при изменении физического представления базы данных
☐ к большей достоверности данных
⊞ к возможному изменению физического представления данных при изменении прикладных программ
☐ к более эффективному взаимодействию пользователей с базой данных

Вариант 3.
В чем состоит логическая и физическая независимость данных в базах данных?

⊞ представление о данных в прикладных программах и физическое представление данных в компьютере независимы.
☐ данные одной прикладной программы независимы от данных другой прикладной программы
⊞ изменение прикладных программ не приводит к изменению физического представления базы данных
⊞ изменение прикладных программ не приводит к изменению программ СУБД

Задача 6. Что такое логическая и физическая целостность базы данных?

Вариант 1.
Основные цели обеспечения логической и физической целостности базы данных?

☐ защита от неправильных действий прикладного программиста
☐ защита от неправильных действий администратора баз данных
⊞ защита от возможных ошибок ввода данных
⊞ защита от машинных сбоев
⊞ защита от возможного появления несоответствия между данными после выполнения операций удаления и корректировки

Вариант 2.
Какие средства используются в СУБД для обеспечения логической целостности?

⊞ Контроль типа вводимых данных
⊞ Описание ограничений целостности и их проверка
☐ Блокировки
☐ Синхронизация работы пользователей

Вариант 3. Какие средства используются в СУБД для обеспечения физической целостности?

☐ контроль типа вводимых данных
☐ описание ограничений целостности и их проверка
⊞ блокировки
⊞ транзакции
⊞ журнал транзакций

Задача 7. Что такое транзакция?

Вариант 1.
В чем суть использования механизма транзакций?

☐ изменения в базу данных вносятся каждой операцией
⊞ изменения в базу данных вносятся только после выполнения определенной последовательности операций
☐ изменения в базу данных вносятся только администратором базы данных
☐ изменения в базу данных вносятся только при определенных условиях

Вариант 2.
При каких условиях система меняет данные в базе данных?

⊞ по завершению транзакции
⊞ по оператору commit
☐ по указанию администратора
☐ по оператору модификации данных

Вариант 3.
Для чего ведется журнал транзакций?

☐ для анализа действий с базой данных
☐ для использования прикладными программами
☐ для проверки правильности данных
⊞ для восстановления базы данных

Задача 8. Что такое синхронизация работы пользователей?

Вариант 1.

Зачем нужна синхронизация?

☐ для ускорения работы прикладных программ
☐ для восстановления базы данных после сбоев
⊞ для предотвращения нарушения достоверности данных
☐ для поддержки деятельности системного персонала

Вариант 2.

Какие средства используются для синхронизации?

⊞ блокировки
☐ транзакции
☐ пароли
☐ описание полномочий

Вариант 3.

Последовательность действий СУБД при синхронизации:

☐ установка блокировки, начало транзакции, снятие блокировки, завершение транзакции
⊞ начало транзакции, установка блокировки, завершение транзакции, снятие блокировки
☐ начало транзакции, установка блокировки, продолжение транзакции, снятие блокировки, завершение транзакции
⊞ начало транзакции, установка блокировки, выполнение транзакции, откат транзакции, снятие блокировки

Литература

1. Дейт К.Дж. Введение в системы баз данных: Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с.
2. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
3. Крёнке Д. Теория и практика построения баз данных. 8-е изд. – СПб.: Питер, 2003. – 800 с.

Лекция 3. Различные архитектурные решения, используемые при реализации многопользовательских СУБД. Краткий обзор СУБД.

В лекции рассматриваются различные варианты технологии работы с базой данных в многопользовательском режиме (централизованная архитектура, компьютерная сеть с файловым сервером, клиент-серверная архитектура). Дается краткий обзор современных СУБД.

Ключевые термины: многопользовательские СУБД, файл-серевер, сеть с файловым сервером, клиент-сервер,трехзвенная архитектура, многозвенная архитектура, современные СУБД.

Цель лекции: показать основные варианты технологии работы нескольких пользователей с одной базой данных, связанные как с основными свойствами вычислительной техники так и с развитием программного обеспечения.

Как уже отмечалось, понятие базы данных изначально предполагало возможность решения многих задач несколькими пользователями. В связи с этим, важнешей характеристикой современных СУБД является наличие многопользовательской технологии работы. Разная реализация таких технологий в разное время была связана как с основными свойствами вычислительной техники так и с развитием программного обеспечения. Дадим краткую характеристику этих технологий в хронологическом порядке.

3.1. Централизованная архитектура

При использовании этой технологии база данных, СУБД и прикладная программа (приложение) располагаются на одном компьютере (мэйнфрейме или персональном компьютере) (рис.3.1.). Для такого способа организации не требуется поддержки сети и все сводится к автономной работе. Работа построена следующим образом:

- База данных в виде набора файлов находится на жестком диске компьютера.
- На том же компьютере установлены СУБД и приложение для работы с БД .
- Пользователь запускает приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку/обновление информации.
- Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД.
- СУБД инициирует обращения к данным, обеспечивая выполнение запросов пользователя (осуществляя необходимые операции над данными).
- Результат СУБД возвращает в приложение.
- Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Рис. 3.1. Централизованная архитектура

Подобная архитектура использовалась в первых версиях СУБД DB2, Oracle, Ingres [1].

Многопользовательская технология работы обеспечивалась либо режимом мультипрограммирования (одновременно могли работать процессор и внешние устройства – например, пока в прикладной программе одного пользователя шло считывание данных из внешней памяти, программа другого пользователя обрабатывалась процессором), либо режимом разделения времени (пользователям по очереди выделялись кванты времени на выполнении их программ). Такая технология была распространена в период «господства» больших ЭВМ (IBM-370, ЕС-1045, ЕС-1060). Основным недостатком этой модели является резкое снижение производительности при увеличении числа пользователей.

3.2. Технология с сетью и файловым сервером (архитектура «файл-сервер»)

Увеличение сложности задач, появление персональных компьютеров и локальных вычислительных сетей явились предпосылками появления новой архитектуры **«файл-сервер»**. Эта архитектура баз данных с сетевым доступом предполагает назначение одного из компьютеров сети в качестве выделенного сервера, на котором будут храниться файлы базы данных [2]. В соответствии с запросами пользователей файлы с файл-сервера передаются на рабочие станции пользователей, где и осуществляется основная часть обработки данных. Центральный сервер выполняет в основном только роль хранилища файлов, не участвуя в обработке самих данных (рис.3.2.).

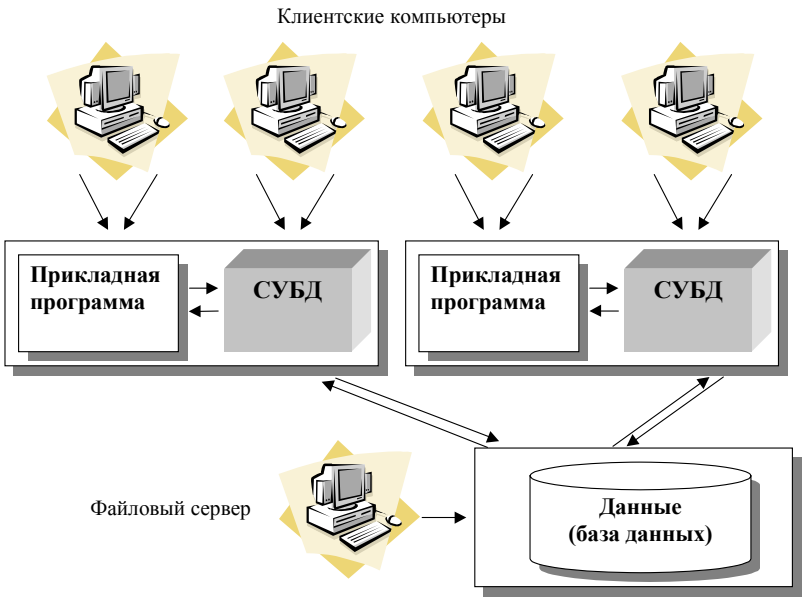


Рис. 3.2.. Архитектура «файл-сервер»

Работа построена следующим образом:

- База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (файлового сервера).
- Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлены СУБД и приложение для работы с БД.
- На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку/обновление информации.
- Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на файловом сервере.
- СУБД инициирует обращения к данным, находящимся на файловом сервере, в результате которых часть файлов БД копируется на клиентский компьютер и обрабатывается, что обеспечивает выполнение запросов пользователя (осуществляются необходимые операции над данными).
- При необходимости (в случае изменения данных) данные отправляются назад на файловый сервер с целью обновления БД.
- Результат СУБД возвращает в приложение.
- Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

В рамках архитектуры «**файл-сервер**» были выполнены первые версии популярных так называемых настольных СУБД, таких, как dBase и Microsoft Access.

В литературе [2] указываются следующие основные недостатки данной архитектуры:

- При одновременном обращении множества пользователей к одним и тем же данным производительность работы резко падает, т.к. необходимо дождаться пока пользователь, работающий с данными, завершит свою работу. В противном случае возможно затирание исправлений, сделанных одними пользователями, изменениями других пользователей.
- Вся тяжесть вычислительной нагрузки при доступе к БД ложится на приложение клиента, так как при выдаче запроса на выборку информации из таблицы вся таблица БД копируется на клиентскую машину и выборка осуществляется на клиенте. Таким образом, неоптимально расходуются ресурсы клиентского компьютера и сети. В результате возрастает сетевой трафик и увеличиваются требования к аппаратным мощностям пользовательского компьютера.
- Как правило, используется навигационный подход, ориентированный на работу с отдельными записями.
- В БД на файл-сервере гораздо проще вносить изменения в отдельные таблицы, минуя приложения, непосредственно из инструментальных средств (например, из

утилиты Database Desktop фирмы Borland для файлов Paradox и dBase); подобная возможность облегчается тем обстоятельством, что фактически у таких СУБД база данных – понятие более логическое, чем физическое, поскольку под БД понимается набор отдельных таблиц, сосуществующих в отдельном каталоге на диске. Все это позволяет говорить о низком уровне безопасности – как с точки зрения хищения и нанесения вреда, так и с точки зрения внесения ошибочных изменений.

- Недостаточно развитый аппарат транзакций служит потенциальным источником ошибок в плане нарушения смысловой и ссылочной целостности информации при одновременном внесении изменений в одну и ту же запись.

3.3. Технология «клиент – сервер»

Использование технологии «**клиент – сервер**» предполагает наличие некоторого количества компьютеров, объединенных в сеть, один из которых выполняет особые управляющие функции (является сервером сети).

Так, архитектура «**клиент – сервер**» разделяет функции приложения пользователя (называемого клиентом) и сервера. Приложение-клиент формирует запрос к серверу, на котором расположена БД, на структурном языке запросов SQL (Structured Query Languague), являющемся промышленным стандартом в мире реляционных БД. Удаленный сервер принимает запрос и переадресует его SQL-серверу БД. SQL-сервер – специальная программа, управляющая удаленной базой данных. SQL-сервер обеспечивает интерпретацию запроса, его выполнение в базе данных, формирование результата выполнения запроса и выдачу его приложению-клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Так как клиентскому приложению посылается результат выполнения запроса, по сети «путешествуют» только те данные, которые необходимы клиенту. В итоге снижается нагрузка на сеть. Поскольку выполнение запроса происходит там же, где хранятся данные (на сервере), нет необходимости в пересылке больших пакетов данных. Кроме того, SQL-сервер, если это возможно, оптимизирует полученный запрос таким образом, чтобы он был выполнен в минимальное время с наименьшими накладными расходами [2, 3]. Архитектура системы представлена на рис. 3.3.

Все это повышает быстродействие системы и снижает время ожидания результата запроса. При выполнении запросов сервером существенно повышается степень безопасности данных, поскольку правила целостности данных определяются в базе данных на сервере и являются едиными для всех приложений, использующих эту БД. Таким образом, исключается возможность определения противоречивых правил поддержания целостности. Мощный аппарат транзакций, поддерживаемый SQL-серверами, позволяет исключить одновременное изменение одних и тех же данных различными пользователями и предоставляет возможность откатов к первоначальным значениям при внесении в БД изменений, закончившихся аварийно [2, 3].

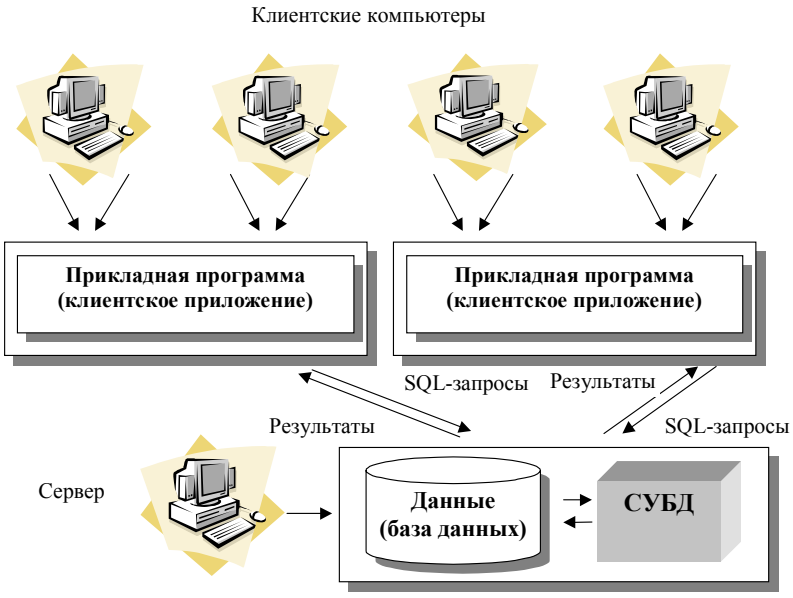


Рис. 3.3. Архитектура «клиент – сервер»

Итак, в результате работа построена следующим образом:

- База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети).
- СУБД располагается также на сервере сети.
- Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлено клиентское приложение для работы с БД.
- На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к СУБД, расположенной на сервере, на выборку/обновление информации. Для общения используется специальный язык запросов SQL, т.е. по сети от клиента к серверу передается лишь текст запроса.
- СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере.
- СУБД инициирует обращения к данным, находящимся на сервере, в результате которых на сервере осуществляется вся обработка данных и лишь результат выполнения запроса копируется на клиентский компьютер. Таким образом СУБД возвращает результат в приложение.
- Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Рассмотрим, как выглядит разграничение функций между сервером и клиентом.

- Функции приложения-клиента:
  - Посылка запросов серверу.
  - Интерпретация результатов запросов, полученных от сервера.
  - Представление результатов пользователю в некоторой форме (интерфейс пользователя).

- Функции серверной части:
  - Прием запросов от приложений-клиентов.
  - Интерпретация запросов.
  - Оптимизация и выполнение запросов к БД.
  - Отправка результатов приложению-клиенту.
  - Обеспечение системы безопасности и разграничение доступа.
  - Управление целостностью БД.
  - Реализация стабильности многопользовательского режима работы.

В архитектуре «клиент – сервер» работают так называемые «промышленные» СУБД. Промышленными они называются из-за того, что именно СУБД этого класса могут обеспечить работу информационных систем масштаба среднего и крупного предприятия, организации, банка. К разряду промышленных СУБД принадлежат MS SQL Server, Oracle, Gupta, Informix, Sybase, , DB2, InterBase и ряд других [2].

Как правило, SQL-сервер обслуживается отдельным сотрудником или группой сотрудников (администраторы SQL-сервера). Они управляют физическими характеристиками баз данных, производят оптимизацию, настройку и переопределение различных компонентов БД, создают новые БД, изменяют существующие и т.д., а также выдают привилегии (разрешения на доступ определенного уровня к конкретным БД, SQL-серверу) различным пользователям [2].

Рассмотрим основные достоинства данной архитектуры по сравнению с архитектурой «файл-сервер»:

- Существенно уменьшается сетевой трафик.
- Уменьшается сложность клиентских приложений (большая часть нагрузки ложится на серверную часть), а следовательно, снижаются требования к аппаратным мощностям клиентских компьютеров.
- Наличие специального программного средства – SQL-сервера – приводит к тому, что существенная часть проектных и программистских задач становится уже решенной.
- Существенно повышается целостность и безопасность БД.

К числу недостатков можно отнести более высокие финансовые затраты на аппаратное и программное обеспечение, а также то, что большое количество клиентских компьютеров, расположенных в разных местах, вызывает определенные трудности со своевременным обновлением клиентских приложений на всех компьютерах-клиентах. Тем не менее, архитектура «клиент – сервер» хорошо зарекомендовала себя на практике, в настоящий момент существует и функционирует большое количество БД, построенных в соответствии с данной архитектурой.

3.4. Трехзвенная (многозвенная) архитектура «клиент – сервер».

Трехзвенная (в некоторых случаях **многозвенная**) **архитектура** (N-tier или multi-tier). представляет собой дальнейшее совершенствование технологии «клиент – сервер». Рассмотрев архитектуру «клиент – сервер», можно заключить, что она является 2-звенной: первое

звено – клиентское приложение, второе звено – сервер БД + сама БД. В **трехзвенной архитектуре** вся бизнес-логика (деловая логика), ранее входившая в клиентские приложения, выделяется в отдельное звено, называемое сервером приложений. При этом клиентским приложениям остается лишь пользовательский интерфейс. Так, в качестве клиентского приложения в описанном выше примере выступает Web-браузер.

Что улучшается при использовании трехзвенной архитектуры? Теперь при изменении бизнес-логики более нет необходимости изменять клиентские приложения и обновлять их у всех пользователей. Кроме того, максимально снижаются требования к аппаратуре пользователей.

Итак, в результате работа построена следующим образом:

- База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети).
- СУБД располагается также на сервере сети.
- Существует специально выделенный сервер приложений, на котором располагается программное обеспечение (ПО) делового анализа (бизнес-логика) [1].
- Существует множество клиентских компьютеров, на каждом из которых установлен так называемый «тонкий клиент» – клиентское приложение, реализующее интерфейс пользователя.
- На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение – тонкий клиент. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к ПО делового анализа, расположенному на сервере приложений.
- Сервер приложений анализирует требования пользователя и формирует запросы к БД. Для общения используется специальный язык запросов SQL, т.е. по сети от сервера приложений к серверу БД передается лишь текст запроса.
- СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере.
- СУБД инициирует обращения к данным, находящимся на сервере, в результате которых результат выполнения запроса копируется на сервер приложений.
- Сервер приложений возвращает результат в клиентское приложение (пользователю).
- Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

3.5. Краткий обзор СУБД

Многие авторы классифицируют СУБД на две большие категории: так называемые «настольные» и «серверные».

3.5.1. Настольные СУБД

Настольные СУБД используются для сравнительно небольших задач (небольшой объем обрабатываемых данных, малое количество пользователей). С учетом этого, указанные СУБД имеют относительно упрощенную архитектуру, в частности, функционируют в режиме файл-сервер, поддерживают не все возможные функции СУБД (например, не ведется журнал транзакций, отсутствует возможность автоматического восстановления базы данных после сбоев и т. п.). Тем не менее, такие системы имеют достаточно обширную область применения. Прежде всего, это государственные (муниципальные) учреждения, сфера образования, сфера обслуживания, малый и средний бизнес. Специфика возникающих там задач заключается в том, что объемы данных не являются катастрофически большими, частота обновлений не бывает слишком высокой, организация территориально обычно расположена в одном небольшом здании, количество пользователей колеблется от одного до 10–15 человек. В подобных условиях использование настольных СУБД для управления информационными системами является вполне оправданным, и они с успехом применяются.

Одними из первых СУБД были так называемые dBase-совместимые программные системы, разработанные разными фирмами. Первой широко распространенной системой такого рода была система dBase III – PLUS (фирма Achton-Tate). Развитый язык программирования, удобный интерфейс, доступный для массового пользователя, способствовали широкому распространению системы. В то же время работа системы в режиме интерпретации обусловливала низкую производительность на стадии выполнения. Это привело к появлению новых систем-компиляторов, близких к системе dBase III – PLUS: Clipper (фирма Nantucket Inc.), FoxPro (фирма Fox Software), FoxBase+ (фирма Fox Software), Visual FoxPro (фирма Microsoft). Одно время достаточно широко использовалась СУБД PARADOX (фирма Borland International).

В последние годы очень широкое распространение получила система управления базами данных Microsoft Access, которая входит в целый ряд версий пакета Microsoft Office(фирма Microsoft).

3.5.2. Серверные СУБД

Для крупных организаций ситуация принципиально меняется. Там использование файл-серверных технологий является неудовлетворительным по описанным выше причинам. Поэтому на передний край борьбы за автоматизацию выходят так называемые серверные СУБД.

Основными производителями таких систем обработки и хранения данных являются 3 корпорации: Oracle, Microsoft и IBM. Диаграмма соотношения объемов продаж соответствующих систем (источник: IDC Report, Май 2006) приводится на рис. 3.4.

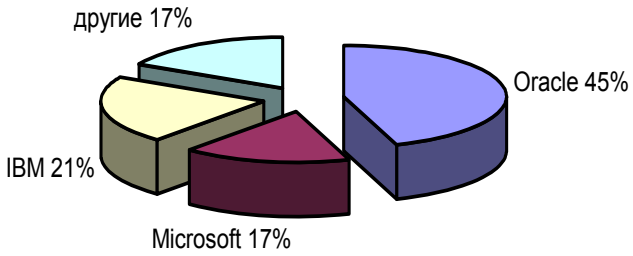


Рис. 3.4. Продажи ПО систем хранения данных в мире

Наиболее распространенными клиент-серверными системами здесь соответственно являются системы Oracle (разработчик компания Oracle), MS SQL Server (разработчик компания Microsoft), DB2 , Informix Dynamic Server (компания IBM).

Дадим краткую характеристику этим системам.

MS SQL Server

К настоящему времени разработано несколько версий систем: MS SQL Server-2000, MS SQL Server -2005, MS SQL Server-2008. Приведем информацию о системе MS SQL Server-2008 с сервера Microsoft (http://www.microsoft.com/rus/SQL/2008/default.mspx)

Microsoft® SQL Server™ 2008 - это законченное предложение в области баз данных и анализа данных для быстрого создания масштабируемых решений электронной коммерции, бизнес-приложений и хранилищ данных. Оно позволяет значительно сократить время выхода этих решений на рынок, одновременно обеспечивая масштабируемость, отвечающую самым высоким требованиям. В SQL Server включена поддержка языка XML и протокола HTTP, средства повышения быстродействия и доступности, позволяющие распределить нагрузку и обеспечить бесперебойную работу, функции для улучшения управления и настройки, снижающие совокупную стоимость владения

Платформа бизнес-анализа SQL Server 2008, тесно интегрированная с Microsoft Office, предоставляет развитую маштабируемую инфраструктуру для внедрения мощных возможностей бизнес-анализа в рабочий процесс всех бизнес-подразделений вашей компании, открывая доступ к нужной бизнес-информации через знакомый интерфейс MS Excel и MS Word.

MS SQL Server-2008 поддерживает создание и работу с корпоративным хранилищем данных, объединяющим информацию со всех систем и приложений, позволяющим получить единую комплексную картину бизнеса вашей компании.

MS SQL Server-2008 предоставляет масштабируемый и высокопроизводительный «процессор данных» - для самых ответственных и требовательных бизнес-приложений, тем, кому необходим высочайший уровень надежности и защиты, позволяя при этом снизить совокупную стоимость владения за счет расширенных возможностей по управлению серверной инфраструктурой.

MS SQL Server-2008 предлагает разработчикам развитую, удобную и функциональную среду программирования, включая средства работы с веб службами, инновационные техно-

логии доступа к данным – все, что необходимо для эффективной работы с данными любых типов и форматов.

Отдельные аспекты MS SQL Server – 2008 будут описаны в лекциях 10 и 14.

Oracle

К настоящему времени разработано несколько версий систем, каждая из которых включает целую линейку продуктов, например Oracle 8, Oracle 9i, Oracle 10g.

Соответствующие линейки продуктов включают как собственно СУБД (например Oracle Database 10g, Oracle Database 11g) , так и средства разработки и анализа данных.

Приведем информацию о системе с сервера Oracle http://www.oracle.com/global/ru/mid/oracle_products/database.html).

Oracle предлагает комплексные, открытые, доступные и удобные в использовании технологические решения. Готовые пакетируемые решения автоматически включают в свою стоимость базу данных, сервер приложений, интеграционную платформу, инструменты аналитики и управления неструктурированными данными. Масштабируемые бизнес-приложения Oracle могут быть легко интегрированы с ИТ-инфраструктурой предприятия без потери уже вложенных в IT инвестиций.

СУБД Oracle Database 11g обеспечивает улучшенные характеристики за счет автоматизации задач администрирования и обеспечения лучших в отрасли возможностей по безопасности и соответствию нормативно-правовым актам в области защиты информации. Появилось больше функций автоматизации, самодиагностики и управления. Среди характеристик системы можно отметить управление большими объемами данных с использованием распределенных таблиц и компрессии, эффективную защиту данных, возможность полного восстановления, возможность интеграции геофизических данных медиа-контента в бизнес-процесс и т.д.

Серверы баз данных компании IBM

К настоящему времени разработаны линейки продуктов DB2 и Informix, включающая как собственно СУБД так и средства разработки и анализа данных (DB2 Universal Database DB2 Personal Edition, DB2 Enterprise 9 и др., а также Informix Dynamic Server, Informix Dynamic Server Express, Informix Extended Parallel Server и др.

Приведем информацию о части таких систем с сервера (http://www-01.ibm.com/software/ru/data/?pgel=ibmhzn)

Универсальный сервер баз данных DB2 Universal Database - это масштабируемая, объектно-реляционная система управления базами данных с интегрированной поддержкой мультимедиа и Web, работающая на системах от персональных компьютеров и серверов на процессорах Intel до Unix, от однопроцессорных систем до симметричных многопроцессорных систем (SMP) и систем с массовым параллелизмом (MPP), на хостах AS/400 и мейнфреймах. DB2 Universal Database объединяет в себе высокую производительность систем обработки транзакций в режиме on-line, объектно-реляционные расширения, усовершенство-

ванные средства оптимизации с возможностями параллельной обработки и поддержкой очень больших баз данных. DB2 Universal Database также имеет новые встроенные средства для облегчения переноса на свою базу приложений, разработанных на других системах управления базами данных, таких как Oracle, Microsoft, Sybase и Informix. Помимо этого, DB2 Universal Database включает в себя дополнительные средства поддержки систем аналитической обработки в реальном времени (OLAP) и систем поддержки принятия решений, множество простых в использовании расширений (DB2 extenders). DB2 Universal Database доступна на абсолютном большинстве ключевых платформ, что дает заказчикам ту гибкость, которая им необходима.

Кроме вышеуказанных зарубежных систем отметим и отечественную разработку – СУБД НИКА, преемницу широко распространенной в Советском Союзе СУБД ИНЕС для ЕС ЭВМ.

В этой лекции был приведен краткий обзор незначительной части существующих СУБД. При огромном разнообразии СУБД вполне естественны споры (которые возникли с момента появления СУБД и, по-видимому, не утихнут никогда) о том, какая СУБД лучше. Нам представляется, что однозначного ответа на этот вопрос не существует. Каждая СУБД имеет свои границы применимости, у каждой из них существуют свои достоинства и недостатки, свое соотношение цена-качество. На форумах Интернета идут постоянные дискуссии пользователей и администраторов баз данных о том, какая СУБД лучше, которые, естественно, не дают окончательного ответа. Любой читатель может иметь личные пристрастия, которые в его глазах уменьшают недостатки и увеличивают достоинства некоторой конкретной СУБД. Общие рекомендации о том, какой из СУБД воспользоваться в каком-то конкретном случае, придумать сложно. Выбор, прежде всего, зависит от класса задач, который нужно решать с использованием СУБД, от тех критериев, которые для соответствующего класса задач являются предпочтительными (стоимость СУБД, простота разработки информационной системы, быстродействие системы при конкретных объемах данных, устойчивость в работе, возможность восстановления, защита от несанкционированного доступа и т.д.). Рекомендация может быть только одна: внимательно изучайте обзоры, читайте пресс-релизы, знакомьтесь с отзывами пользователей, сопоставляйте их наблюдения о плюсах и минусах систем с вашими потребностями и возможностями.

Краткие итоги. В лекции рассмотрены различные архитектурные решения, используемые при реализации многопользовательских СУБД. Централизованная архитектура. Технология с сетью и файловым сервером (архитектура «файл-сервер»). Архитектура «клиент – сервер» (распределенная модель вычислений). Трехзвенная (многозвенная) архитектура) клиент – сервер. Дан обзор современных СУБД (настольные СУБД, серверные СУБД).

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Какие технологии работы с базой данных поддерживают многопользовательский режим?

Вариант 1.
Какие технологии работы с базой данных поддерживают многопользовательский режим?

⊞ технология с централизованной архитектурой базы данных
⊞ технология с сетью и файловым сервером
⊞ технология клиент-сервер
⊞ технология с трехзвенной архитектурой

Вариант 2.
С чем связано развитие многопользовательских технологий работы с базами данных?

⊞ с развитием СУБД
⊞ с развитием вычислительных сетей
☐ с развитием технологий программирования
☐ с ростом квалификации программистов

Вариант 3.
Основные достоинства многопользовательского режима работы с базой данных

☐ возможность использования прикладных программ других пользователей
☐ сокращение затрат машинного времени
⊞ возможность работы многих пользователей с базой данных
☐ сокращение количества обращений к базе данных

Задача 2. Что такое архитектура файл-сервер?

Вариант 1.
Где расположена база данных в такой архитектуре?

☐ на компьютере пользователя
⊞ на специально выделенном компьютере – сервере
☐ на компьютере пользователя и на специально выделенном компьютере – сервере
☐ на всех компьютерах пользователей в локальной сети

Вариант 2.
Где расположены программы пользователя и программы СУБД?

⊞ на компьютере пользователя
☐ на специально выделенном компьютере – сервере
☐ программа пользователя на компьютере пользователя, СУБД на специально выделенном компьютере – сервере
⊞ СУБД расположена на всех компьютерах пользователей в локальной сети

Вариант 3.
На каком компьютере происходит работа с базой данных?

⊞ на компьютере одного пользователя
☐ на специально выделенном компьютере – сервере
☐ прикладные программы работают на компьютере пользователя, программы работают на специально выделенном компьютере – сервере
⊞ прикладные программы и программы СУБД работают на компьютере пользователя.

Задача 3. Как идет обмен информацией между компьютерами в технологии файл-сервер?

Вариант 1.
Что делает компьютер пользователя?

⊞ выполняет прикладную программу
⊞ выполняет программы СУБД
⊞ реализует запросы пользователя к базе данных
⊞ выполняет прикладную программу и программы СУБД

Вариант 2.
Что делает файл-сервер?

☐ формирует ответы на запросы к базе данных
⊞ используется как внешняя память для хранения базы данных
☐ выполняет программы СУБД
☐ выполняет прикладные программы и программы СУБД

Вариант 3.
Как идет обмен информацией между компьютерами в технологии файл-сервер?

☐ в компьютер пользователя считываются все файлы базы данных
☐ в компьютер пользователя считываются только данные, удовлетворяющие запросу пользователя
⊞ в компьютер пользователя считываются только те файлы базы данных, которые необходимы для выполнения запросов
⊞ в компьютер пользователя считываются файлы базы данных, указанные в прикладной программе.

Задача 4. Что такое архитектура клиент-сервер

Вариант 1.
Где расположена база данных в такой архитектуре?

☐ на компьютере пользователя
⊞ на специально выделенном компьютере – сервере
☐ на компьютере пользователя и на специально выделенном компьютере – сервере
☐ на всех компьютерах пользователей в локальной сети

Вариант 2.
Где расположены программы пользователя и программы СУБД в архитектуре клиент-сервер?

☐ на компьютере пользователя
☐ на специально выделенном компьютере – сервере
⊞ программа пользователя на компьютере пользователя, СУБД на специально выделенном компьютере – сервере
☐ СУБД расположена на всех компьютерах пользователей в локальной сети

Вариант 3.
На каком компьютере происходит работа с базой данных в архитектуре клиент-сервер?

☐ на компьютере одного пользователя
⊞ на специально выделенном компьютере – сервере
⊞ прикладные программы работают на компьютере пользователя, программы СУБД работают на специально выделенном компьютере – сервере
⊞ прикладные программы и программы СУБД работают на компьютере пользователя.

Задача 5. Как идет обмен информацией между компьютерами в технологии клиент-сервер?

Вариант 1.
Что делает компьютер – клиент?

☐ выполняет прикладную программу
☐ выполняет программы СУБД
☐ реализует запросы пользователя к базе данных
⊞ выполняет прикладную программу и программы СУБД

Вариант 2.
Что делает сервер?

⊞ формирует ответы на запросы к базе данных
☐ используется как внешняя память для хранения базы данных
⊞ выполняет программы СУБД
☐ выполняет прикладные программы и программы СУБД

Вариант 3.
Как осуществляется обмен информацией между компьютером-клиентом и сервером?

☐ в компьютер-клиент считываются все файлы базы данных
⊞ в компьютер-клиент считываются только данные, удовлетворяющие запросу пользователя
☐ в компьютер-клиент считываются только те файлы базы данных, которые необходимы для выполнения запросов
☐ в компьютер-клиент считываются файлы базы данных, указанные в прикладной программе

Задача 6. Что такое трехзвенная (многозвенная) архитектура

Вариант 1. Что отличает трехзвенную архитектуру от архитектуры клиент-сервер?

☐ большее количество компьютеров пользователей
☐ большее количество серверов баз данных
⊞ наличие серверов других типов
☐ другой способ взаимодействия с сервером баз данных

Вариант 2. Где выполняются программы пользователя в трехзвенной архитектуре?

☐ на компьютере пользователя
☐ на сервере баз данных
⊞ на компьютере пользователя и сервере приложений
☐ на сервере приложений

Вариант 3. Что делает сервер приложений?

☐ выполняет прикладные программы пользователя
⊞ формирует запросы к базе данных и обрабатывает результаты запросов
☐ формирует интерфейс пользователя
☐ отображает результаты обработки на компьютере пользователя

Задача 7. Сравнение архитектуры клиент-сервер с файл-серверной архитектурой.

Вариант 1.
Какие черты характерны для компьютеров-клиентов в архитектуре клиент-сервер по сравнению с файл-серверной архитектурой?

☐ увеличение объема прикладной программы
☐ уменьшение объема прикладной программы
⊞ уменьшение объема производимых вычислений
⊞ увеличение объема занимаемой памяти
☐ уменьшение объема занимаемой памяти

Вариант 2.
Как меняется объем данных, передаваемых по локальной сети в архитектуре клиент-сервер по сравнению с файл-серверной архитектурой?

☐ немного уменьшается
☐ увеличивается
☐ остается таким же
⊞ существенно уменьшается

Вариант 3.
За счет чего улучшаются характеристики целостности и безопасности данных?

☐ из-за уменьшения объема передаваемых данных
☐ за счет более эффективного формирования запросов
☐ за счет реализации соответствующих функций СУБД на клиентских компьютерах
⊞ за счет реализации соответствующих функций СУБД на сервере баз данных

Задача 8. Как классифицируются современные СУБД?

Вариант 1.
Какие СУБД относятся к клиент-серверным?

☐ ACCESS
⊞ MS SQL-сервер
⊞ ORACLE
⊞ DB2

Вариант 2.
Какие СУБД относятся к файл-серверным?

⊞ ACCESS
☐ MS SQL-сервер
☐ ORACLE
⊞ FoxPro

Вариант 3.
Какие СУБД используются для организации баз данных в крупных организациях (относятся к промышленным)?

☐ ACCESS
⊞ MS SQL-сервер
⊞ ORACLE
☐ FoxPro

Литература

1. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е изд. СПб.: Питер, 2003.
2. Шумаков П.В. Delphi 3 и создание приложений баз данных. М.: Изд-во «Нолидж», 1998.
3. Мамаев Е. Microsoft SQL Server 2000 в подлиннике. СПб.: Изд-во BHV, 2001.

Лекция 4. Различные представления о данных в базах данных. Основные этапы проектирования баз данных

В лекции рассматриваются различные представления о данных в базах данных. Описываются модели данных (внешнее представление, концептуальная модель, структура хранения) и основные этапы проектирования базы данных. Рассматривается жизненный цикл проектирования базы данных.

Ключевые термины: концептуальные требования пользователя, концептуальная модель, модель данных СУБД, логическая модель, обобщенное представление о данных, внешнее представление, трехуровневая архитектура базы данных, проектирование баз данных, основные этапы проектирования базы данных.

Цель лекции: Показать существование различных представлений о данных (различных моделей) у разных групп лиц, работающих с данными. Рассмотреть отражение этих представлений в трехуровневой архитектуре базы данных (внешний уровень, концептуальный уровень, внутренний уровень), сформулировать достоинство трехуровней архитектуры. Выделить основные этапы пректирования базы данных как процесса построения вышеуказанных моделей.

4.1. Различные представления о данных в базах данных

Создание базы данных предполагает интеграцию данных, предназначенных для решения нескольких прикладных задач разных пользователей. Соответственно, при интеграции данных должны учитываться требования к данным каждого пользователя, основанные на его представлении о данных и связях между ними. Далее эти требования должны обобщаться в единое представление, которое и будет служить основой для построения единой базы данных (рис. 4.1).

Обобщение представлений всех пользователей о данных называется концептуальной моделью (схемой) БД. Концептуальная модель представляет информационное описание предметной области с учетом логических взаимосвязей, поэтому её еще называют инфологической (информационно-логической) моделью. В модели отсутствуют какие-либо понятия, связанные с ЭВМ, памятью ЭВМ, способами размещения данных в памяти ЭВМ, и, по сути, это модель только предметной области.

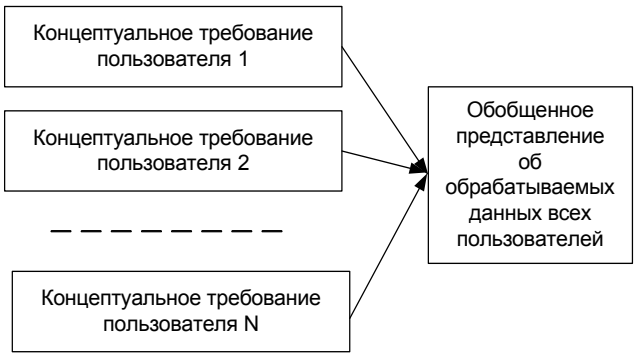


Рис. 4.1. Обобщение представления пользователей о данных

Как уже отмечалось, для создания базы данных и работы с ней используется система управления базами данных. ***Каждая конкретная СУБД поддерживает определенный вид данных (форматов записей и отношений), называемый моделью данных СУБД.***

Следующий этап разработки базы данных предполагает выбор представления концептуальной модели с помощью модели данных конкретной СУБД. ***Полученное таким образом представление концептуальной модели называется логической моделью БД. Или другими словами, логическая модель – это концептуальная схема, специфицированная в языке конкретной СУБД.*** Логическая модель представляет данные и элементы данных вне зависимости от их содержания и среды хранения. Далее разработчик системы средствами СУБД отображает полученную логическую модель БД в память ЭВМ и определяет методы доступа. Полученное представление данных в памяти ЭВМ называется внутренним представлением или структурой хранения. Прикладные программы работают с логической моделью, причем каждому пользователю представляется подмножество этой логической модели (подсхема), отражающее его представление о предметной области. Каждая прикладная программа «видит» и обрабатывает только те данные, которые необходимы именно ей.

Соответствующее «видение» данных прикладными программами (пользователями) представляет собой внешние представления. Взаимосвязь вышеуказанных моделей изображена на рис.4.2.

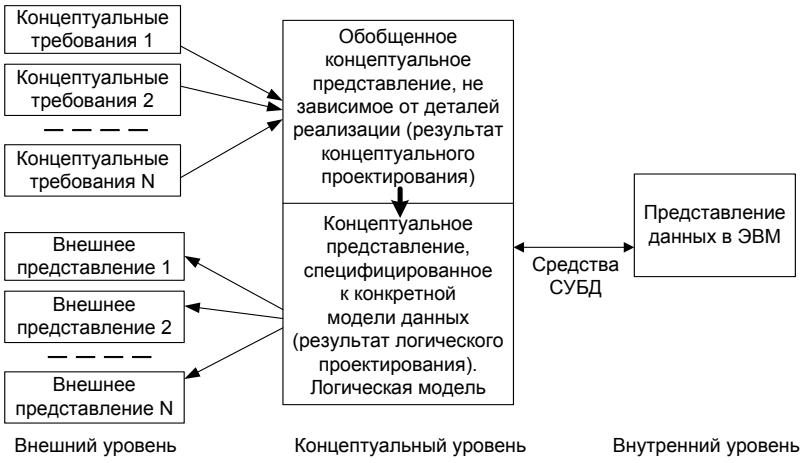


Рис. 4.2. Различные представления о данных в БД

На данной схеме выделены три различных уровня описания данных (внешний, концептуальный, внутренний). ***Эти уровни формируют так называемую трехуровневую архитектуру*** ANSI/SPARC, предложенную в 1975 г. Комитетом планирования стандартов и норм SPARC (Standards Planning and Requirements Committee) Национального института стандартизации США (American National Standards Institute – ANSI). Основная цель этой архитектуры состоит в отделении пользовательского представления о данных в базе данных от их физического представления. Использование таких представлений о данных позволяет обеспечить выполнение основного требования к БД – независимости программ и данных. При изменении прикладных программ может измениться соответствующее внешнее пред-

ставление, но логическая модель данных не изменяется и, соответственно, не будут изменяться другие прикладные программы. При изменении внутреннего представления (структур хранения) логическая модель не изменяется, соответственно, не изменяются прикладные программы.

Использование соответствующих представлений также позволяет четко разграничить полномочия различных лиц, работающих с базой данных.

Соответствующие представления позволяют описать «видение» базы данных разными лицами, работающими с ней:

- внешнее представление – представление специалиста предметной области (пользователя);
- внешнее представление и логическая модель – представление прикладного программиста, разрабатывающего конкретное приложение для пользователя;
- логическая модель и внутреннее представление – представление системного программиста, администрирующего базу данных.

4.2. Основные этапы проектирования базы данных

Проектирование данных (базы данных) представляет собой процесс последовательного отображения исследуемых явлений реального мира в виде данных в памяти ЭВМ (рис. 4.3.).

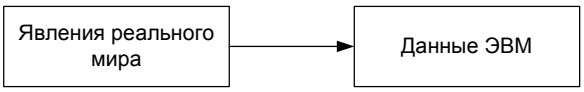


Рис. 4.3. Общая схема пректирования

Конкретные явления реального мира, представляющие интерес для проводимого исследования, будем называть предметной областью.

Проектирование (моделирование) базы данных представляет собой многоэтапный процесс. Основные этапы этого процесса приведены на рис. 4.4.).

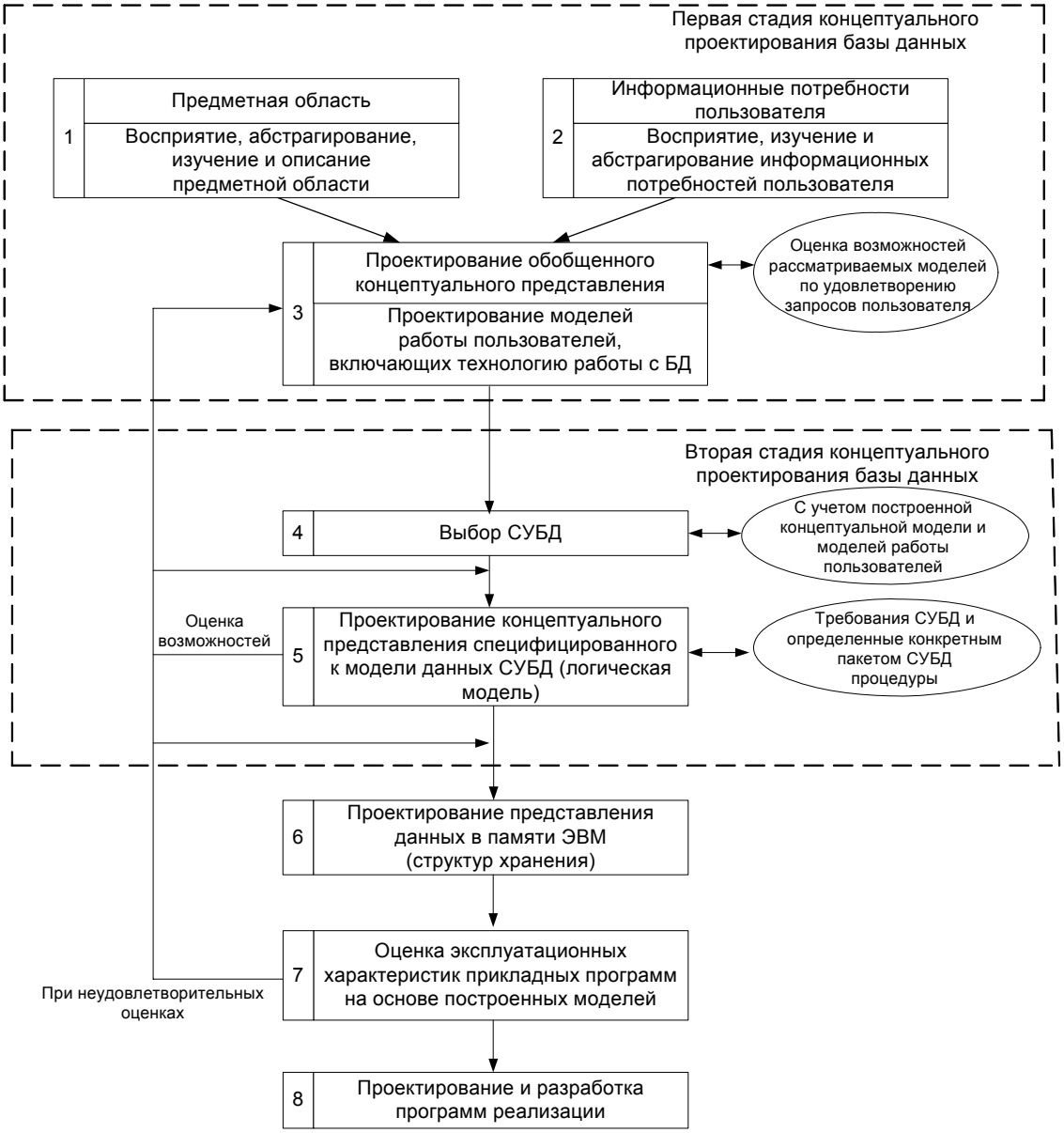


Рис. 4.4. Этапы проектирования базы данных

Подробно действия, отраженные на приведенном рисунке, будут рассмотрены в следующих лекциях. Здесь дадим лишь краткие комментарии к соответствующим блокам.

В блоках 1,2 необходимо особое внимание обратить на слово «абстрагирование». Имеется ввиду, что проектирование базы данных нужно вести не под конкретный документ, обрабатываемый пользователем, и не под конкретные действия пользователя с этим документом, а под обобщенный (абстрактный) образ документов и обобщенные (абстрактные) действия пользователей. Например, рассматривать документ не с конкретными числами строк и столбцов, а с абстрактными числами n и m; вместо требуемого пользователем поиска по конкретному полю (например, фамилии) рассматривать поиск по любому полю и т. д. Это очень важно, так как конкретные формы документов и действия пользователей при работе с ними достаточно часто изменяются. В этом случае при проектировании базы данных под

конкретные формы документов и конкретные действия придется перепроектировать базу данных. что связано с существенными временными и стоимостными затратами.

Очень важным является выбор СУБД (блок 4), от которого в значительной степени зависит работоспособность построенной базы данных. Проблема выбора СУБД уже обсуждалась в лекции 3. Заметим здесь, что выбор СУБД зависит от количества форм документов, от сложности связей между данными, от объема обрабатываемых данных, от количества пользователей, работающих с БД и т. д.

Ранее отмечалось, что отображение логической модели базы данных в структуру хранения (представление данных в памяти компьютера) осуществляется системой управления базой данных. Тем не менее, во многих СУБД для повышения эффективности функционирования базы данных представляется возможность выбора ряда параметров, управляющих представлением данных в памяти компьютера. Выбор таких параметров и подразумевается в блоке 6.

Заметим, что очень важно при проектировании базы данных делать оценки ее возможной работоспособности. Так, по завершении проектирования обобщенного концептуального представления нужно попытаться оценить необходимое число производимых операций с элементами моделей при реализации возможных запросов пользователей. При невозможности в рамках построенной модели ответить на какой-то запрос пользователя или при значительном числе производимых при этом операций (что приведет к невозможности реализации соответствующего запроса в реальном масштабе времени) необходим возврат по схеме рис. 4.4. на шаг назад (построение более эффективного обобщенного концептуального представления). Аналогичные оценки необходимо делать и при завершении других этапов проектирования (блоки 5, 7). При этом возможен возврат назад на один или несколько шагов. Так, например, при проектировании логической модели (блок 5) не удается достичь адекватного представления концептуальной модели средствами модели данных СУБД. В этом случае необходимо либо вернуться на шаг назад и выбрать другую СУБД, либо вернуться к блоку 3 и изменить вид концептуальной модели. Так же, если полученные при реализации блока 7 оценки эксплуатационных характеристик не отвечают требованиям пользователя, возможны пересмотры всех ранее полученных решений (блоки 7, 6, 5, 4, 3). Кроме этого, необходим возврат на проектирование обобщенного концептуального представления при изменении внешних требований пользователей, а также при выявленных ошибках проектирования.

Краткие итоги: Рассмотрены различные представления о данных в базах данных - модели обрабатываемых данных (внешнее представление, концептуальная модель, структура хранения). Представлено отражение этих представлений в трехуровневой архитектуре базы данных (внешний уровень, концептуальный уровень, внутренний уровень), сформулировано достоинство трехуровней архитектуры. Описаны основные этапы проектирования базы данных как процесса построения вышеуказанных моделей и жизненный цикл проектирования базы данных (создание, апробация, исправление ошибок и улучшение характеристик, опытная эксплуатация).

Вопросы данной лекции рассматриваются в [1-6].

Контрольные тесты

Задача 1.Что такое концептуальная модель?

Вариант 1.
Что такое концептуальная модель?

☐ Интегрированные данные
☐ база данных
⊞ обобщенное представление пользователей о данных
☐ описание представления данных в памяти компьютера

Вариант 2.
Что входит в представление концептуальной модели?

⊞ информационное описание предметной области
⊞ логические взаимосвязи между данными
☐ описание представления данных в памяти компьютера
☐ описание решаемых прикладных задач

Вариант 3.
Как соотносятся понятия информационно-логической модели и обобщенного концептуального представления?

⊞ одно и тоже
☐ это разные понятия
☐ обобщенное концептуальное представление является частью информационно-логической модели
☐ информационно-логическая модель является частью обобщенного концептуального представления

Задача 2. Что такое логическая модель базы данных?

Вариант 1.
Как соотносятся понятия логической модели и концептуальной модели?

☐ это разные понятия
⊞ логическая модель это вариант представления концептуальной модели
☐ это одно и то же
☐ логическая модель является частью концептуальной модели

Вариант 2.
Какая связь между логической моделью базы данных и СУБД?

☐ это не связанные понятия
⊞ логическая модель базы данных использует спецификации СУБД
⊞ СУБД отображает логическую модель базы данных в структуру хранения
☐ логическая модель базы данных описывает структуру хранения данных системой управления базами данных

Вариант 3.
Какое описание данных используется прикладными программами при работе с базой данных?

☐ описание структуры представления базы данных в памяти компьютера
☐ описание структуры хранения данных системой управления базами данных
⊞ описание логической модели данных
☐ описание данных в прикладных программах

Задача 3. Какие уровни выделяются в архитектуре базы данных?

Вариант 1.
Как называются уровни архитектуры базы данных?

☐ нижний
⊞ внешний
⊞ концептуальный
⊞ внутренний
☐ верхний

Вариант 2.
Какой из уровней используется специалистом предметной области?

☐ нижний
⊞ внешний
☐ концептуальный
☐ внутренний
☐ верхний

Вариант 3.
Какой из уровней используется прикладным программистом?

☐ нижний
☐ внешний
⊞ концептуальный
☐ внутренний
☐ верхний

Задача 4. Из каких составляющих элементов состоят уровни архитектуры базы данных?

Вариант 1.
Какие понятия соответствуют внешнему уровню архитектуры базы данных?

⊞ концептуальные требования пользователей
⊞ внешние представления пользователей
☐ концептуальная модель
☐ обобщенное представление

Вариант 2.
Какие понятия соответствуют концептуальному уровню архитектуры базы данных?

☐ концептуальные требования пользователей
⊞ логическая модель базы данных
⊞ концептуальная модель
☐ обобщенное представление пользователей

Вариант 3.
Какие понятия соответствуют внутреннему уровню архитектуры базы данных?

☐ логическая модель базы данных
☐ обобщенное представление пользователей
⊞ структура хранения данных
⊞ методы доступа к данным

Задача 5. Что представляет собой процесс проектирования базы данных?

Вариант 1.
Основные этапы проектирования базы данных:

⊞ изучение предметной области
⊞ проектирование обобщенного концептуального представления
⊞ проектирование концептуального представления, специфицированного к модели данных СУБД (логической модели)
☐ разработка прикладных программ

Вариант 2.
Из каких составляющих состоит процесс проектирования концептуальной модели?

⊞ проектирование обобщенного концептуального представления (инфологической модели)
⊞ выбор СУБД
⊞ проектирование концептуального представления, специфицированного к модели данных СУБД (логической модели)
☐ проектирование представления данных в памяти компьютера (структур хранения)

Вариант 3.
Какие действия выполняются на этапе проектирования структур хранения?

☐ выбор СУБД
☐ разработка прикладных программ
☐ выбор способа размещения данных в памяти компьютера
⊞ выбор параметров размещения данных в памяти компьютера, представляемых СУБД

Задача 6. Какие этапы входят в первую и вторую стадию концептуального проектирования?

Вариант 1.
Из каких этапов состоит первая стадия концептуального проектирования?

- ⊞ изучение предметной области
- ⊞ проектирование обобщенного концептуального представления
- ☐ проектирование концептуального представления, специфицированного к модели данных СУБД (логической модели)
- ☐ проектирование представления данных в памяти компьютера (структур хранения)
- ☐ разработка прикладных программ

Вариант 2.
Какие этапы проектирования базы данных не входят в первую стадию концептуального проектирования?

- ☐ проектирование обобщенного концептуального представления (инфологической модели)
- ⊞ выбор СУБД
- ⊞ проектирование концептуального представления, специфицированного к модели данных СУБД (логической модели)
- ⊞ проектирование представления данных в памяти компьютера (структур хранения)

Вариант 3.
Какие этапы проектирования базы данных входят во вторую стадию концептуального проектирования?

- ☐ изучение предметной области
- ☐ проектирование обобщенного концептуального представления
- ⊞ проектирование концептуального представления, специфицированного к модели данных СУБД (логической модели)
- ⊞ проектирование представления данных в памяти компьютера (структур хранения)

Задача 7. Что понимается под понятием «абстрагирование» при описании предметной области и информационных потребностей пользователя?

Вариант 1.
Что понимается под термином «абстрагирование» при описании предметной области

- ☐ описание форм конкретных обрабатываемых документов
- ☐ описание абстрактного документа, не связанного с рассматриваемой предметной областью
- ⊞ описание документов, представляющих абстрактный образ обрабатываемых документов
- ☐ описание обобщенного представления действий всех пользователей

Вариант 2.
Что понимается под термином «абстрагирование» при описании информационных потребностей пользователей?

- ☐ описание конкретных задач пользователя при работе с базой данных
- ☐ описание абстрактных действий с базой данных, не связанных с предметной областью
- ⊞ описание абстрактных действий с базой данных, обобщающих действия всех пользователей
- ☐ абстрактное описание документов, с которыми работают все пользователи

Вариант 3.
Что не соответствует понятию «абстрагирование» ?

- ⊞ описание форм конкретных обрабатываемых документов
- ☐ описание абстрактного документа, не связанного с рассматриваемой предметной областью
- ☐ описание документов, представляющих абстрактный образ обрабатываемых документов
- ⊞ описание конкретных задач пользователя при работе с базой данных
- ☐ описание абстрактных действий с базой данных, не связанных с предметной областью
- ☐ описание абстрактных действий с базой данных, обобщающих действия всех пользователей
- ☐ абстрактное описание документов, с которыми работают все пользователи

Задача 8. С чем связана необходимость возврата к предыдущим этапам проектирования базы данных?

Вариант 1.
Как необходимо оценивать результат завершенного этапа проектирования базы данных?

⊞ по возможности ответа на все возможные запросы пользователей
⊞ по числу элементарных действий, необходимых для ответа на все возможные запросы пользователей
☐ по отсутствия дублирования информации
⊞ по адекватности представления предметной области

Вариант 2.
Что в процессе проектирования базы данных обуславливает необходимость возврата на начало этапа или на предыдущие этапы?

⊞ ошибки проектирования
⊞ изменение требований пользователей
⊞ невозможность ответа на все возможные запросы пользователей
⊞ слишком большое число элементарных действий, необходимых для ответа на все возможные запросы пользователей

Вариант 3.
Какие этапы проектирования могут повторно пересматриваться?

☐ изучение предметной области
⊞ проектирование обобщенного концептуального представления
⊞ выбор СУБД
⊞ проектирование концептуального представления, специфицированного к модели данных СУБД (логической модели)
⊞ проектирование представления данных в памяти компьютера (структур хранения)

Литература

1. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Щерса. – М.: Мир, 1980. – 664 с.
2. Дейт К.Дж. Введение в системы баз данных: Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с.
3. Ульман Дж. Основы систем баз данных: Пер. с англ. / Под ред. М.Р. Когаловского. – М.: Финансы и статистика, 1983. – 334 с.
4. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
5. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Н.Новгород: Изд-во ННГУ, 2004. 271 с.
6. Толстобров А.П. Управление данными. Учебное пособие. Воронеж: Изд-во Воронежского ГУ, 2007 – 205 с.

Лекция 5. Первая стадия концептуального проектирования базы данных (концептуальное моделирование)

Лекция посвящена моделированию предметной области. Здесь рассматриваются понятия, с помощью которых описывается предметная область, средства графического представления концептуальной модели предметной области в виде ER-диаграммы, основные приемы, используемые при моделировании.

Ключевые термины: информационное описание предметной области, атрибут, сущность, класс сущностей, связь, типы связей, диаграмма сущность-связь, ER-диаграмма, концептуальная модель, этапы построения концептуальной модели, ограничения целостности.

Цель лекции: показать, как описывается предметная область при концептуальном моделировании (с помощью каких понятий, средств представления и приемов построения) и как обеспечивается достоверность информации в базе данных за счет ограничений целостности концептуальной модели.

5.1. Описание информационного представления предметной области. ER-диаграмма.

Иллюстрацию вводимых понятий и этапов проектирования базы данных будем проводить на примере близкой для читателя конкретной предметной области: представление данных о студентах вуза, Дадим краткое описание рассматриваемой предметной области. В вузе имеется несколько факультетов, на каждом их которых ведется подготовка по нескольким специальностям или направлениям. Для каждой специальности на факультете есть свой учебный план, в котором приводится перечень изучаемых учебных курсов с указанием количества часов занятий. Студенты изучают соответствующие дисциплины, сдают экзамены и зачеты, получают оценки.

Чаще всего концептуальная модель представляется в виде диаграммы сущностей – связей (entity – relationship) или ER-диаграммы. Процесс построения ER-диаграммы называется ER-моделированием.

Введем основные понятия, с помощью которых описывается предметная область.

Сущность (Entity) или объект – то, о чем будет накапливаться информация в информационной системе (нечто такое, за чем пользователь хотел бы наблюдать).

Если в системе обрабатывается информация о факультетах, сущностью будет являяться факультет, если о студентах, сущность – студент и т.п.

Имя сущности при ER-моделировании, как правило, записывается заглавными буквами. Каждая сущность обладает определенным набором свойств (рассматриваем только свойства, представляющие интерес для пользователей в рамках проводимого исследования), которые запоминаются в информационной системе. Так, например, в качестве свойств сущности ФАКУЛЬТЕТ можно указать номер вакультета, название факультета, в качестве свойств сущности СТУДЕНТ можно указать фамилию, дату рождения, место рождения, в качестве свойств сущности ЭКЗАМЕН – предмет, дату проведения экзамена, экзаменаторов.

Для информационного описания сущности вводится понятие атрибута.

Атрибут – поименованное свойство (характеристика) сущности. Атрибут представляет собой информационное отображение свойства сущности и принимает конкретное значение из множества допустимых значений. Так, например, для сущности ФАКУЛЬТЕТ атрибут «название» у конкретного экземпляра сущности принимает конкретное значение «вычислительной математики и кибернетики». Таким образом, атрибут представляет информационное описание количественных или качественных свойств сущности, описывает состояние сущности, позволяет идентифицировать сущность. Информация о сущности представляется совокупностью атрибутов. ***Такую совокупность атрибутов часто называют записью об объекте.***

Совокупность сущностей, характеризующихся в информационной системе одним и тем же перечнем свойств, называется классом сущностей (набором объектов). Так, например, совокупность всех сущностей СТУДЕНТ составляет класс сущностей СТУДЕНТ, совокупность всех сущностей ФАКУЛЬТЕТ составляет класс сущностей ФАКУЛЬТЕТ. Класс сущностей описывается перечнем свойств сущностей, составляющих этот класс.

Экземпляром сущности будем называть конкретную сущность (сущность с конкретными значениями соответствующих свойств). Выше мы определили сущность как то, о чем будет накапливаться информация в информационной системе. Это только одна сторона. Информация должна не просто храниться сама по себе, а использоваться для удовлетворения информационных потребностей пользователя. Для реализации подавляющего числа запросов пользователю прежде всего необходимо найти интересующий его экземпляр сущности (с целью обработки, корректировки, удаления). ***Поэтому важнейшим свойством сущности является однозначная идентификация ее экземпляров по одному или группе атрибутов (уникальному идентификатору)***. У сущности ФАКУЛЬТЕТ это, например, номер факультета, у сущности СТУДЕНТ это может быть атрибут «фамилия», если у всех студентов разные фамилии, группа атрибутов «фамилия», «имя», «отчество», или специально введенный уникальный идентификатор, например дополнительно введенный атрибут «код студента».

Наиболее распрстраненным способом представления концептуальной модели является так называемая ER-диаграмма. В разных источниках используются разные системы обозначений в ER-диаграммах. На практике использование различных способов записи ER-диаграмм не представляет особой сложности – беглое ознакомление с соответствующим разделом документации позволяет быстро освоить используемую систему обозначений. В данном пособии в ER-диаграмме класс сущностей будем представлять в виде четырехугольника. В четырехугольнике записано уникальное имя класса сущности (прописными буквами) и имена атрибутов строчными буквами.

Пример класса сущностей СТУДЕНТ и конкретного экземпляра сущности показан на рис. 5.1.

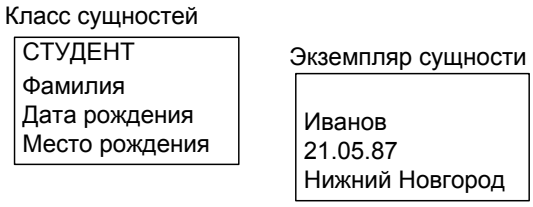


Рис. 5.1.. Класс сущностей и экземпляр сущности

Для реализации иформационных потребностей пользователя недостаточно найти интересующий его экземпляр сущности. Информационные потребности тесно связаны с функциональными взаимоотношениями, существующими в организации (например, необходимо определить, на каком факультете учится конкретный студент). Для реализации таких запросов (информационных потребностей пользователя) используются существующие в предметной области взаимоотношения между сущностями. ***Соответствующие взаимоотношения сущностей выражаются связями (Relationships***). Различают классы связей и экземпляры связей. ***Классы связей – это взаимоотношения между классами сущностей, а экземпляры связи – взаимоотношения между экземплярами сущностей.***

Класс связей может затрагивать несколько классов сущностей. Число классов сущностей, участвующих в связи, называется степенью связи $n = 2, 3, \ldots$ Так, например, класс сущностей СТУДЕНТ связан с классом сущностей ФАКУЛЬТЕТ связью «учится на факультете». Степень этой связи равна двум. При n=2 связь называется бинарной. Заметим, что связь нужно рассматривать как двустороннюю: «студент учится на факультете» и «на факультете учатся студенты». Рассмотрим классификацию бинарных связей. В зависимости от того, сколько экземпляров сущности одного класса связаны со сколькими экземплярами сущности другого класса, различают следующие **типы связей:**

- Связь 1:1. Одиночный экземпляр сущности одного класса связан с одиночным экземпляром сущности другого класса. Примером является связь между классами сущностей ФАКУЛЬТЕТ и УЧЕБНЫЙ ПЛАН ПО СПЕЦИАЛЬНОСТИ ДЛЯ ФАКУЛЬТЕТА (каждому факультету соответствует свой учебный план по специальности или направлению).
- Связь 1:*M*. Единый экземпляр сущности одного класса связан со многими экземплярами сущности другого класса. Примером является связь между классами сущностей ФАКУЛЬТЕТ и СТУДЕНТ (на одном факультете учатся много студентов).
- Связь *M*:*N*. Несколько экземпляров сущности одного класса связаны с несколькими экземплярами сущности другого класса. Примером является связь между классами сущностей ФАКУЛЬТЕТ и СПЕЦИАЛЬНОСТЬ (на факультете может быть несколько специальностей и одна и таже специальность может быть на нескольких факультетах).

Числа, описывающие типы бинарных связей (1:1, 1:*M*, *M*:*N*), обозначают максимальное количество сущностей на каждой стороне связи. Эти числа называются максимальными кар-

динальными числами, а соответствующая пара чисел называется максимальной кардинальностью.

В данном пособии на ER-диаграммах связи между сущностями будем обозначать стрелками, рядом со стрелками указываем имя связи, а также тип связи. Пример ER-диаграммы, представлющего сущности СТУДЕНТ, ФАКУЛЬТЕТ, СПЕЦИАЛЬНОСТЬ и их взаимосвязи приводится на рис. 5.2.

Нпомним, что каждый экземпляр сущности должен уникально идентифицироваться (иметь уникальный идентификатор). Так как могут быть несколько студентов с одинаковой фамилией, введем дополнительный атрибут «код студента». У сущностей ФАКУЛЬТЕТ и СПЕЦИАЛЬНОСТЬ атрибут «номер» является уникальным идентификатором.

Рис.5.2. Пример фрагмента ER-диаграммы.

Заметим, что по этой ER-диаграмме можно указать последовательность действий, производимых при реализации запроса пользователей. Например, для реализации запроса «на каком факультете учится студент Иванов» необходимо выполнить следующие действия: найти среди экземпляров сущности СТУДЕНТ экземпляр с фамилией Иванов, перейти по связи «Студент учится на факультете» к экземпляру сущности ФАКУЛЬТЕТ, значение атрибута «Название» этого экземпляра и есть искомое название факультета. Отметим также, что иногда на ER-диаграммах две связи между сущностями изображают одной двухсторонней стрелкой или просто линией. Заметим, что на приведенной ER-диаграмме не представлены какие-либо способы реализации этих связей (на логическом и, тем более, на физическом уровнях). Соответствующие способы реализации связей зависят от возможностей модели данных конкретной СУБД и будут рассмотрены в следующей лекции (лекция 6) на второй

стадии концептуального проектирования при представлении концептуальной модели средствами модели данных СУБД.

5.2. Построение концептуальной модели в виде ER-диаграммы

5.2.1 Основные этапы построения

Как уже отмечалось, **концептуальная модель** представляет собой обобщение представлений разных пользователей о данных. В связи с этим построение концептуальной модели, как правило, происходит в два этапа. На первом этапе производится сбор и анализ характеристик данных и строятся так называемые модели локальных представлений (локальные модели). Чаще всего локальная модель отражает представление отдельного пользователя (отдельной функциональной задачи). Иногда такая модель может описывать и некоторую независимую область данных нескольких функциональных задач (нескольких приложений). Здесь необходимо отметить, что моделирование представлений отдельных пользователей приводит к снижению уровня интеграции данных, а моделирование совместных представлений группы пользователей – к повышению сложности проектирования. В связи с этим при выборе области данных для локального моделирования приходится выбирать компромиссное решение между вышеуказанными вариантами.

При разработке концептуальной модели, прежде всего, следует определить сущности. С этой целью нужно сделать следующее:

- необходимо понять, какая информация должна храниться и обрабатываться и можно ли это определить как сущность;
- присвоить этой сущности имя;
- выявить атрибуты сущности и присвоить им имя;
- определить уникальный идентификатор сущности.

Выявив сущности, необходимо определить, какие связи имеются между ними.

При определении связей (естественно, рассматриваем только те связи, которые имеют отношение к решаемым задачам обработки данных) необходимо учитывать следующее:

- то, как экземпляр одной сущности связан с экземпляром другой сущности;
- то, как должны быть установлены связи, чтобы была возможность ответа на все запросы пользователей (исходя из их информационных потребностей).

Далее необходимо присвоить связям имена и определить тип связей.

На втором этапе построенные локальные модели объединяются в обобщенную концептуальную модель.

5.2.2. Моделирование локальных представлений

Прежде всего, необходимо отметить, что построенная модель должна удовлетворять ряду требований:

- адекватно отражать представление пользователя о данных;
- давать возможность ответа на возможные запросы пользователя, причем делать это с минимальными затратами по количеству просматриваемых сущностей;

- представлять данные с минимальным дублированием.

Процесс построения модели, удовлетворяющей указанным требованиям, является творческим, и формализовать его, как правило, невозможно. Тем не менее можно указать некоторые способы порождения вариантов при моделировании. Выбор одного из таких вариантов на основе оценок объемов дублирования и числа просматриваемых объектов при ответах на запросы пользователей позволяет улучшить эксплуатационные характеристики проектируемой базы данных.

Вариативность моделирования обусловливается неоднозначностью выбора сущностей, атрибутов и связей. В одном варианте можно что-то взять за сущность, в другом варианте это же можно взять за атрибут (несколько атрибутов), в третьем варианте это можно определить как связь. Так, например, ранее мы определили сущеость ФАКУЛЬТЕТ с атрибутами «номер факультета», «название факультета». Введем сущность КАФЕДРА с атрибутами «номер кафедры», «название кафедры». Между этими сущностями есть связь «факультет состоит из кафедр». Возможен другой вариант, в котором вышеуказанная связь представляется через атрибуты сущности (у сущности ФАКУЛЬТЕТ введем дополнительные атрибуты, представляющие номера и названия вскх кафедр этого факультета).

После того как выбран рациональный вариант локальной модели, производится редактирование введенных наименований сущностей, атрибутов и связей. Здесь выполняются следующие действия:

- устраняются расплывчатые наименования (все наименования должны однозначно пониматься каждым пользователем);
- устраняются синонимы (различные наименования одного и того же понятия);
- устраняются омонимы (одно и то же наименование разных понятий).

Эти действия, вообще говоря, носят итерационный характер, т.к. после их выполнения вновь могут возникать и расплывчатые наименования, и синонимы, и омонимы.

5.2.3. Объединение локальных моделей

На этом этапе ранее построенные модели локальных представлений отдельных пользователей (или групп пользователей) объединяются в единую концептуальную модель. Объединение локальных моделей производится следующими путями:

- слияние идентичных элементов;
- установление связей между наборами сущностей разных моделей;
- введение новых агрегированных элементов для представления связей между элементами разных моделей;
- обобщение различных подобных типов сущностей, позволяющее трактовать эти сущности как одну обобщенную сущность.

Рассмотрим каждый из этих путей.

Слияние идентичных элементов

Два или более элементов модели идентичны, если они имеют одинаковое смысловое значение.

Объединение моделей с идентичными элементами осуществляется путем «слияния» этих элементов в один. Два набора сущностей СПЕЦИАЛЬНОСТЬ в модели 1 и 2 имеют одинаковое смысловое значение (рис. 5.3.), и могут быть заменены одним набором сущностей (рис. 5.4.).

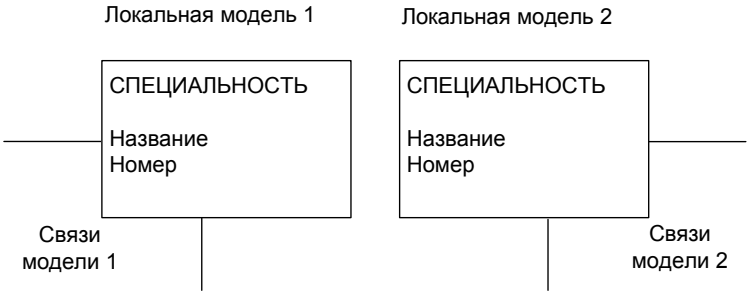


Рис. 5.3.. Модели с идентичным элементом

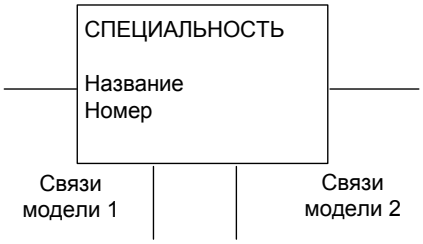


Рис.5.4.. Объединенная модель

Установление связей между наборами сущностей разных моделей

При рассмотрении наборов сущностей объединяемых моделей необходимо выявление связей между ними, т.к. именно эти связи и определяют в конечном итоге интегрированную базу данных.

Введение агрегированных элементов

При объединении моделей связь между элементами разных моделей может рассматриваться как новый элемент.

Рассмотрим в качестве примера моделирование информационного представления сдачи студентом экзаменов. Можно выделить ряд локальных представлений (рис. 5.5.).

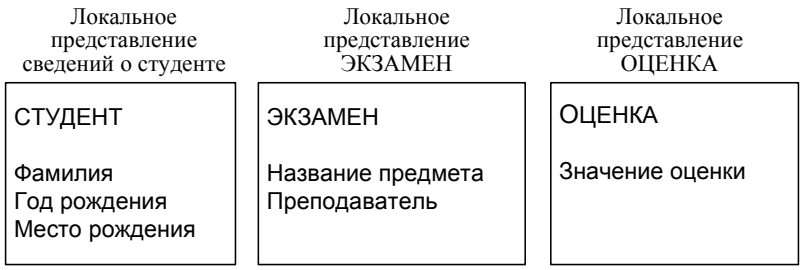


Рис.5.5. Локальные представления

Объединяя локальные представления, устанавливаем новые связи (рис. 5.6.).

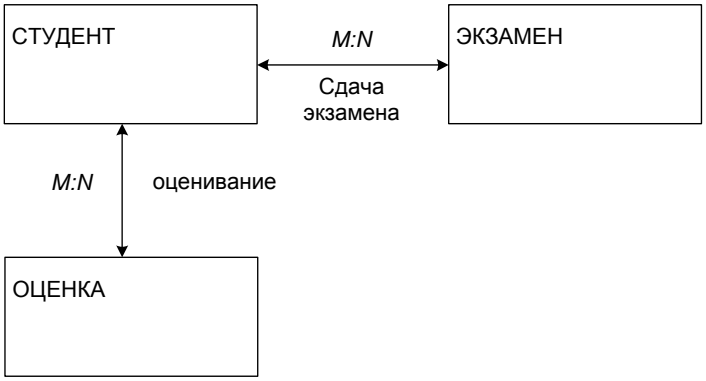


Рис.5.6.. Объединение локальных представлений

Как уже отмечалось, одним из показателей «зрелости» модели является возможность ответа на запросы пользователей, и установление связей преследует именно эту цель. Нетрудно видеть, что какие бы связи в рассматриваемой модели ни вводились, невозможно ответить на запрос «какую оценку получил студент А по дисциплине В». В таком случае необходимо использовать принцип агрегации – необходимую связь между элементами модели ввести как некоторый новый элемент. В данном примере можно определить этот новый агрегированный элемент как ЭКЗАМЕН СТУДЕНТА (рис. 5.7.).

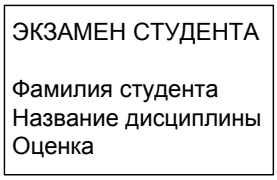


Рис. 5.7. Агрегированный элемент.

Далее процесс объединения локальных моделей продолжается обычным образом.

Обобщение подобных типов сущностей

Рассмотрим локальные модели разных факультетов, например – модель факультета вычислительной математики и кибернетики (ВМК), модель экономического факультета и так далее. В локальную модель факультета ВМК входят сущности СПЕЦИАЛЬНОСТИ ФАКУЛЬТЕТА ВМК и СТУДЕНТЫ ФАКУЛЬТЕТА ВМК, в локальную модель экономического факультета входят, соответственно, сущности СПЕЦИАЛЬНОСТИ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА и СТУДЕНТЫ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА (рис. 5.8.).

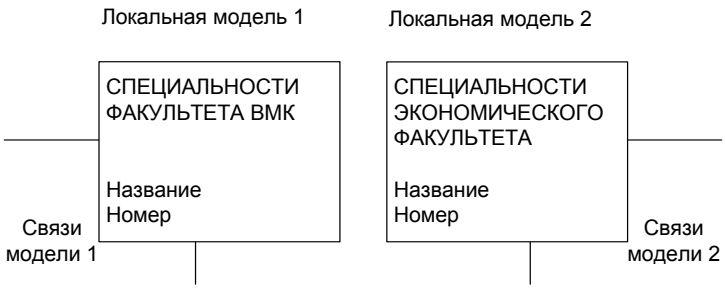


Рис. 5.8. Модели с подобным элементом

Два набора сущностей СПЕЦИАЛЬНОСТИ ФАКУЛЬТЕТА ВМК и СПЕЦИАЛЬНОСТИ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА в моделях 1 и 2 имеют одинаковое смысловое значение и могут быть заменены одним набором сущностей с добавлением нового атрибута – название факультета (рис. 5.9.).

Рис. 5.9. Пример обобщенной сущности

Отметим, что в данном случае подобным образом можно слить и все остальные сущности локальных моделей факультетов, так как сущности СТУДЕНТЫ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА и СТУДЕНТЫ ВМК также имеют одинаковое смысловое значение и их также можно объединить. Однако в общем случае каждая локальная модель может содержать сущности и связи, которых нет в других локальных моделях.

Рассмотрим другой пример. Предположим, что мы храним данные о студентах (фамилия, имя, отчество, курс, группа) и о преподавателях (фамилия, имя, отчество, кафедра, должность). Соответственно, в предметной области выделяем две сущности – СТУДЕНТ и ПРЕПОДАВАТЕЛЬ.

Эти разные сущности можно в некоторых случаях трактовать как подобные. Для обобщения соответствующих сущностей необходимо, прежде всего, обобщить их атрибуты. Заметим, что атрибуты «Фамилия, Имя, Отчество» у обеих сущностей совпадают, атрибуты «Кафедра» и «Курс», «Группа» показывают место работы (учебы) и их можно заменить обобщенным атрибутом «Место работы (учебы)». Атрибут «Должность» можно использовать и у сущности СТУДЕНТ, если в качестве значения соответствующего атрибута использовать значение «студент». Тогда две сущности ПРЕПОДАВАТЕЛЬ и СТУДЕНТ можно трактовать как подобные и заменить их на обобщенную сущность. Дадим этой обобщенной сущности название КАДРОВАЯ ЕДИНИЦА (рис. 5.10.).

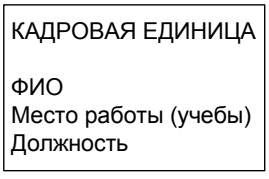


Рис. 5.10. Пример обобщенной сущности

У студента атрибут «Место работы (учебы)» будет принимать значение соответствующее атрибутам «Курс. Группа», у преподавателя – название кафедры. Обобщенная модель представлена на рис. 5.11.

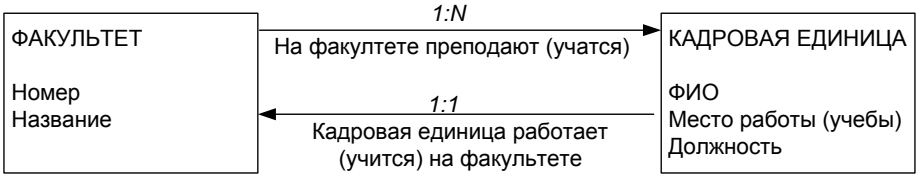


Рис. 5.11. Обобщенная модель

В этом случае почти в два раза упрощается структура концептуальной модели, и соответственно, структура базы данных. Для работы с данными о преподавателях и студентах достаточно одного набора программ. Таким образом, обобщение подобных типов объектов может существенно сократить последующие затраты на программирование.

В процессе объединения локальных представлений, как и при локальном моделировании, производится редактирование наименований (т.к. здесь появляются новые наименования). Процесс объединения также носит итерационный характер и продолжается до тех пор, пока не будут интегрированы все представления, согласованы и устранены все противоречия, отредактированы все наименования. Полученное в результате объединения локальных представлений обобщенное представление и является концептуальной моделью.

5.3. Ограничения целостности

Под целостностью базы данных понимается то, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную область (правильная) информация.

Огромный объем данных, вводимых в базу данных (причем разные данные могут вводиться разными пользователями), обусловливает большое число ошибок ввода (занесения). Заметим, что при традиционной «бумажной» обработке информации также достаточно часто встречаются данные, записанные неверно. Но человек, работая с определенными данными, неявно использует для контроля имеющиеся у него представления об этих данных. Например, сотрудник отдела кадров, увидев в карточке работника год рождения 1693, сразу заметит эту ошибку и предположит, что просто переставлены две цифры и реальный год рождения 1963. То есть в представлениях сотрудника заключены некоторые логические ограничения на данные. Очевидно, что для контроля правильности вводимых данных при работе с базой данных целесообразно сформировать и использовать ограничения.

Соответствующие ограничения обычно разделяют на 3 группы: внешние, специально конструируемые и внутренние. К предметной области относятся первые две группы, которые мы кратко охарактеризуем в этом подразделе. Внутренние ограничения относятся уже к модели данных и будут рассматриваться в разделе, посвященном модели данных.

Внешние ограничения.

Эти ограничения связаны с адекватностью отражения предметной области. Например, сотрудник организации не может быть моложе 17 и старше 90 лет. Соответствующее ограничение на год рождения (GR) можно записать следующим образом:

Текущий год – 17 > GR > Текущий год – 90.

Одним из способов задания таких ограничений является перечисление конечного множества допустимых значений какого-либо атрибута (так называемый «перечислимый» тип данных). Например, должность преподавателя в вузе может принимать одно из следующих значений: профессор, доцент, старший преподаватель, преподаватель, ассистент. Вводимое значение должности для конкретного экземпляра, не совпадающее с одним из перечисленных значений, является ошибкой.

Ограничения, описанные с помощью специальных конструкций.

Например, в базу данных вуза вводятся данные о числе студентов и преподавателей. По нормативным документам задано конкретное значение отношения числа студентов к числу преподавателей. Проверку этого отношения можно использовать для контроля достоверности данных. Такие конструкции строятся исходя из специфики данных рассматриваемой предметной области. Можно, например, построить много конструкций следующего вида: сумма значений по заданному атрибуту по всем экземплярам сущностей должна совпадать со значением определенного атрибута в экземпляре другой сущности.

Таким образом, на стадии ER-моделирования для повышения достоверности данных необходимо сформулировать соответствующие ограничения на данные. В идеальном случае каждое значение атрибута должно каким-то образом контролироваться. Использование этих ограничений позволяет существенно повысить достоверность данных в базе данных.

Краткие итоги. Рассмотрен процесс моделирования предметной области. Определены используемые при этом основные понятия (сущность, атрибут, идентификатор, связь, типы связей, ER-диаграмма).

Рассмотрены основные этапы моделирования сущностей и связей (моделирование локальных представлений, объединение локальных моделей с ипользованием понятий идентичность, агрегация, обобщение).

Дано понятие ограничений целостности, имеющих непосредственное отношение к предметной области (внешние ограничения; ограничения, описанные с помощью специальных конструкций).

Более подробно с материалом этой лекции можно познакомиться в [1-10].

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. С помощью какого основного понятия описывается то, о чем будет накапливаться информация в информационной системе?

Вариант 1.

Как называется основное понятие, с помощью которого описывается то, о чем будет накапливаться информация в информационной системе?

☐ атрибут
⊞ объект
⊞ сущность
☐ идентификатор

Вариант 2.

Чем отличаются понятия сущность и объект в базах данных?

⊞ одно и тоже
☐ сущность используется для описания объекта
☐ это разные понятия
☐ объект используется для описания сущности

Вариант 3.

Что из следующих примеров можно определить как сущность?

☐ название экзамена
☐ фамилию студента
⊞ факультет
⊞ оценка
⊞ предмет

Задача 2. Какие понятия используются для описания сущности?

Вариант 1.
Как называется понятие, используемое для описания сущности?

⊞ свойство
⊞ атрибут
☐ объект
☐ экземпляр

Вариант 2.
Чем отличаются понятия свойство и атрибут?

☐ одно и то же
⊞ атрибут это свойство, принимающее конкретные значения
☐ свойство используется для описания атрибута
☐ атрибут описывает конкретное свойство

Вариант 3.
Как описывается сущность?

⊞ совокупностью атрибутов
☐ набором экземпляров
☐ совокупностью объектов
⊞ записью об объекте

Задача 3. В чем разница между классом сущностей и экземплярами сущности?

Вариант 1.
Что такое класс сущностей?

☐ набор экземпляров сущностей
⊞ совокупность сущностей с одинаковыми свойствами
☐ совокупность атрибутов
☐ совокупность сущностей с одинаковыми значениями атрибутов

Вариант 2.
Какое понятие используется для представления конкретной сущности?

⊞ экземпляр сущности
☐ атрибут сущности
☐ идентификатор сущности
☐ класс сущности

Вариант 3.
Что такое экземпляр сущности?

⊞ сущность с конкретными значениями свойств
☐ совокупность атрибутов
☐ сущность с конкретным названием
☐ сущность с определенным именем идентификатора

Задача 4. Как определяется понятие связи?

Вариант 1.
Чем определяется существование связи между сущностями

⊞ функциональными взаимоотношениями между сущностями
⊞ информационными связями между сущностями
☐ информационными потребностями пользователя
⊞ свойствами сущностей

Вариант 2.
Какие бывают типы связей?

⊞ один к одному
⊞ один к многим
⊞ многие к многим
⊞ многие к одному

Вариант 3.
Между какими элементами рассматриваются связи?

⊞ между сущностями
⊞ между экземплярами сущностей
☐ между атрибутами
⊞ между классами сущностей

Задача 5. В чем разница между классами связей и экземплярами связей?

Вариант 1.
Что такое класс связей?

⊞ взаимоотношения (набор связей) между классами сущностей
☐ набор связей типа «многие к многим»
☐ набор связей типа «один к многим»
☐ набор связей между экземплярами сущностей

Вариант 2.
Что такое экземпляры связей?

⊞ взаимоотношения между экземплярами сущностей
☐ взаимоотношения (набор связей) между классами сущностей
☐ взаимоотношения между атрибутами
☐ взаимоотношения между сущностями

Вариант 3.
Что такое степень связи?

⊞ число классов сущности, участвующих в связи
☐ максимальное количество экземпляров сущностей на каждой стороне связи
☐ число экземпляров сущности, участвующих в связи
☐ количество связей между экземплярами сущностей

Задача 6. Что такое ER-диаграмма?

Вариант 1.
Для чего используется ER-диаграмма?

⊞ графическое представление концептуальной модели
⊞ графическое представление сущностей и связей между ними
⊞ графическое представление обобщенного представления пользователей о данных
☐ графическое представление всех сущностей
☐ графическое представление связей

Вариант 2.
Что представляется на ER-диаграмме

⊞ сущности
⊞ связи
⊞ типы связей
⊞ атрибуты
☐ экземпляры сущностей

Вариант 3.
Как на ER-диаграмме представляются способы реализации связей?

⊞ не представляются
☐ в виде адресных ссылок
☐ представляются на логическом уровне
☐ представляются на физическом уровне

Задача 7. Основные этапы построения концептуальной модели.

Вариант 1.
Какой порядок действий при построении концептуальной модели?

⊞ определение сущностей, определение атрибутов, установление связей
☐ определение атрибутов, определение сущностей, установление связей
☐ выбор связей, определение сущностей, определение атрибутов
☐ выбор экземпляров сущностей, установление связей между экземплярами

Вариант 2.
Что такое локальное моделирование?

☐ моделирование одной сущности
☐ моделирование представления одного пользователя
⊞ моделирование представлений группы пользователей, использующих общие данные
☐ моделирование представлений группы пользователей, использующих разные данные

Вариант 3.
Как редактируются локальные модели?

⊞ устраняются расплывчатые наименования сущностей и атрибутов
⊞ устраняются синонимы в наименованиях сущностей и атрибутов
⊞ устраняются омонимы в наименованиях сущностей и атрибутов
⊞ изменяются наименования сущностей и атрибутов

Задача 8. Как локальные модели объединяются в обобщенную концептуальную модель?

Вариант 1.
Какие приемы используются при объединении локальных моделей?

☐ отбрасываются некоторые идентичные элементы
⊞ сливаются идентичные элементы
⊞ локальные модели объединяются по новым связям
☐ локальные модели объединяются по имеющимся в них связям
⊞ подобные типы сущностей обобщаются
☐ подобные типы сущностей исключаются
⊞ вводятся новые сущности

Вариант 2.

Что такое «введение агрегированного элемента»?

☐ объединение нескольких сущностей
☐ разбиение сущности на несколько сущностей
⊞ определение новой сущности
⊞ рассмотрение связи как новой сущности

Вариант 3.
Что такое «обобщение подобных типов сущностей»?

☐ подобные типы сущностей сливаются
☐ подобные типы сущностей удаляются
⊞ определяется новая сущность
⊞ подобные типы сущностей заменяются на другую сущность

Задача 9. Что такое ограничения целостности?

Вариант 1.
Зачем нужны ограничения целостности?

⊞ для обеспечения правильного ввода данных в базу данных
⊞ для обеспечения достоверной информации в базе данных
☐ для проверки правильности работы прикладных программ
☐ для уменьшения ошибок при поиске данных

Вариант 2.
Какие существуют типы ограничений целостности ?

⊞ внешние
⊞ внутренние
☐ специально конструируемые в прикладных программах
⊞ специально конструируемые в программах СУБД

Вариант 3.
Откуда берутся внешние и специально конструируемые ограничения?

⊞ определяются предметной областью
☐ определяются СУБД
☐ определяются прикладными программами
⊞ определяются пользователем
☐ определяются программистом

Литература

1. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Щерса. – М.: Мир, 1980. – 664 с.
2. Крёнке Д. Теория и практика построения баз данных. 8-е изд. – СПб.: Питер, 2003. – 800 с.
3. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
4. Карпова Т. Базы данных. Модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
5. Хансен Г., Хансен Дж. Базы данных: разработка и управление: Пер. с англ. – М.: ЗАО «Издательство «БИНОМ», 1999. – 704 с.
6. Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н. Базы и банки данных. М.: ВШ, 1986, 1992.
7. Ульман Дж. Д., Уидом Дж. Введение в системы баз данных: Пер. с англ. – М.: Лори, 2000. – 374 с.
8. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Н.Новгород: Изд-во ННГУ, 2004. 271 с.
9. Толстобров А.П. Управление данными. Учебное пособие. Воронеж: Изд-во Воронежского ГУ, 2007 – 205 с.
10. Горев А., Ахаян Р., Макашарипов С. Эффективная работа с СУБД. СПб.: Питер, 1997. – 700 с.

Лекция 6. Вторая стадия концептуального проектирования (Модели данных СУБД. представление концептуальной модели средствами модели данных СУБД)

Лекция посвящена второй стадии концептуального проектирования – представлениию концептуальной модели в терминах модели данных определенной СУБД. Здесь дается общее понятие модели данных СУБД, рассматриваются типовые классические модели данных, рассматриваются принципы автоматизированного проектирования баз данных.

Ключевые термины: модель данных, сетевая модель, иерархическая модель, реляционная модель, многомерная модель, представление концептуальной модели, логическое проектирвание, автоматизированное проектирование баз данных.

Цель лекции: дать общее представление о модели данных СУБД как средства для представления концептуальной модели при создании базы данных, рассмотреть типовые модели данных (сетевая модель, иерархическая модель, реляционная модель, многомерная модель), показать как представляяется концептуальная модель в разных СУБД, рассмотреть основные принципы работы средств автоматизированного проектирования баз данных.

6.1. Представление концептуальной модели средствами модели данных СУБД

Общие представления о моделях данных СУБД

В соответствии с основными этапами проектирования базы данных после построения концептуальной модели выбирается система управления базой данных, с помощью которой будет организована база данных и работа с ней. Каждая СУБД поддерживает определенные виды и типы данных, а также средства представления связей между данными, составляющими модель данных СУБД. Вторая стадия проектирования базы данных состоит в представлении построенной на предидущей стадии концептуальной модели средствами модели данных СУБД или в отображении концептуальной модели в модель данных СУБД. ***Этот этап часто называют логическим проектированием базы данных. Полученная при этом модель часто также называется концептуальной моделью или схемой (но специфицированной к понятиям модели данных СУБД). В некоторых источниках полученную модель называют логической структурой данных или моделью данных базы данных.***

Можно по-разному характеризовать понятие **модели данных СУБД**. С одной стороны, ***модель данных СУБД – это способ структурирования данных, которые рассматриваются как некоторая абстракция в отрыве от предметной области.*** С другой стороны, ***модель данных СУБД– это инструмент представления концептуальной модели предметной области и динамики ее изменения в виде базы данных.***

Учитывая обе вышеуказанные стороны, определим основные структуры моделей данных СУБД, используемые для представления концептуальной модели предметной области (сущностей, атрибутов, связей).

Элемент данных (поле) – наименьшая поименованная единица данных. Используется для представления значения атрибута.

С элементом данных неразрывно связано понятие «тип данных», который может принимать соответствующее поле. В разных СУБД могут использоваться разные типы данных, наиболее распространенными из которых (используемые во многих СУБД) являются следующие: числовой (numeric), символьный (char), дата (date) и т.д.

Запись – поименованная совокупность полей. Используется для представления совокупности атрибутов сущности (записи о сущности).

Экземпляр записи – запись с конкретными значениями полей.

Первичный ключ – минимальный набор полей записи, однозначно идентифицирующих экземпляр записи файла.

Файл – поименованная совокупность экземпляров записей одного типа. Используется для представления однородного набора сущностей.

Набор файлов – поименованная совокупность файлов, обрабатываемых в системе. Используется для представления нескольких наборов сущностей.

Введем понятие «группа», обобщающее понятия «файл» и «запись».

Группа – это поименованная совокупность элементов данных или элементов данных и других групп.

Важнейшим понятием концептуальной модели является понятие связи между сущностями (наборами сущностей). В моделях данных СУБД соответствующее понятие отражается понятием **«групповое отношение»**.

Групповое отношение – поименованное бинарное отношение, заданное на двух множествах экземпляров рассматриваемых групп. По характеру бинарных связей различают групповые отношения вида 1:1, 1:M, M:1, M:N. Пары чисел называют коэффициентами группового отношения. В групповом отношении один член группы назначается владельцем отношения, другой – членом.

База данных – поименованная совокупность экземпляров групп и групповых отношений.

Для представления группового отношения используется две формы:

а) **Графовая**. Группы изображаются вершинами графа, связи между группами – дугами, направленными от группы-владельца к группе-члену с указанием имени отношения и коэффициента.

По типу графов различают:

- *иерархическую модель (**граф без циклов – дерево**);*
- *сетевую модель (ориентированный граф общего вида).*

б) **Табличная**. Связь между группами изображается таблицей, столбцы которой представляют ключи соответствующих групп. Для формального описания таблицы используется математическое (теоретико-множественное) понятие отношения. Соответствующая модель данных называется реляционной моделью.

Модель данных СУБД описывается следующим образом:

- определены возможные типы и характеристики логических структур данных (полей, записей, файлов);
- заданы правила составления структур более общего типа из структур более простых типов (например, записей из полей, файлов из записей и т.д.);
- определен способ представления связей (отношений) между файлами и записями) с помощью дополнительных полей ;
- определены возможные действия над структурами и правила их выполнения, включающие:
  - основные элементарные операции над данными;
  - обобщенные операции (процедуры);
  - средства контроля относительно простых условий корректности операций добавления, обновления или удаления данных (ограничения), реализуемые автоматически запускаемыми при выполнении вышеуказанных операций специальными процедурами (триггерами);
  - средства контроля сколь угодно сложных условий корректности выполнения определенных действий (правила);
  - специальный класс процедур (триггеры).

В качестве основных элементарных операций обычно рассматриваются следующие: поиск записи с заданным значением ключа, чтение нужной записи, добавление записи, корректировка, удаление. В моделях данных СУБД также предусматриваются специальные операции для установления групповых отношений.

Обобщенные операции или процедуры – последовательность операций, реализующая определенный алгоритм обработки данных. Процедуры могут инициироваться СУБД автоматически, а также могут запускаться пользователем. Примерами процедур являются процедуры копирования БД, восстановления БД, процедуры, вычисляющие значения определенных атрибутов в БД по значениям других атрибутов, и т.п.

Средства контроля используются для реализации ограничений целостности концептуальной модели. Простейшие средства контроля – ограничения – используются для реализации как внешних ограничений концептуальной модели, так и внутренних ограничений модели данных. В качестве последних ограничений, в частности, реализованы ограничения на ввод данных несоответствующего типа, несоответствующей характеристики (по числу битов, по числу полей, по количеству записей и т.п.). Более сложные средства контроля (правила) позволяют вызывать выполнение определенной последовательности операций (сколь угодно сложной) при изменении или добавлении данных в БД и тем самым реализовывать ограничения целостности, описанные с помощью специальных конструкций.

Построение модели данных базы данных (отображение концептуальной модели в модель данных СУБД)

После первой стадии концептуального проектирования у нас сформировано обобщенное представление пользователей о данных, как правило, представленное в виде ER-диаграммы .На следующей стадии (после того, как выбрана определенная СУБД с конкретной моделью данных) необходимо записать концептуальную схему в терминах и понятиях выбранной СУБД. На этой стадии каждая сущность концептуальной модели описывается как запись, состоящая из полей. Каждый атрибут описывается как поле с типом и характеристиками, возможными в выбранной СУБД. Описываются связи концептуальной модели в понятиях, соответствующих выбранной СУБД, определяется порядок реализации запрсов пользователей к базе данных с помощью типовых операций СУБД и т. д.

Результатом этой стадии проектирования будет концептуальная модель, специфицированная к конкретной СУБД.

6.2 Типовые модели данных СУБД и представление концептуальной модели

6.2.1. Сетевая модель данных

Это одна из наиболее ранних моделей данных СУБД. Типовая сетевая модель данных характеристику наиных (Data Base Task Group – DBTG) системного комитета CODASYL (Conference of Data System Languages), основными функциями которого были анализ известных фирменных систем обработки управленческих данных с единых позиций и в единой терминологии, обобщение опыта организации таких систем и разработка рекомендаций по созданию соответствующих систем. Структура данных сетевой модели определяется в терминах раздела 6.1 (элемент, запись, группа, групповое отношение, файл, база данных).

Реализация групповых отношений в сетевой модели осуществляется с использованием специально вводимых дополнительных полей - указателей (адресов связи или ссылок), которые устанавливают связь между владельцем и членом группового отношения. Запись может состоять в отношениях разных типов (1:1, 1:*N*, *M*:*N*). Заметим, что если один из вариантов установления связи 1:1 очевиден (в запись – владелец отношения, поля которой соответствуют атрибутам сущности, включается дополнительное поле – указатель на запись – член отношения), то возможность представления связей 1:*N* и *M*:*N* таким же образом весьма проблематична. Поэтому наиболее распространенным способом организации связей в сетевых СУБД является введение дополнительного типа записей (и соответственно, дополнительного файла), полями которых являются указатели.

Рассмотрим для примера представление группового отношения *M*:*N*. В модель вводится дополнительная группа (дополнительный вид записей). Элементы этой записи представляют собой указатели на две исходные группы и указатели на экземпляры рассматриваемой дополнительной записи, связывающие их в список (цепь), соответствующий *M* и (или) *N* членам группового отношения (рис. 6.1.).

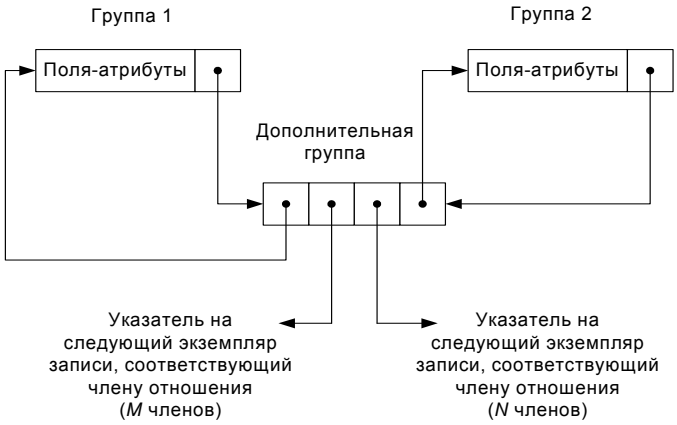


Рис. 6.1. Представление связей типа *M*:*N*

Представление связей 1:1, 1:*M*, *N*:1 является частным случаем связи типа *M*:*N* и осуществляется аналогично рассмотренному выше.

Заметим, что группа может быть членом более чем одного группового отношения. В этом случае вводится несколько дополнительных групп-указателей, а в группе – владельце отношений вводится несколько полей – указателей на дополнительные группы. Тогда множество записей (групп) и связей между ними образует некую сетевую структуру (ориентированный граф общего вида). Вершинами графа являются группы; дугами графа, направленными от владельца к члену группового отношения, – связи между группами.

Сетевая модель данных поддерживает все необходимые операции над данными, реализованные как действия со списковыми структурами. Сетевая модель данных является, вероятно, наиболее общей по возможностям представления концептуальной модели. По сути, любая ER-диаграмма без каких-либо изменений представляется средствами сетевой модели. К недостаткам сетевой модели обычно относят сложность получаемой на её основе концептуальной схемы и большую трудоемкость понимания соответствующей схемы внешним пользователем.

Рассмотрим пример записи части ER-диаграммы (СТУДЕНТ и ФАКУЛЬТЕТ) из предыдущей лекции в терминах сетевой СУБД. Для примера рассмотрим несколько экземпляров сущности СТУДЕНТ и сущности ФАКУЛЬТЕТ (рис. 6.2.).

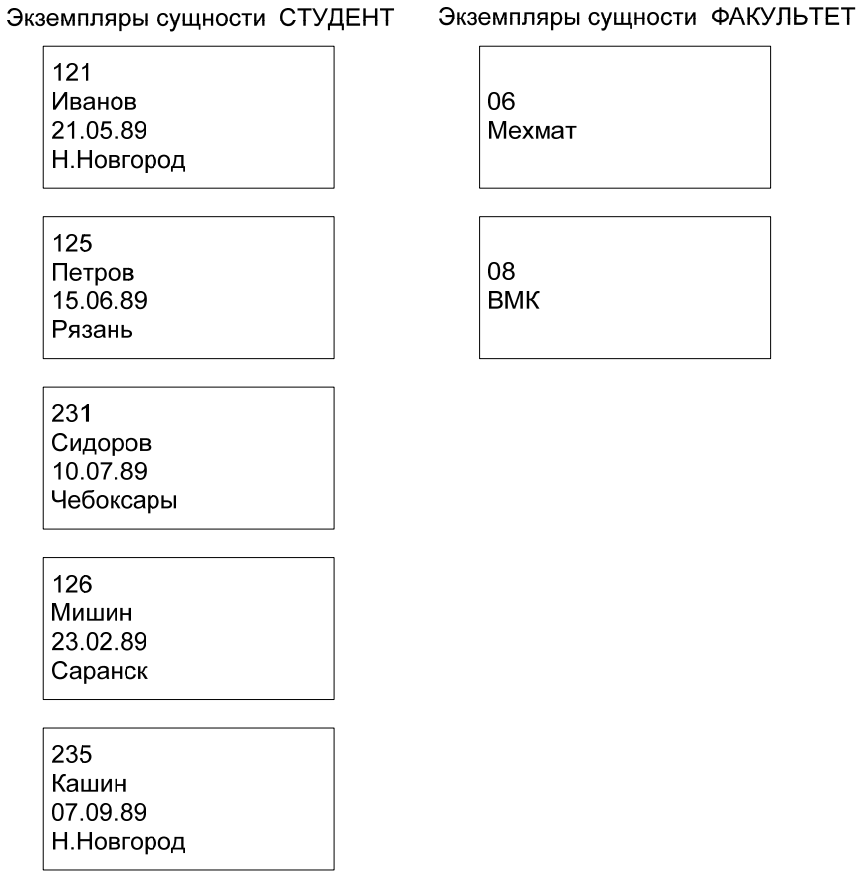


Рис. 6.2. Примеры экземплярв сущностей.

Пусть студенты Иванов, Петров, Мишин учатся на факультете ВМК, Сидоров и Кашин на механико-математическом факультете. Тогда сетевая модель соответствующего фрагмента ER-диаграммы будет выглядеть следующим образом (рис. 6.3).

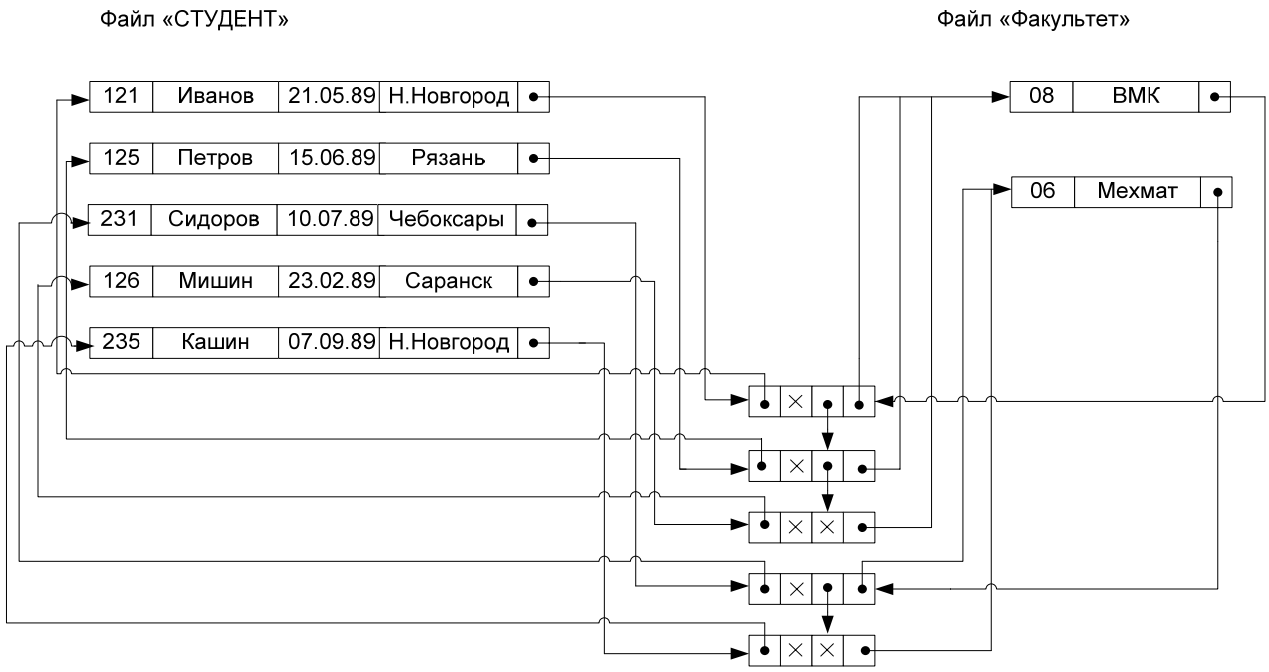


Рис. 6.3. Пример сетевой модели концептуального представления.

Заметим, что в дополнительном файле один из указателей не потребовался, так как рассматриваемая связь имеет тип 1:*N*, а не *M:N*. Значок X обозначает отсутствие дальнейшей связи.

Наиболее существенным недостатком сетевой модели является «жесткость» получаемой концептуальной схемы. Связи закреплены в записях в виде указателей. При появлении новых аспектов использования этих же данных может возникнуть необходимость установления новых связей между ними. Это требует введения в записи новых указателей, т.е. изменения структуры БД, и, соответственно, переформирования всей базы данных.

СУБД, поддерживающие сетевую модель, широко использовались на вычислительных системах серии IBM 360/370 (ЕС ЭВМ). В качестве примеров таких систем можно указать IDMS, UNIBAD (БАНК), и их аналоги СЕДАН, СЕТОР. На персональных компьютерах сетевые СУБД не получили широкого распространения. Примером сетевой СУБД для персонального компьютера является db_VISTA III. Отметим, что система db_VISTA реализована на языке С и поэтому является переносимой. Система может эксплуатироваться на ПЭВМ типа IBM PC, SUN, Macintosh.

6.2.2. Иерархическая модель данных

Это также одна из наиболее ранних моделей данных. Реализация групповых отношений в иерархической модели, как и в сетевой, может осуществляться с помощью указателей и представляется в виде графа. Однако, в отличие от сетевой модели, здесь существует ряд принципиальных особенностей.

1. Групповые отношения являются отношениями соподчиненности. Группа (запись) – владелец отношения имеет подчиненные группы – члены отношений. Исходная группа называется предком, подчиненная – потомком.
2. Групповые отношения образуют иерархическую структуру, которую можно описать как ориентированный граф следующего вида:
   - имеется единственная особая вершина (соответствующая группе), называемая корнем, в которую не заходит ни одно ребро (группа не имеет предков);
   - во все остальные вершины входит только одно ребро (все остальные группы имеют одного предка), а исходит произвольное количество ребер (группы имеют произвольное количество потомков);
   - отсутствуют циклы.
3. Иерархическая модель данных может представлять совокупность нескольких деревьев. В терминологии иерархической модели деревья, описывающие структуру данных, называются деревьями описания данных, а сами структурированные данные (база данных) – деревьями данных.

Особенностью реализации операций поиска в иерархической модели является то, что операция всегда начинает поиск с корневой вершины и специфицирует иерархический путь (последовательность связанных вершин) от корня до вершины, экземпляры которой удовлетворяют условиям поиска.

Необходимо отметить, что программы, реализующие операции иерархической модели, существенно проще, чем аналогичные программы для сетевой модели, т.к. здесь много легче осуществлять навигацию по структуре. Целесообразность появления иерархической модели обусловлена, конечно, тем, что большинство организационных систем реального мира имеют иерархическую структуру (административное деление страны, организационная структура предприятия и т.п.). Соответствующее концептуальное представление также будет иметь иерархическую структуру и естественным образом может быть описано в терминах иерархической модели. В качестве недостатков иерархической модели можно назвать вышеуказанные недостатки сетевой.

СУБД, поддерживающие иерархическую модель, достаточно широко использовались на вычислительных системах IBM 360/370 (ЕС ЭВМ). В качестве примеров таких систем можно указать IMS, ОКА и широко тиражируемую в СССР отечественную разработку ИНЕС. Примером иерархической СУБД для персональных ЭВМ является отечественная система НИКА (адаптация системы ИНЕС к IBM PC).

6.2.3. Реляционная модель данных

Учитывая отмеченные в предыдущих разделах недостатки сетевых и иерархических моделей, можно сформулировать желательные требования к модели данных:

- модель должна быть понятна пользователю, не имеющему особых навыков в программировании;
- появление новых аспектов использования данных и необходимость введения новых связей не должны приводить к реструктуризации всей модели данных и базы данных в целом.

Моделью данных, удовлетворяющей вышеуказанным требованиям, является **реляционная модель**, часто называемая также **табличной**.

Основным используемым понятиям здесь является понятие отношения, представляемого в виде таблицы, столбцы которой соответствуют атрибутам сущности (структура строки таблицы аналогична структуре записи). Каждый атрибут может принимать определенное множество значений, называемое доменом. Строка таблицы с конкретными значениями полей здесь называется кортежем (соответствует понятию «экземпляр записи»). Поля таблицы предполагаются элементарными (неделимыми). Таким образом, понятие «таблица» здесь соответствует понятию «файл» модели данных. Первичный ключ здесь –минимальный набор атрибутов, однозначно идентифицирующий кортеж в отношении.

Групповое отношение может представляться двумя способами. При первом способе в таблицы, соответствующие группам – членам отношения, добавляются столбцы ключевых полей (атрибутов) другого члена отношения (связь описывается через ключевые атрибуты). При втором способе групповое отношение определяется как дополнительная группа (дополнительная таблица). Столбцами этой дополнительной таблицы являются ключи групп – членов отношения. Таким образом, при любом способе соответствующая модель данных представляет собой совокупность структур таблиц.

Рассмотрим пример записи ER-диаграммы (см. рис. 5.2.) в терминах реляционных баз данных.

Сначала представим таблицы, соответствующие сущностям.

Таблица СТУДЕНТ

Код	Фамилия	Дата рождения	Место рождения

Таблица ФАКУЛЬТЕТ

Номер	Название

Таблица СПЕЦИАЛЬНОСТЬ

Номер	Название

Представим таблицы, описывающие связи.

Таблица «Студент учится на факультете»

Код студента	Номер факультета

Таблица «Студент учится по специальности»

Код студента	Номер специальности

Таблица «На факультете имеются специальности»

Номер факультета	Номер специальности

Заметим, что здесь реализован вышеописанный второй способ представления групповых отношений. Очевидно, что можно построить много вариантов таблиц, представляющих соответствующую ER-диаграмму.

Для приведенных таблиц не указаны домены атрибутов. Отсутствие указания доменов приводит к неоднозначной интерпретации содержания таблицы. Например, две таблицы с одинаковыми атрибутами

Код студента	Фамилия	Дата рождения	Место рождения

могут иметь разное смысловое значение и, соответственно, разное содержание (в одной таблице содержатся сведения о всх студентах факультета, в другой таблице сведения только о старостах факультета). Для устранения неоднозначной интерпретации в случае отсутствия указания доменов используют имя таблицы (СТУДЕНТ; СТАРОСТА)

Для формального описания таблицы используется теоретико-множественное понятие отношения.

Схемой отношения R называется перечень имен атрибутов отношения (соответствующих столбцам таблицы) с указанием доменов этих атрибутов и обозначается $R(A_1, A_2, \ldots, A_n)$; $\{A_i\} \subseteq D_i$, где $\{A_i\}$ – множество значений, принимаемых атрибутом A_i ($i = \overline{1,n}$).

Совокупность схем отношений, используемых для представления концептуальной модели, называется схемой реляционной базы данных, а текущие значения соответствующих отношений – реляционной базой данных.

В качестве основного недостатка реляционной модели можно указать дублирование информации при представлении связей.

Необходимо отметить, что большинство СУБД для персональных ЭВМ поддерживают именно реляционную модель данных. В качестве примеров таких наиболее распространенных СУБД можно указать все dBase-подобные системы, DB2, Paradox, Access, FoxPro, Oracle, MS SQL Server.

Более подробно реляционная модель данных будет рассмотрена в следующей лекции.

6.2.4. Многомерная модель данных

Вернемся к понятию «сущность» концептуальной модели.

Сущность – это то, о чем накапливается информация в информационной системе. Часто оказывается, что информация об определенной сущности зависит еще от ряда параметров. Рассмотрим, например, сущность УСПЕВАЕМОСТЬ СТУДЕНТОВ со следующими атрибутами: число двоек, число троек, число четверок, число пятерок.

Значение атрибутов зависит от параметров «курс», «учебный год». Если использовать для описания соответствующей концептуальной схемы реляционную модель, то необходимо вводить множество таблиц УСПЕВАЕМОСТЬ СТУДЕНТОВ по каждому году для каждого курса. Так, при 5 курсах и необходимости анализировать данные за 10 лет число таблиц будет равно пятидесяти. Дублируются аналогичные структуры всех таблиц, достаточно сложна обработка данных, связанная с анализом однотипных данных при изменении значения одного из параметров и т.д.

Наиболее подходящей моделью данных для этого случая является так называемая многомерная модель, используемая в технологии OLAP (OnLine Analytical Processing – оперативная аналитическая обработка). Отметим, что многомерность модели данных означает здесь многомерное логическое представление структуры информации и, вообще говоря, не связана с многомерностью визуализации.

Многомерные структуры представляются как гиперкубы данных. Каждая грань куба является размерностью. Основными понятиями, используемыми в многомерных моделях данных, являются «измерение» (dimension) и «ячейка» (cell).

Измерение – упорядоченный набор значений, принимаемых конкретным параметром, соответствующий одной из граней гиперкуба. Для нашего примера можно указать в качестве измерений: учебный год – 2006-2007, 2007-2008, 2008-2009; курсы – 1,2,3 и т.д.

Ячейка или показатель – это поле, соответствующее атрибуту сущности, значение которого однозначно определяется фиксированным набором значений параметров (значениями «измерений», например, 2008-2009 учебный год, первый курс).

В многомерной модели данных определяется ряд дополнительных операций, среди которых можно выделить операции «формирование среза» и «агрегация».

При формировании среза пользователю по его запросу предоставляется некоторое подмножество гиперкуба, полученное в результате фиксаций пользователем одного или нескольких значений параметров. Операция «агрегация» обеспечивает переход к более общему представлению информации из гиперкуба пользователю, например суммируя значения показателей по всем значениям одного из параметров, допустим, по всем курсам.

Такая модель позволяет легко сравнивать данные при разных значениях параметров, строить графики зависимости значений конкретных атрибутов от значений определенных параметров (например, изменение атрибута по годам) и т.п. Поэтому основное назначение технологии OLAP – обработка информации для проведения анализа и принятия решения.

Массовое использование СУБД, поддерживающих многомерную модель данных, только начинается. В качестве наиболее известных СУБД такого типа можно указать Oracle Express Server.

6.3. Средства автоматизированного проектирования концептуальной модели

Средства автоматизированного проектирования концептуальной модели привлекают к себе в настоящее время большой интерес и используются в процессе создания структуры базы данных и интерфейса пользователя для доступа к данным.

Причина применения этих средств состоит в использовании в подавляющем большинстве реальных разработок баз данных спиральной модели жизненного цикла программного обеспечения, что предусматривает последовательное создание нескольких версий программного обеспечения. Каждая следующая версия включает в себя предыдущую (возможно, не полностью) и является ответом на замечания пользователя, полученные в результате тестирования предыдущей версии. При создании баз данных первая модель программного обеспечения, к сожалению, очень редко является удачной. Чаще всего заказчик отвергает первую версию, так как она недостаточно полно отвечает его требованиям. Причина такой ситуации заключается в том, что заказчик не может сразу, до создания начальной версии программы, четко и полно сформулировать свои требования. Обычно после получения первого варианта программного обеспечения заказчик выдвигает дополнительные требования, которые нельзя реализовать в рамках созданной базы данных. Это вынуждает разработчиков вносить изменения в структуру базы данных, а также, соответственно, в интерфейс пользователя для доступа к базе данных. Таких итераций может быть несколько до момента получения решения, адекватного запросам заказчика. Но даже после получения удовлетворительного решения процесс разработки базы данных не завершается. Жизнь не стоит на месте, и запросы заказчика меняются с течением времени. Часть этих изменений можно реализовать без изменения структуры базы данных, изменяя только интерфейс пользователя, другие же требуют изменения и интерфейса, и структуры базы данных. Надо заметить, что подобные изменения являются очень болезненными – работа по их внесению может оказаться трудоемкой и, что самое неприятное, потребовать замены большого количества отлаженного программного кода. Иными словами, замененный код был написан впустую, на самом деле его не нужно было писать.

Таким образом, создание работоспособной базы данных можно условно разделить на три этапа – проектирование базы данных, в процессе которого создаются рабочие прототипы, кодирование – создание структур баз данных и законченного интерфейса пользователя и сопровождение готовой базы данных.

Основная идея применения средств автоматизированного проектирования баз данных заключается в том, что процесс ручного кодирования начинается только после окончания процесса проектирования. На стадии проектирования схема базы данных и интерфейс пользователя для доступа к базе данных создаются автоматически, исходя из описания концептуальной модели, с помощью так называемых CASE-средств (Computer Aided Software/System Engineering). Конечно, созданный таким образом интерфейс не является законченным программным продуктом, однако он позволяет заказчику оценить возможности конечного продукта и внести свои коррективы. Только после одобрения заказчиком рабочего прототипа разработчики приступают к ручному кодированию – созданию законченного приложения.

При сопровождении все повторяется, за тем исключением, что генерируется не все приложение целиком, а только часть, которую надо изменять.

На практике чаще всего CASE-средства используются для создания схемы базы данных в виде ER-диаграмм и генерации структур баз данных для конкретной СУБД. После получения от заказчика изменений разработчики вносят соответствующие исправления в диаграмму «сущность – связь» и заново генерируют структуры баз данных. Средства автоматической генерации интерфейсов используются реже.

В настоящее время практически каждый производитель СУБД предлагает собственный программный продукт автоматизированного проектирования. Это Oracle Designer (Oracle), Power Desinger (Sybase) и другие. Демонстрационные версии данных программных продуктов можно загрузить с соответствующих сайтов (www.oracle.com, www.sybase.com).

Кроме того, на рынке представлены решения третьих фирм, не производящих СУБД. Одними из самых распространенных являются программные продукты фирмы AllFusion – AllFusion ERwin Data Modeler и AllFusion Process Modeler (ранее – BPwin) и другие. На российском рынке данные программы предлагает фирма Interface Ltd. (www.interface.ru). Создание диаграммы «сущность – связь» осуществляется с помощью AllFusion ERwin Data Modeler, дальнейшее моделирование, включая генерациюпрограммного кода создания базы данных производится с помощью программы AllFusion Process Modeler.

Создав наглядную модель базы данных можно оптимизировать структуру БД и добиться её полного соответствия требованиям и задачам организации. Визуальное моделирование повышает качество создаваемой базы данных, продуктивность и скорость её разработки.

На сайте Interface Ltd. доступна для загрузки демонстрационная версия AllFusion ERwin Data Modeler, которая представляет собой полнофункциональную версию, ограниченную по времени.

Краткие итоги. В лекции рассмотрены вопросы, тносящиеся ко второй стадии концептуального проектирования – представлениию концептуальной модели в терминах модели данных определенной СУБД. Дано описание общего представления о модели данных (основные используемые понятия - элемент, запись, файл; основные составляющие описания). Рассмотрены модели данных СУБД как инструмент представления концептуальной модели (сетевая модель данных, иерархическая модель данных, реляционная модель данных, мно-

гомерная модель данных и OLAP-технология). Приведены примеры записи концептуальной модели в терминах конкретной модели данных СУБД. Приводятся сведения о средствах автоматизированного проектирования концептуальной модели.

Вопросы данной лекции рассматриваются в [1-8].

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Общие представления о модели данных?

Вариант 1.
Что такое модель данных СУБД?

⊞ способ структурирования данных в СУБД
⊞ виды и типы данных, поддерживаемые СУБД
⊞ инструмент представления концептуальной модели в конкретной СУБД
☐ концептуальная модель, специфицированная к конкретной СУБД

Вариант 2.
Какие понятия используются при описании данных в терминах модели данных?

☐ атрибут
☐ сущность
⊞ поле
⊞ запись
☐ строка
⊞ экземпляр записи
⊞ файл

Вариант 3. .
Какие формы используются для представления группового отношения?

⊞ графовая
⊞ табличная
☐ строковая
☐ столбцовая
⊞ реляционная

Задача 2. Что входит в описание модели данных СУБД?

Вариант 1.
Как описываются структуры данных в модели данных СУБД?

⊞ представляются конкретные типы данных и их характеристики
☐ предлагается описать любые типы данных и их характеристики
⊞ определены способы составления структур более общего вида из структур простых видов
☐ предлагается описать способы составления структур более общего вида из структур простых видов
⊞ представляются конкретные средства реализации связей
☐ предлагается описать необходимые способы реализации связей

Вариант 2.
Какие возможные действия входят в описание модели данных СУБД?

⊞ элементарные операции над данными
⊞ обобщенные операции над данными (процедуры)
⊞ средства контроля ограничений целостности
☐ операции по анализу данных
☐ действия, реализуемые прикладными программами
☐ типы и характеристики структур данных
☐ операции над данными

Задача 3.Как концептуальная модель специфицируется в терминах модели данных СУБД?

Вариант 1.
Как представляются сущности ER-диаграммы при отображении обобщенного представления средствами модели данных СУБД?

☐ записями
☐ атрибутами
☐ файлами
⊞ таблицами

Вариант 2.
Как представляются атрибуты ER-диаграммы при отображении обобщенного представления средствами модели данных СУБД?

⊞ полями с указанием выбранного типа данных СУБД и характеристики данных
☐ полями с указанием задаваемыми пользователем типом данных и характеристики данных
☐ экземплярами записей
☐ конкретными значениями

Вариант 3.
Как представляются связи, изображенные на ER-диаграмме при отображении обобщенного представления средствами модели данных СУБД?

☐ с помощью стрелок
☐ с помощью указателей
⊞ с помощью понятий, описанных в выбранной СУБД
☐ с помощью терминов, определенных пользователем
☐ с помощью понятий ER-диаграммы

Задача 4. Что такое сетевая модель данных?

Вариант 1.
Как представляется сущность в сетевой модели?

☐ записью
☐ графом
☐ строкой таблицы
⊞ вершиной графа

Вариант 2.
Как представляется групповое отношение (связь) в сетевой модели?

⊞ указателем
⊞ дугой
☐ дополнительным файлом
☐ записью

Вариант 3.
Основные особенности сетевой модели:

☐ простота алгоритмов поиска
☐ поиск начинается с корневой вершины
⊞ удобство представления любой концептуальной модели
☐ добавление новых сущностей и связей не требует изменения всей структуры базы данных
⊞ высокая трудоемкость программирования

Задача 5. Что такое иерархическая модель данных?

Вариант 1.
Как представляется сущность в иерархической модели?

☐ записью
☐ деревом
☐ строкой таблицы
⊞ вершиной графа

Вариант 2.
Как представляется групповое отношение (связь) в иерархической модели?

⊞ указателем
⊞ ребром
☐ записью
☐ деревом
☐ вершиной графа

Вариант 3.
Основные особенности иерархической модели.

⊞ простота алгоритмов поиска
⊞ поиск начинается с корневой вершины
☐ удобство представления любой концептуальной модели
☐ добавление новых сущностей и связей не требует изменения всей структуры базы данных

Задача 6. Что такое реляционная модель данных?

Вариант 1.
Как представляется сущность в реляционной модели?

☐ строкой таблицы
☐ столбцом таблицы
⊞ таблицей
☐ набором таблиц

Вариант 2.
Как представляется групповое отношение (связь) в реляционной модели?

☐ строками таблицы
⊞ столбцами таблицы
⊞ таблицей
☐ набором таблиц

Вариант 3.
Каковы основные достоинства реляционной модели?

⊞ понятна для пользователя
⊞ добавление новых сущностей и связей не требует изменения всей структуры базы данных
⊞ поддерживается многими СУБД
☐ не требует навыков работы с компьютером

Вариант 4.
Как выглядит концептуальная схема реляционной базы данных?

⊞ набор заголовков таблиц с именами атрибутов
☐ набор таблиц с данными
⊞ совокупность схем отношений
☐ текущие значения соответствующих отношений

Задача 7.Что такое многомерная модель данных?

Вариант 1.
Когда целесообразно использовать многомерную модель данных?

☐ большое количество таблиц с данными
☐ большой объем данных в таблицах
⊞ большое количество таблиц одной структуры при разных значениях параметров.
☐ большое количество атрибутов в таблице

Вариант 2.
Для каких основных целей используется многомерная модель?

☐ для быстрого поиска информации
⊞ для сравнительного анализа
⊞ для оперативной аналитической обработки
☐ в технологии olap

Вариант 3.
Какие понятия характеризуют многомерный куб?

⊞ измерение
⊞ ячейка
⊞ поле
⊞ показатель
⊞ размерность

Вариант 4.
Какие дополнительные операции определены в многомерной модели?

☐ анализ данных
⊞ срез данных
⊞ агрегация данных
☐ визуализация данных

Задача 8. Что такое автоматизированное проектирование баз данных?

Вариант 1.
Какие основные причины использования программных систем автоматизированного проектирования?

⊞ необходимость создания нескольких последовательных версий базы данных
⊞ целесообразность создания на первом этапе пробной версии базы данных
⊞ необходимость сокращения затрат на начальное проектирование базы данных
☐ необходимость сокращения затрат на окончательную доработку базы данных

Вариант 2.
Какие этапы создания базы данных поддерживаются средствами автоматизированного проектирования?

⊞ разработка ER-диаграммы
⊞ разработка программ создания структуры базы данных
⊞ разработка интерфейса пользователя
☐ разработка прикладных программ

Вариант 3.
При использовании каких СУБД для создания базы данных можно пользоваться средствами автоматизированного проектирования?

☐ любых
⊞ тех, для которых предназначено соответствующее средство
⊞ для каких-то этапов создания базы данных – любых, для других этапов – только тех, для которых предназначено соответствующее средство
☐ СУБД и средства автоматизированного проектирования обязательно должны быть разработаны одной фирмой-производителем

Литература

1. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Щерса. – М.: Мир, 1980. – 664 с.
2. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для вузов. – СПб.: КОРОНА принт, 2000. – 416 с.
3. Горев А., Ахаян Р., Макашарипов С. Эффективная работа с СУБД. СПб.: Питер, 1997. – 700 с.
4. Карпова Т. Базы данных. Модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
5. Саймон А.Р. Стратегические технологии баз данных: менеджмент на 2000 год: Пер. с англ. / Под ред. и с предисл. М.Р. Когаловского. – М.: Финансы и статистика, 1999. – 479 с.
6. Ульман Дж. Д., Уидом Дж. Введение в системы баз данных: Пер. с англ. – М.: Лори, 2000. – 374 с.
7. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Н.Новгород: Изд-во ННГУ, 2004. 271 с.
8. Хаббард Дж. Автоматизированное проектирование баз данных: Пер. с англ. под ред. А.Л. Щерса. – М.: Мир, 1984. – 296 с..

Лекция 7. Формализация реляционной модели

В лекции рассматриваются вопросы, связанные с формализацией наиболее распространенной в настоящее время модели данных СУБД – реляционной модели. Здесь рассматривается формализованное описание отношений и средств манипулирования данными в реляционной модели.

Ключевые термины: атрибут, домен, схема отношения, отношение, ключ отношения, реляционное исчисление, реляционная алгебра, операции реляционной алгебры, объединение, разность, декартово произведение, проекция, селекция, пересечение, θ-соединение, естественное соединение.

Цель лекции: рассмотреть формализованное описание реляционной модели и операций манипулирования данными как основу для использования математических методов проектирования баз данных и основу создания языков запрсов к базе данных.

7.1. Формализованное описание отношений и схемы отношений

Как уже отмечалось в п. 6.2.3, реляционная модель описывает представление данных в виде двумерной таблицы, называемой отношением. Наименованиями столбцов этой таблицы **служат имена атрибутов. Рассмотрим формализованное описание соответствующих понятий.**

Пусть$A_1, A_2,\ldots, A_n$ имена атрибутов. Каждому имени атрибута A_i соответствует допустимое множество значений, которые может принимать атрибут A_i. Это множество значений D_i называется доменом атрибута A_i, $i = \overline{1,n}$***. По определению, домены являются непустыми конечными или счетными множествами. Уточним, что в теории реляционных баз данных домен рассматривается как множество значений одного (причем простого) типа данных. Понятию домена D_i соответствует множество значений, стоящих в столбце A_i рассматриваемой таблицы.***

Схемой отношения R $\{A_1, A_2, \ldots, A_n\}$ называеся конечное множество имен атрибутов $\{A_1, A_2, \ldots, A_n\}$, причем атрибут A_i принимает значение из множества D_i (i=1, 2,…n), где n – арность отношения.

Понятию «схема отношения» соответствует описание структуры двумерной таблицы (имена столбцов и допустимые множества значений).

Пусть $D = D_1 \cup D_2 \cup \ldots \cup D_n$.

Отношением r со схемой R называется конечное множество отображений $\{t_1, t_2,\ldots, t_p\}$ из множества R: $\{A_1, A_2, \ldots, A_n\}$ в множество D:$\{ D_1 \cup D_2 \cup \ldots \cup D_n \}$, таких, что $t_k(A_i) \in D_i$, $k = \overline{1,p}$ ***;*** $i = \overline{1,n}$.

Отображение t_k называется k-м кортежем, n – размерность кортежа.

Понятию *k*-го кортежа соответствует множество значений, стоящих в *k*-й строке рассматриваемой таблицы.

Понятию отношения *r* соответствует множество значений, стоящих во всех строках рассматриваемой таблицы.

Ключом отношения r со схемой R называется минимальное подмножество $K = \{A_{i1}, A_{i2}, \ldots, A_{im}\} \subseteq \{A_1, A_2, \ldots, A_n\}$, ***где*** $\{i1, i2, \ldots, im\} \subseteq \{1, 2, \ldots, n\}$, ***такое, что любые два различных кортежа*** $t_1, t_2 \in r$ $(t_1 \neq t_2)$ ***не совпадают по значениям множества*** $K = \{A_{i1}, A_{i2}, \ldots, A_{im}\}$.

Возможны случаи, когда отношение *r* имеет несколько ключей. Такие ключи называются потенциальными (возможными). ***Выбранный из них ключ для идентификации кортежей называется первичным ключом***. Таким образом, достаточно знать значение кортежа на множестве *K*, чтобы однозначно его идентифицировать. Ключ используется для представления связей между отношениями. С этой целью первичный ключ одного отношения включается в структуру (набор атрибутов) связанного с ним отношения. Для второго отношения соответствующий ключ называется внешним ключом.

Совокупность схем отношений, используемых для представления концептуальной модели, называется схемой реляционной базы данных (реляционной моделью данных). Текущие значения соответствующих отношений называются реляционной базой данных.

Выпишем реляционную модель данных примера из предыдущей лекции (см. рис. 6.3.). Введем обозначения атрибутов всех соответствующих сущностей. Пусть A_1 – код студента, A_2 – фамилия, A_3 – дата рождения, A_4 – место рождения, A_5 – номер факультета, A_6 – название факультета, A_7 – номер специальности, A_8 – название специальности. Обозначим схему отношения СТУДЕНТ как *R*1, ФАКУЛЬТЕТ как *R*2, СПЕЦИАЛЬНОСТЬ как *R*3, СТУДЕНТ УЧИТСЯ НА ФАКУЛЬТЕТЕ как *R*4, СТУДЕНТ УЧИТСЯ ПО СПЕЦИАЛЬНОСТИ как *R*5, НА ФАКУЛЬТЕТЕ ИМЕЮТСЯ СПЕЦИАЛЬНОСТИ как *R*6.

Тогда реляционная модель соответствующего примера описывается следующей совокупностью схем отношений:

$R1(A_1, A_2, A_3, A_4)$

$R2(A_5, A_6)$

$R3(A_7, A_8)$

$R4(A_1, A_5)$

$R5(A_1, A_7)$

$R6(A_5, A_7)$

Напомним, что понятие «схема отношения» соответствует описанию структуры таблицы. Таблица с заполненными значениями (заполненными строками) соответствует понятие «отношение». Для данного примера отношения, соответствующие вышеуказанным схемам отношений будем обозначать

*r*1, *r*2, *r*3, *r*4, *r*5, *r*6,

Отметим следующие свойства отношения:

1. Отношение имеет имя, которое отличается от имен всех других отношений.

2. Каждое значение элементов кортежей представляется простым (атомарным) типом данных.
3. Каждый атрибут имеет уникальное имя.
4. Значения всех атрибутов являются атомарными (неделимыми). Это следует из определения домена как множества значений простого типа данных, т.е. среди значений домена не могут содержаться множества.
5. Порядок рассмотрения атрибутов в схеме отношения (отношении) не имеет значения, т.к. для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута.
6. Порядок рассмотрения кортежей в отношении не имеет значения, т.к. отношение представляет собой множество кортежей, а элементы множества, по определению теории множеств, неупорядочены.

7.2. Манипулирование данными в реляционной модели

Для манипулирования данными в реляционной модели используются два формальных аппарата:

- реляционная алгебра, основанная на теории множеств;
- реляционное исчисление, базирующееся на исчислении предикатов первого порядка.

Механизмы реляционной алгебры и реляционного исчисления эквивалентны, т.е. для любого допустимого выражения реляционной алгебры можно построить эквивалентную формулу реляционного исчисления и наоборот

Отличаются два этих формальных аппарата уровнем процедурности. Выражения реляционной алгебры строятся на основе алгебраических операций (высокого уровня), и подобно тому, как интерпретируются арифметические и логические выражения, выражение реляционной алгебры также имеет процедурную интерпретацию. Другими словами, запрос, представленный на языке реляционной алгебры, может быть реализован как последовательность элементарных алгебраических операций с учетом их старшинства и возможного наличия скобок.

Для формулы реляционного исчисления однозначная интерпретация (соответствующая однозначная последовательность действий), вообще говоря, отсутствует. Формула только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения. Поэтому языки реляционного исчисления являются более непроцедурными или декларативными.

Операции, реализуемые с помощью указанных аппаратов, обладают важным свойством: они замкнуты на множестве отношений. Это означает, что выражения реляционной алгебры и формулы реляционного исчисления определяются над отношениями реляционных БД и результатом вычисления также являются отношения. В результате любое выражение или формула могут интерпретироваться как отношение, что позволяет использовать их в других выражениях или формулах.

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

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

7.3. Операции реляционной алгебры

Операции реляционной алгебры определены на множестве отношений и являются замкнутыми относительно этого множества (образуют алгебру). Оказывается, что любой произвольный запрос к БД можно представить в виде последовательности, составленной из пяти основных операций реляционной алгебры. Рассмотрим эти операции.

Объединение $r \cup s$

Объединением отношений r и s называется множество кортежей, которые принадлежат или r, или s, или им обоим. Для операции объединения требуется одинаковая арность отношений.

Для примера, пусть

r

a	b	a
d	a	f
c	b	d

s

b	g	a
d	a	f

тогда

$r \cup s$

a	b	a
d	a	f
c	b	d
b	g	a

Заметим, что с помощью операции объединения может быть реализовано добавление нового кортежа к имеющемуся отношению. В этом случае r – исходное отношение, s – отношение, содержащее один добавляемый кортеж.

Разность $r - s$

Разностью отношений r и s называется множество кортежей, принадлежащих r, но не принадлежащих s. Для этой операции также требуется одинаковая арность отношений.

$r - s$

a	b	a
c	b	d

Заметим, что с помощью операции разности может быть реализовано удаление кортежа из имеющегося отношения. В этом случае *r* – исходное отношение, *s* – отношение, содержащее один удаляемый кортеж.

Декартово произведение *r* ×*s*

Пусть r и s – отношения арности k1 и k2 соответственно. Декартовым произведением r×s называется множество кортежей длины k1+k2, первые k1 компонентов которых образуют кортежи, принадлежащие r, а последние k2 – кортежи, принадлежащие s.

$r \times s$

a	b	a	b	g	a
a	b	a	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

Проекция $\pi_{A_{i1}, A_{i2}, \ldots, A_{im}}(r)$

Проекция $\pi_{A_{i1}, A_{i2}, \ldots, A_{im}}(r)$ ***есть множество кортежей, получаемых из кортежей отношения r выбором столбцов с именами A_{i1}, A_{i2}, …, A_{im}.***

Другими словами, это операция построения «вертикального» подмножества, получаемого путем выбора определенных атрибутов и исключения остальных. Повторяющиеся кортежи исключаются.

$\pi_{1,3}(r)$

a	a
d	f
c	d

Выбор (селекция) $\sigma_F(r)$

Пусть F – формула, образованная: операндами, являющимися константами или именами атрибутов, арифметическими операторами сравнения, логическими операторами (и, или, не), тогда выбором (селекцией) σ_F ***называется множество кортежей, компоненты которого удовлетворяют условию, заданному формулой F.***

$\sigma_{(1)=(3)}(r) =$

a	b	a

Здесь F:(1)=(3) – содержимое первого столбца равно содержимому третьего столбца.

Приведем ряд примеров представления запросов с помощью формальных операций для реляционной модели (СТУДЕНТ, ФАКУЛЬТЕТ, СПЕЦИАЛЬНОСТЬ), рассмотренной выше.

Пример 1.

Сформировать список студентов (фамилия).

Рассмотрим схему отношения СТУДЕНТ.

Атрибут «Фамилия» обозначен здесь A_1 Для ответа на запрос необходимо взять проекцию отношения $r1$ на столбец A_1.

.
$$\pi_{A_1}(r1)$$

Пример 2.

Выдать список фамилий и дат рождений студентов, которым на текущую дату (*date*) больше 35 лет.

Рассмотрим то же отношение $r1$. Сначала выбираем студентов, которым больше 35 лет:

$$\sigma_{(A_3)+35<date}(r1)\,.$$

Затем берем проекцию полученного отношения на столбцы

$$\pi_{A_1,A_3}(\sigma_{(A_3)+35<date}(r1))\,.$$

Заметим, что можно было бы выполнить эти две операции в другой последовательности – сначала проекция, а затем селекция. Предлагается оценить, какой из этих вариантов лучше по оценке числа выполняемых элементарных действий и объему требуемой памяти.

Пример 3.

Выдать список фамилий студентов, обучающихся по специальности «Информационные технологии». Название специальности является атрибутом отношения $r3$. Если бы в этом отношении присутствовал атрибут «фамилия», то задача решалась бы аналогично примеру 2. В отношении $r3$ присутствует атрибут «код студента», а «фамилия» присутствует в отношении $r1$. Для ответа на этот запрос необходимо связывать по «код студента» отношение $r3$ и отношение $r1$.

Сначала выберем из отношения $r3$ кортежи с названием специальности «Информационные технологии». Обозначим полученное отношение $rp1$. (Дальнейшие промежуточные отношения будем обозначать последовательно $rp1$, $rp2$, $rp3$ и т.д.).

$$rp1 = \sigma_{(A_8)="Информационныетехнологии"}(r3)\,.$$

Далее нас будет интересовать только атрибут A_1 – «код студента». Поэтому возьмем проекцию на эти столбцы.

$$rp2 = \pi_{A_1}(rp1)\,.$$

Далее необходимо связать отношения $r1$ и $rp2$ (склеить таблицы). Для склейки таблиц используется операция «декартово произведение»:

$$rp3 = r1 \times rp2\,.$$

В отношении $r3$ присутствуют два одинаковых столбца: A_1 из отношения $r1$ и A_1 из отношения $rp2$. Выбирая из отношения $rp3$ строки, в которых значения в соответствующих столбцах совпадают, получим сведения о студентов, обучающихся по специальности «Информационные технологии»

$$rp4 = \sigma_{(A_1 \cdot r1)=(A_1 \cdot rp2)}(rp3),$$

где $A_1 \cdot r1$ и $A_1 \cdot rp2$ обозначают соответственно столбец A_1 соответствующей первой и второй составной части декартова произведения. Теперь осталось только выбрать фамилии соответствующих студентов

$$rp5 = \pi_{A_2}(rp4).$$

Получаем требуемый результат. Заметим, что для экономии действий и памяти, перед тем как склеивать таблицы, целесообразно было сделать операцию проекции отношения $r1$ на столбцы A_1, A_2. (чтобы не включать в декартово произведение лишние столбцы).

Введенные пять основных операций реляционной алгебры позволяют реализовать любой запрос к реляционной базе данных. Однако наряду с основными операциями достаточно часто удобно использовать так называемые дополнительные операции реляционной алгебры (которые могут быть выражены через основные).

Пересечение $r \cap s$

Пересечением отношений r и s называется множество кортежей, принадлежащих как r, так и s. Пересечение может быть выражено через операции разности

$$r \cap s = r - (r - s).$$

θ-соединение $r \underset{i\theta j}{\triangleright\triangleleft} s$

θ-соединение r и s по столбцам A_i и A_j представляет собой множество таких кортежей в декартовом произведении r и s, что i-й компонент r находится в отношении θ с j-м компонентом s, где θ – арифметический оператор сравнения. Если θ является оператором равенства, то эта операция называется эквисоединением

$$r \underset{i\theta j}{\triangleright\triangleleft} s = \sigma_{i\theta(l+j)}(r \times s),$$

где l – арность отношения r.

Пример.

r

1	2	3
4	5	6
7	8	9

s

3	1
6	2

$r \underset{(2)<(1)}{\triangleright\triangleleft} s$

1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

Заметим, что в примере 3 две последовательно идущие операции (декартово произведение и селекция) вместе как раз представляют операцию соединения. Причем использование декартова произведения для соединения таблиц обязательно обусловливает использование селекции как следующей операции для установления связи между таблицами. Поэтому

целесообразно использовать такую объединенную операцию и программно реализовывать в СУБД именно операцию соединения.

Естественное соединение $r \triangleright\triangleleft s$

Операция применима тогда и только тогда, когда столбцы имеют имена (являются атрибутами). Операция применима к отношениям, у которых есть одинаковые атрибуты.

Пусть

$$r = (A_1, ..., A_k, B_1,..., B_n),\ s = (A_1, ..., A_k, C_1,..., C_m),$$

имена $A_1, ..., A_k$ ***совпадают.***

Тогда $r \triangleright\triangleleft s$ ***определяется следующим образом***

$$r \triangleright\triangleleft s = \pi_{B_1,..., B_n, A_1,..., A_k, C_1,..., C_m}(\sigma_{r.A_1=s.A_1 \& r.A_2=s.A_2 \& ... r.A_k=s.A_k}(r \times s)).$$

Для подчеркивания важности приведенных операций реляционной алгебры, а также для уточнения понятия реляционной СУБД приведем следующее определение одного из ведущих специалистов в области реляционных баз данных К.Дж. Дейта: **«Будем называть систему реляционной, если она поддерживает, по крайней мере, реляционные базы данных, т.е. базы данных, которые могут восприниматься пользователем как таблицы и только как таблицы, операции селекции, проекции и соединения реляционной алгебры, не требуя при этом, чтобы каким-то образом были предопределены физические пути доступа для поддержки этих операций»**.

Краткие итоги: В лекции рассматриваются вопросы, связанные с формализацией наиболее распространенной в настоящее время модели данных СУБД – реляционной модели. Формальное описание реляционной модели и полученные на этой основе математические методы и алгоритмы позволяют формализовать ряд шагов проектирования реляционной базы данных , получить оптимальную (по определенным критериям) структуру базы данных и эффективные алгоритмы обработки. Здесь рассматривается формализованное описание отношений, формальные средства манипулирования данными в реляционной модели (дано понятие реляционного исчисления и реляционной алгебры, приводятся основные операции реляционной алгебры). Приводятся примеры представления запросов как последовательность формальных операций реляционной алгебры.

Вопросы, рассматриваемые в данной лекции, более подробно описаны в [1-5].

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Что такое схема отношения?

Вариант 1.
Что называется схемой отношения R?

⊞ множество имен атрибутов
☐ множество названий сущностей
☐ множество кортежей
☐ множество доменов

Вариант 2.
Чему соответствует понятие «схемы отношения»?

☐ двумерной таблице
⊞ описанию структуры конкретной таблицы
☐ описанию структуры любой таблицы
☐ множеству значений в таблице

Вариант 3.
Что соответствует имени атрибута в схеме отношения?

⊞ множество значений определенного типа данных
⊞ домен
☐ кортеж
☐ множество значений разных типов данных

Задача 2. Что такое отношение?

Вариант 1.
Что называется отношением?

☐ множество имен атрибутов таблицы
☐ множество названий сущностей
⊞ множество кортежей таблицы
☐ множество доменов таблицы

Вариант 2.
Чему соответствует понятие «отношения»?

☐ описанию структуры конкретной таблицы
☐ описанию структуры любой таблицы
☐ множеству значений в двумерной таблице
⊞ множеству строк в двумерной таблице

Вариант 3.
Что такое ключ отношения?

☐ подмножество атрибутов, таких что любые два кортежа отношения не совпадают по значениям этого подмножества
⊞ минимальное подмножество атрибутов, таких что любые два кортежа отношения не совпадают по значениям этого подмножества
☐ максимальное подмножество атрибутов, таких что любые два кортежа отношения не совпадают по значениям этого подмножества
☐ множество всех атрибутов

Задача 3. Что такое реляционная модель базы данных и реляционная база данных?

Вариант 1.
Что называется реляционной моделью базы данных?

⊞ совокупность схем отношений, используемых для представления концептуальной модели
☐ совокупность отношений, реализующих концептуальную модель
☐ текущие значения отношений
☐ модель данных реляционной СУБД

Вариант 2.
Что называется реляционной базой данных?

☐ совокупность схем отношений, используемых для представления концептуальной модели
☐ совокупность схем отношений, реализующих концептуальную модель
⊞ текущие значения отношений, описываемых концептуальной моделью
☐ модель данных реляционной СУБД

Вариант 3.
Какой формальный аппарат используется в реляционной модели для описания запросов к базе данных?

⊞ операции реляционной алгебры
⊞ формулы реляционного исчисления
☐ аппарат схем отношений
☐ ER-диаграммы

Задача 4. Операции объединения и пересечения отношений

Вариант 1.
Какие требования к отношениям накладываются для применения этих операций?

☐ одинаковое число строк
⊞ одинаковое число столбцов
☐ одинаковые названия столбцов
☐ равные размеры таблиц

Вариант 2.
Что называется объединением отношений?

⊞ множество кортежей, принадлежащих одному или другому отношению, или им обоим
☐ множество кортежей, принадлежащих одному или другому отношению
☐ множество кортежей, принадлежащих обоим отношениям
☐ множество кортежей, одна часть которого представляет кортеж из первого отношения, вторая часть – кортеж из второго отношения

Вариант 3.
Что называется разностью отношений?

☐ множество кортежей, которые представляют кортежи из первого отношения за минусом тех значений, которые входят в кортежи второго отношения
⊞ множество кортежей, принадлежащих первому отношению, но не принадлежащих второму отношению
☐ множество кортежей отношения, которое получается из первого отношения удалением атрибутов второго отношения
☐ множество атрибутов, которое получается из первого отношения удалением атрибутов второго отношения

Задача 5. Что такое операция «декартово произведение»?

Вариант 1.
Что представляет собой результат операции «декартово произведение» двух отношений?

☐ схему отношения, составленную из двух схем отношений
⊞ новое отношение со схемой отношения, составленной из двух исходных схем отношений
☐ множество всевозможных кортежей, первая часть которых представляет кортежи первого отношения, вторая часть - кортежи второго отношения
☐ множество кортежей, получаемых добавлением к кортежам первого отношения кортеж из соответствующей строчки второго отношения

Вариант 2.
Если арность отношений, участвующих в операции «декартово произведение» равна соответственно $k1$ и $k2$, чему равна арность полученного отношения?

⊞ $k1+ k2$
☐ $k1* k2$
☐ $k1 - k2$
☐ $k1+ k1 * k2$

Вариант 3.
Если арность отношений, участвующих в операции «декартово произведение» равна соответственно $k1$ и $k2$, чему равно количество кортежей в полученном отношении?

☐ $k1+ k2$
⊞ $k1* k2$
☐ $k1 - k2$
☐ $(k1+ k2) * k2$

Вариант 4.
Для чего используется операция «декартово произведение»?

⊞ для «склейки» таблиц
⊞ для перехода от значений атрибута в одной таблице к такому же значению атрибута в другой таблице
☐ для объединения таблиц
☐ для поиска данных в таблицах

Задача 6. С помощью каких операций осуществляется выборка данных из таблицы?

Вариант 1.
С помощью какой операции выбираются нужные столбцы таблицы?

☐ селекция
⊞ проекция
☐ декартово произведение
☐ разность

Вариант 2.
С помощью какой операции выбираются нужные кортежи отношения?

☐ проекция
☐ декартово произведение
☐ разность
⊞ селекция

Вариант 3.
Какие операнды могут входить в формулу, определяющую условия выборки?

⊞ имена атрибутов
⊞ константы
⊞ арифметические операторы сравнения
⊞ логические операторы сравнения

Задача 7. В чем смысл операций соединения?

Вариант 1.
Какие операции входят в операции соединения?

⊞ селекция
⊞ проекция
⊞ декартово произведение
☐ разность

Вариант 2.
Для чего нужны операции соединения?

⊞ для «склейки» таблиц
⊞ для перехода от значений атрибутов в одной таблице к таким же значениям атрибутов в другой таблице
⊞ для объединения таблиц с совпадающими значениями одного или нескольких атрибутов
⊞ для реализации выборки данных на основе использования двух таблиц, связанных общими атрибутами

Вариант 3.
В чем отличие операции «θ-соединение» от операции «естественное соединение»?

☐ используется меньше операций реляционной алгебры
⊞ сравниваются значения одного общего атрибута
☐ накладывается меньше условий на исходные отношения
☐ при сравнении значений может использоваться больше арифметических операторов

Задача 8. Какая система работы с базой данных является реляционной?

Вариант 1.
Как пользователь должен воспринимать реляционную базу данных?

⊞ как набор таблиц
☐ как иерархическую структуру
☐ как наборы записей с указателями
☐ как совокупность файлов

Вариант 2.
Какие операции должна поддерживать реляционная система?

☐ поиск
☐ добавление
☐ удаление
⊞ селекция
⊞ проекция
⊞ соединение

Вариант 3.
Как программист указывает физические пути доступа к данным в памяти компьютера при работе в реляционных системах?

⊞ не указывает, указывает только операции
☐ указывает в программе выполнения операции
☐ указывает в прикладной программе
☐ указывает при обращении к операции

Литература

1. Дейт К.Дж. Введение в системы баз данных: Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с.
2. Карпова Т. Базы данных. Модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
3. Ульман Дж. Основы систем баз данных: Пер. с англ. / Под ред. М.Р. Когаловского. – М.: Финансы и статистика, 1983. – 334 с.
4. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Н.Новгород: Изд-во ННГУ, 2004. 271 с.
5. Ульман Дж. Д., Уидом Дж. Введение в системы баз данных: Пер. с англ. – М.: Лори, 2000. – 374 с.

Лекция 8. Использование формального аппарата для оптимизации схем отношений

Лекция посвящена вопросам оптимизации схем отношений на основе формальных методов теории реляционных баз данных. Разбирается пример приведения таблицы к третьей нормальной форме, оптимальной по ряду показателей. Рассматриваются вопросы целостности данных в реляционных СУБД.

Ключевые термины: выбор рациональной схемы отношения, аномалия включения, аномалия удаления, аномалия обновления, функциональная зависимость, аксиомы Армстронга, нормальная форма, декомпозиция схемы отношения, нормализация, условия целостности реляционной модели.

Цель лекции: показать возможность эффективного использования формальных методов построения оптимальной (по определенным показателям) структуры реляционной базы данных путем нормализации схем отношений.

8.1. Проблема выбора рациональных схем отношений

При представлении концептуальной схемы в виде реляционной модели возможны различные варианты выбора схем отношений. Одни варианты выбора рассматривались в предыдущих разделах (п. 6.2.3), другие получаются объединением (или разбиением) некоторых схем отношений. От правильного выбора схем отношений, представляющих концептуальную схему, в значительной степени будет зависеть эффективность функционирования базы данных.

Рассмотрим для примера конкретную схему отношений и проанализируем её недостатки. Предположим, что данные о студентах, факультетах, специальностях, включены в таблицу со следующей схемой отношения:

СТУДЕНТ (Код студента, Фамилия, Название факультета, Название специальности).

Эта схема отношений обусловливает следующие недостатки соответствующей базы данных:

- Дублирование информации (избыточность). У студентов, обучающихся на одном факультете, будет повторяться название факультета. Для разных факультетов будут повторяться специальности.
- Потенциальная противоречивость (аномалии обновления). Если, например, изменится название специальности, то изменяя её в одном кортеже (у одного студента), необходимо изменять и во всех других кортежах, где она присутствует.
- Потенциальная возможность потери сведений (аномалии удаления). При удалении информации о всех студентах, поступающих на определенную специальность, мы теряем все сведения об этой специальности.
- Потенциальная возможность невключения информации в базу данных (аномалии включения). В базе данных будут отсутствовать сведения о специальности, если на ней нет обучающихся студентов.

В теории реляционных баз данных существуют формальные методы построения реляционной модели базы данных, в которой отсутствует избыточность и аномалии обновления, удаления и включения.

Нормализация. Первая нормальная форма.

Построение рационального варианта схем отношений (обладающего лучшими свойствами при операциях включения, модификации и удаления данных, чем все остальные наборы схем) осуществляется путем так называемой нормализации схем отношений. Нормализация производится в несколько этапов. На начальном этапе схема отношений должна находиться в первой нормальной форме (1НФ).

Отношение находится в первой нормальной форме, если все атрибуты отношения принимают простые значения (атомарные или неделимые), не являющиеся множеством или кортежем из более элементарных составляющих.

Рассмотрим следующий пример.

Таблица представляет сущность ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ

Код студента	Фамилия	Код экзамена	Предмет и дата	Оценка
1	Сергеев	1	Математика 5.06.08	4
2	Иванов	1		5
1	Сергеев	2	Физика 9.06.08	5
2	Иванов	2		5

Данная таблица не находится в первой нормальной форме (ненормализованная), так как на пересечении строки и четвертого столбца располагается не одно значение, а несколько.

Перейдем к первой нормальной форме. Для этого разнесем значения предмета и даты в разные столбцы и запишем для каждой строчки информацию по экзамену.

Код студента	Фамилия	Код экзамена	Предмет	Дата	Оценка
1	Сергеев	1	Математика	5.08.03	4
2	Иванов	1	Математика	5.08.03	5
1	Сергеев	2	Физика	9.08.03	5
2	Иванов	2	Физика	9.08.03	5

Теперь на пересечении любой сроки и любого столбца находится одно значение и, следовательно, данная таблица находится в первой нормальной форме.

Далее отношение, представленное в первой нормальной форме, последовательно преобразуется во вторую и третью нормальные формы. Процесс построения второй и третьей нормальных форм будет описан в следующих подразделах. При некоторых предположениях о данных третья нормальная форма является искомым наилучшим вариантом.

Если эти предположения не выполняются, то процесс нормализации продолжается и отношение преобразуется в четвертую и пятую нормальные формы. Построение соответствующих форм описано в литературе и в данной книге не рассматривается.

Прежде чем перейти к построению второй нормальной формы, необходимо определить ряд формальных понятий.

8.2. Функциональные зависимости (зависимости между атрибутами отношения)

Пусть R ($A_1, A_2, ..., A_n$) – схема отношения, а X и Y – подмножества $\{A_1, A_2, \ldots, A_n\}$.

Функциональная зависимость на отношении R – это утверждение вида «Если два кортежа R совпадают по атрибутам множества $X \subset \{A_1, A_2, \ldots, A_n\}$ (т.е. эти кортежи имеют в соответствующих друг другу компонентах одни и те же значения для каждого атрибута множества X), то они должны совпадать и по атрибутам множества $Y \subset \{A_1, A_2, \ldots, A_n\}$. Формально эта зависимость записывается выражением $X \rightarrow Y$, причем говорится, что X функционально определяет Y. Часто используется другое утверждание: ***X функционально определяет Y или Y функционально зависит от X (обозначается X → Y) тогда и только тогда, когда каждое значение множества X отношения R связано с одним значением множества Y отношения R. Иначе говоря, если два кортежа R совпадают по значению X, они совпадают и по значению Y.***

Замечание. Вообще говоря, под термином «отношение» могут подразумеваться два понятия:

- отношение как переменная, которая может принимать разные значения (таблица, в строки и столбцы которой могут быть вписаны разные значения);
- отношение, как набор конкретных значений (таблица с заполненными элементами).

Функциональные зависимости характеризуют все отношения, которые могут быть значениями схемы отношения R в принципе. Поэтому единственный способ определить функциональные зависимости – внимательно проанализировать семантику (смысл) атрибутов.

Функциональные зависимости являются, в частности, ограничениями целостности, поэтому целесообразно проверять их при каждом обновлении базы данных.

Пример функциональных зависимостей для отношения ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ

Код студента→ Фамилия

Код студента, Код экзамена→ Оценка

Пример функциональных зависимостей для отношения СТУДЕНТ, приведенного в начале настоящей лекции

Код студента→ Фамилия,

Код студента→ Факультет

Заметим, что последняя зависимость существует при условии, что один студент не может обучаться на нескольких факультетах.

Полное множество функциональных зависимостей

Для каждого отношения существует вполне определенное множество функциональных зависимостей между атрибутами данного отношения. Причем из одной или более функциональных зависимостей, присущих рассматриваемому отношению, можно вывести другие функциональные зависимости, также присущие этому отношению.

Заданное множество функциональных зависимостей для отношения R обозначим F, полное множество функциональных зависимостей, которые логически можно получить из F, называется замыканием F и обозначается F^+.

Если множество функциональных зависимостей совпадает с замыканием данного множества, то такое множество функциональных зависимостей называется полным.

Введенные понятия позволяют формально определить понятие ключа.

Пусть существует некоторая схема R с атрибутами $A_1A_2…A_n$, F – некоторое множество функциональных зависимостей и X – некоторое подмножество R. Тогда X называется ключом, если, во-первых, в F^+ существует зависимость $X \rightarrow A_1A_2…A_n$ и, во-вторых, ни для какого подмножества Y, входящего в X, зависимость $Y \rightarrow A_1A_2…A_n$ не принадлежит F^+.

Полной функциональной зависимостью называется зависимость неключевого атрибута от всего составного ключа.

Частичной функциональной зависимостью будем называть зависимость неключевого атрибута от части составного ключа.

Для вычисления замыкания множества функциональных зависимостей используются следующие **правила вывода** (аксиомы Армстронга):

Пусть известна некоторая схема отношения $R\ \{A_1, A_2,…, A_n\}$ с множеством атрибутов $U=\{A_1, A_2,…, A_n\}$ и множество функциональных зависимостей F, заданных на множестве U.

Аксиома рефлективности. Если Y входит в X, а X входит в U ($Y \subseteq X \subseteq U$), то $X \rightarrow Y$ логически следует из F. Это правило дает тривиальные зависимости, так как в них правая часть содержится в левой части.

Аксиома пополнения. Если $X \rightarrow Y$ и Z есть подмножество U, то $XZ \rightarrow YZ$. В данном случае функциональная зависимость $X \rightarrow Y$ либо содержалась в исходном множестве F, либо может быть выведена из F с использованием описываемых аксиом.

Аксиома транзитивности. Если $X \rightarrow Y$ и $Y \rightarrow Z$, то $X \rightarrow Z$.

Справедлива следующая **теорема**. *Аксиомы Армстронга являются полными и надежными.*

Это значит, что используя их мы выведем все возможные функциональные зависимости, логически следующие из F, и не выведем никаких лишних зависимостей.

Существует несколько других правил вывода, которые следуют из аксиом Армстронга.

Правило самоопределения. $X \rightarrow X$.

Правило объединения. Если $X \rightarrow Y$ и $X \rightarrow Z$, то $X \rightarrow Y \cup Z$.

Правило псевдотранзитивности. Если $X \rightarrow Y$ и $W \cup Y \rightarrow Z$, то $X \cup W \rightarrow Z$.

Правило композиции. Если $X \rightarrow Y$ и $Z \rightarrow W$, то $X \cup W \rightarrow Y \cup W$.

Правило декомпозиции. Если $X \rightarrow Y$ и Z входит в Y, то $X \rightarrow Z$.

Надо отметить, что вычисление замыкания множества функциональных зависимостей является трудоемкой задачей при достаточно большом количестве атрибутов (за счет выписывания большого количества тривиальных зависимостей).

8.3. Декомпозиция схемы отношения

Последовательный переход от одной нормальной формы к другой при нормализации схем отношений реализуется через декомпозицию. Основной операцией, с помощью которой осуществляется декомпозиция, является проекция.

Декомпозицией схемы отношения $R = \{A_1, A_2, \ldots A_n\}$ называется замена ее совокупностью подмножеств R, таких, что их объединение дает R. При этом допускается, чтобы подмножества были пересекающимися.

Алгоритм декомпозиции основан на следующей теореме.

Теорема о декомпозиции. ***Пусть R(A, B, C) – отношение, A, B, C – атрибуты.***

Если R удовлетворяет зависимости A→B, то R равно соединению его проекций A, B и A, C

$$R(A, B, C) = R(A, B), \quad R(A, C)$$

При нормализации необходимо выбирать такие декомпозиции, которые обладают свойством соединения без потерь. В этом случае, декомпозиция должна обеспечить то, что запросы (выборка данных по условию) к исходному отношению и отношениям, получаемым в результате декомпозиции, дадут одинаковый результат. Соответствующее условие будет выполняться, если каждый кортеж отношения R может быть представлен как естественное соединение его проекций на каждое из подмножеств. Для проверки, обладает ли декомпозиция данным свойством, используется специальные алгоритмы, описанные в литературе (в данной книге не рассматриваются).

Вторым важнейшим желательным свойством декомпозиции является свойство сохранения функциональных зависимостей. Стремление к тому, чтобы декомпозиция сохраняла зависимости, естественно. Функциональные зависимости являются некоторыми ограничениями на данные. Если декомпозиция не обладает этим свойством, то для того чтобы проверить, не нарушаются ли при вводе данных условия целостности (функциональные зависимости), нам приходится соединять все проекции.

Таким образом, для правильно построенного проекта базы данных необходимо, чтобы декомпозиции обладали свойством соединения без потерь, и желательно, чтобы они обладали свойством сохранения функциональных зависимостей.

8.4 .Выбор рационального набора схем отношений путем нормализации

Вторая нормальная форма (2НФ)

Отношение находится в 2НФ, если оно находится в 1НФ и каждый неключевой атрибут зависит от всего первичного ключа (не зависит от части ключа).

Для перевода отношения в 2НФ необходимо, используя операцию проекции, разложить его на несколько отношений следующим образом:

1) построить проекцию без атрибутов, находящихся в частичной функциональной зависимости от первичного ключа;

2) построить проекции на части составного ключа и атрибуты, зависящие от этих частей.

Третья нормальная форма (3НФ)

Отношение находится в 3НФ, если оно находится в 2НФ и каждый ключевой атрибут нетранзитивно зависит от первичного ключа.

Отношение находится в 3НФ в том и только том случае, если все неключевые атрибуты отношения взаимно независимы и полностью зависят от первичного ключа.

Оказывается, что любая схема отношений может быть приведена к 3НФ декомпозицией, обладающей свойствами соединения без потерь и сохраняющей зависимости.

Мотивировка третьей нормальной формы

Третья нормальная форма исключает избыточность и аномалии включения и удаления.

К сожалению, 3НФ не предотвращает все возможные аномалии.

Нормальная форма Бойса-Кодда (НФБК)

Если в R для каждой зависимости X→A, где A не принадлежит X, X включает в себя некоторый ключ, то говорят, что данное отношение находится в нормальной форме Бойса-Кодда.

Детерминантом функциональной зависимости называется минимальная группа атрибутов, от которой зависит некоторый другой атрибут или группа атрибутов, причем эта зависимость нетривиальная.

Отношение находится в НФБК тогда и только тогда, когда каждый его детерминант является потенциальным ключом.

НФБК является более строгой версией 3НФ. Иными словами, любое отношение, находящееся в НФБК, находится в 3НФ. Обратное неверно.

Мотивировка нормальной формы Бойса-Кодда

В нормальной форме Бойса-Кодда не существует избыточности и аномалий включения, удаления и модификации. Оказывается, что любая схема отношения может быть приведена в нормальную форму Бойса-Кодда таким образом, чтобы декомпозиция обладала свойством соединения без потерь. Однако схема отношения может быть неприводимой в НФБК с сохранением зависимостей. В этом случае приходится довольствоваться третьей нормальной формой.

8.5. Пример нормализации до 3НФ

Для улучшения структуры реляционной базы данных (устранения возможных аномалий) необходимо привести все таблицы базы данных к третьей нормальной форме или в более высокой форме (если это возможно). Таким образом, задача сводится к проверке нормализации всех сущностей, отображающихся в таблицы базы данных. Если таблица, получающаяся из некоторой сущности, не является таблицей в третьей нормальной форме, то она должна быть заменена на несколько таблиц, находящихся в третьей нормальной форме.

Продолжим рассмотрение примера с отношением ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ

В начале этой лекции мы привели отношение к первой нормальной форме.

Код сту-	Фамилия	Код	Предмет	Дата	Оценка

дента		экзамена			
1	Сергеев	1	Математика	5.08.03	4
2	Иванов	1	Математика	5.08.03	5
1	Сергеев	2	Физика	9.08.03	5
2	Иванов	2	Физика	9.08.03	5

Ключом данного отношения будет совокупность атрибутов – Код студента и Код экзамена.

Для более краткой записи процесса нормальзации введем следующие обозначения:

КС – код студента, КЭ – код экзамена, Ф – фамилия, П – предмет, Д – дата, О-оценка.

Выпишем функциональные зависимости

КС, КЭ → Ф, П, Д, О

КС, КЭ → Ф

КС, КЭ → П

КС, КЭ → Д

КС, КЭ → О

КЭ → П

КЭ → Д

КС → Ф

В соответствии с определением, отношение находится во второй нормальной форме (2НФ), если оно находтяс в 1НФ и каждый неключевой атрибут зависит от первичного ключа и не зависит от части ключа. Здесь атрибуты П, Д, Ф зависят от части ключа. чобы избавиться от этих зависимостей необходимо произвести декомпозицию отношения. Для этого используем теорему о декомпозиции.

Имеем отношение *R*(КС, Ф, КЭ, П, Д, О). Возьмем зависимость КС → Ф в соответствии с формулировкой теоремы исходное отношение равно соединению его проекций *R*1(КС, Ф) и *R*2(КС, КЭ, П, Д, О).

В отношении *R*1(КС, Ф) существует функциональная зависимость КС → Ф, ключ КС – составной не ключевой атрибут Ф не зависит от части ключа. Это отношение находтся в 2 НФ. Так как в этом отношении нет транзитивных зависимостей, отношение *R*(КС, Ф) находится в 3НФ, что и требовалось.

Рассмотрим отношение *R*2(КС, КЭ, П, Д, О) с составным ключом КС, КЭ. Здесь есть зависимость КЭ → П, КЭ → Д, КЭ → П, Д. Атрибуты П,Д зависят от части ключа, следовательно отношение не находится в 2НФ. В соответствии с теоремой о декомпозиции исходное отношение (используем функциональную зависимость КЭ → П, Д) равно соединению проекций *R*3(КЭ, П, Д), *R*4(КС, КЭ, О). В отношении *R*3(КЭ, П, Д) существуют функциональные зависимости КЭ → П, КЭ → Д, КЭ → П, Д. Зависимости неключевых атрибутов от части ключа нет, следовательно отношение находится в 2НФ. Транзитивных зависимостей в этом отношении так же нет, следовательно отношение находится в 3НФ.

Таким образом, исходное отношение приведено в к трем отношениям, каждое из которых находится в третьей нормальной форме *R*1(КС, Ф), *R*3(КЭ, П, Д), *R*4(КС, КЭ, О).

Заметим, что в отношении *R*4 атрибуты КС, КЭ являются внешними ключами, используемыми для установления связей с другими отношениями. Представим полученную модель в виде диаграммы объектов-связей (ER-диаграммы). Для наглядности и возможности последующего программирования перейдем к английским названиям объектов (отношений) и атрибутов.

Отношение *R*1 представляет объект student с атрибутами id_st (первичный ключ), surname.

Отношение *R*3 представляет объект exam_st с атрибутами id_ex (первичный ключ), subject, date.

Отношение *R*4 представляет объект mark_st с атрибутами id_st (внешний ключ), id_ex (внешний ключ), mark. Первичный ключ здесь id_st, id_ex.

Соответствующая ER-диаграмма изображена на рис. 8.1.

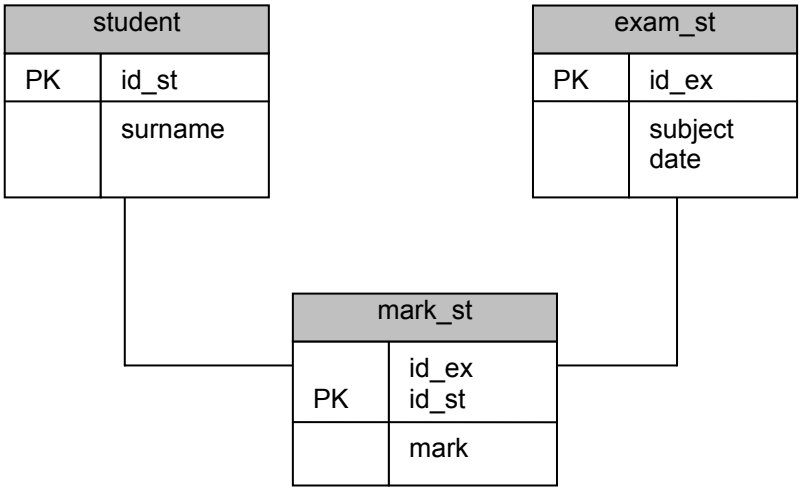


Рис. 8.1. ER-диаграмма, представляющая рассмотренный фрагмент предметной области.

8.6. Целостная часть реляционной модели. Реализация условия целостности данных в современных СУБД

Напомним, что под целостностью базы данных понимается то, что в ней содержится полная, непротиворечивая и адекватно отражающая предметную часть (правильная) информация. Поддержка целостности в реляционных БД основана на выполнении следующих требований.

1. Первое требование называется **требованием целостности сущностей**. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е., другими словами, любое отношение должно обладать определенным первичным ключом. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

2. Второе требование называется **требованием целостности по ссылкам**. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира

представляются в реляционной БД в виде нескольких кортежей нескольких отношений. Связь между отношениями осуществляется с помощью миграции ключа.

Пример внешнего ключа.

СТУДЕНТ (Код студента, Фамилия) сдает ЭКЗАМЕН (Код студента, Предмет, Оценка).

Атрибут Код студента сущности ЭКЗАМЕН называется ***внешним ключом,*** поскольку его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения – отношения Студент (мы предполагаем, что поле Код студента является ключом отношения Студент).

Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором такой же атрибут является первичным ключом.

Требование целостности по ссылкам или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа в ссылающемся отношении в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать).

Ограничения целостности сущности и по ссылкам должны поддерживаться СУБД. Для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа. (В Access для этого предназначена специальная реализация целочисленного поля – поле типа «Счетчик».) С целостностью по ссылкам дела обстоят несколько более сложно.

Понятно, что при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа.

Но как быть при удалении кортежа из отношения, на которое ведет ссылка?

Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

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

Заметим, что все современные СУБД поддерживают и целостность сущностей, и целостность по ссылкам, но позволяют пользователям выключать данные ограничения и, таким

образом, строить базы данных, не соответствующие реляционной модели. Опыт показывает, что отход от основных положений реляционной модели приводит к краткосрочному выигрышу – алгоритмы становятся проще, но впоследствии серьезно усложняют задачу, особенно ее сопровождение.

Краткие итоги: Лекция посвящена вопросам оптимизации схем отношений (структуры реляционной базы данных) на основе формальных методов теории реляционных баз данных. Здесь рассматривается ряд необходимых для этого понятий (функциональная зависимость, нормальные формы, декомпозиция схем отношений). Разбирается пример приведения таблицы к третьей нормальной форме, оптимальной по ряду показателей (исключающей избыточность, аномалии включения и удаления). Рассматриваются вопросы реализации целостности данных в реляциионных СУБД.

В лекции рассматриваются вопросы использование формального аппарата для оптимизации схем отношений. Сформулирована проблема выбора рациональных схем отношений и пути реализации такого выбора путем нормализации (последовательного преобразования схемы отношения в ряд нормальных форм). Для формального описания соответствующего процесса определены понятие функциональной зависимости (зависимости между атрибутами отношения), ключа, сформулированы правила вывода множества функциональных зависимостей, понятие декомпозиции схемы отношения. Определены первая, вторая, третья нормальные формы и нормальная форма Бойса-Кодда. Приведен пример нормализации до 3НФ. Рассмотрены вопросы реализации условий целостности данных в реляционных СУБД.

Вопросы настоящей лекции рассматриваются в [1-8].

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. В чем состоит задача выбора рациональных схем отношений?

Вариант 1.
Какие проблемы устраняются за счет выбора рациональных схем отношений?

⊞ дублирование
⊞ потенциальная противоречивость
⊞ потенциальная возможность потери сведений
⊞ потенциальная возможность не включения информации в базу данных
☐ увеличение количества схем отношений

Вариант 2.
С чем связано основное дублирование информации в реляционной базе данных?

☐ с повторением одинаковых строк в одной таблице
☐ с повторением одинаковых столбцов в одной таблице
⊞ с повторением одинаковых значений атрибутов в одной таблице
☐ с повторением одинаковых значений атрибута в разных таблицах

Вариант 3.
Какие аномалии необходимо устранить при проектировании реляционной базы данных?

☐ создания
⊞ удаления
⊞ обновления
⊞ включения
☐ выключения

Задача 2. Как механизм используется для выбора рациональных схем отношений?

Вариант 1.
Как осуществляется выбор рациональных схем отношений?

⊞ путем нормализации
⊞ путем последовательного преобразования отношений к ряду нормальных форм
☐ путем объединения схем отношений
⊞ путем декомпозиции схем отношений

Вариант 2.
Что такое нормализация?

⊞ последовательное преобразование отношений к ряду нормальных форм
☐ определенное объединение схем отношений
⊞ определенная декомпозиция схем отношений
☐ преобразование отношений с использованием операций реляционной алгебры

Вариант 3.
Что такое первая нормальная форма?

⊞ значения всех атрибутов отношения являются простыми
⊞ значения всех атрибутов отношения являются неделимыми
⊞ значения всех атрибутов отношения являются атомарными
☐ значения всех атрибутов отношения являются кортежами
☐ значения некоторых атрибутов отношения являются атомарными
☐ значения некоторых атрибутов отношения являются кортежами

Задача 3. Дать характеристику функциональных зависимостей.

Вариант 1.
Что такое X функционально определяет Y?

⊞ каждое значение множества X связано с одним значением множества Y
⊞ если два кортежа совпадают по значениям X, то они совпадают по значениям Y
☐ Y является функцией X
☐ Y зависит от X
☐ каждое значение множества Y связано с одним значением множества X
☐ если два кортежа совпадают по значениям Y, то они совпадают по значениям X

Вариант 2.
Что характеризуют функциональные зависимости?

☐ схему отношения
⊞ все возможные значения отношения
⊞ все возможные значения строк отношения
⊞ отношение как переменную

Вариант 3.
Как можно использовать функциональные зависимости для защиты логической целостности базы данных?

⊞ как ограничения целостности
⊞ для проверки выполнения функциональной зависимости при обновлении данных
☐ для проверки правильности работы прикладных программ
☐ для автоматизированного формирования соответствующих данных

Задача 4. Как осуществляется нормализация схем отношений?

Вариант 1.
Что такое декомпозиция схемы отношения?

- ☐ замена схемы отношения $R = \{A_1, A_2, \ldots A_n\}$ совокупностью подмножеств R: $R1$, $R2$ таких, что $R = R1 \cap R2$
- ⊞ замена схемы отношения $R = \{A_1, A_2, \ldots A_n\}$ совокупностью подмножеств R: $R1$, $R2$ таких, что $R = R1 \cup R2$
- ☐ замена схемы отношения $R = \{A_1, A_2, \ldots A_n\}$совокупностью подмножеств R: $R1$, $R2$ таких, что $R = R1 \times R2$
- ☐ замена схемы отношения $R = \{A_1, A_2, \ldots A_n\}$совокупностью подмножеств R: $R1$, $R2$ таких, что $R = R1 + R2$

Вариант 2.
Как формулируется теорема о декомпозиции?

- ⊞ если $R(A, B, C)$ удовлетворяет зависимости $A \rightarrow B$, то R равно соединению проекций $R(A, B)$, $R(A, C)$
- ☐ если $R(A, B, C)$ удовлетворяет зависимости $A \rightarrow B$, то R равно соединению проекций $R(A, B)$, $R(B, C)$
- ☐ если $R(A, B, C)$ удовлетворяет зависимости $A \rightarrow B$, то R равно соединению проекций $R(A, C)$, $R(B, C)$
- ⊞ если $R(A, B, C)$ удовлетворяет зависимости $A \rightarrow B$, то R равно соединению проекций $R(A, C)$, $R(A, B)$

Вариант 3.
Какими свойствами должны обладать декомпозиции при нормализации?

- ⊞ сохранение функциональных зависимостей
- ⊞ соединения без потерь
- ☐ разбиение без потерь
- ☐ сохранение ключа

Задача 5. Приведение ко второй нормальной форме.

Вариант 1.
При каких условиях отношение находится во второй нормальной форме?

⊞ если оно находится в первой нормальной форме и каждый неключевой атрибут зависит от всего первичного ключа
☐ если оно находится в первой нормальной форме и каждый неключевой атрибут зависит от части первичного ключа
☐ если оно находится в первой нормальной форме и каждый неключевой атрибут не зависит от первичного ключа
⊞ если оно находится в первой нормальной форме и каждый неключевой атрибут не зависит от части первичного ключа

Вариант 2.
Как осуществляется приведение ко второй нормальной форме?

⊞ производится декомпозиция с использованием функциональной зависимости, в которой неключевой атрибут зависит от части первичного ключа
☐ сначала схема отношения приводится к первой нормальной форме
☐ производится декомпозиция с использованием функциональной зависимости, в которой неключевой атрибут зависит от всего первичного ключа
☐ производится декомпозиция с использованием функциональной зависимости, в которой неключевой атрибут зависит от неключевого атрибута

Вариант 3.
Какие аномалии устраняются второй нормальной формой?

☐ удаления
☐ избыточность
☐ обновления
☐ включения
⊞ никакие

Задача 6. Приведение к третьей нормальной форме.

Вариант 1.
При каких условиях отношение находится в третьей нормальной форме?

- ☐ если оно находится во второй нормальной форме и каждый неключевой атрибут зависит от всего первичного ключа
- ☐ если оно находится во второй нормальной форме и каждый неключевой атрибут нетранзитивно зависит от части первичного ключа
- ⊞ если оно находится во второй нормальной форме и каждый неключевой атрибут нетранзитивно зависит от первичного ключа
- ☐ если оно находится во второй нормальной форме и каждый неключевой атрибут не зависит от части первичного ключа

Вариант 2.
Как осуществляется приведение к третьей нормальной форме?

- ☐ производится декомпозиция с использованием функциональной зависимости, в которой неключевой атрибут зависит от части первичного ключа
- ⊞ сначала схема отношения приводится ко второй нормальной форме
- ⊞ производится декомпозиция с использованием функциональной зависимости, в которой неключевой атрибут транзитивно зависит от первичного ключа
- ☐ производится декомпозиция с использованием функциональной зависимости, в которой неключевой атрибут нетранзитивно зависит от первичного ключ

Вариант 3.
Какие аномалии устраняются третьей нормальной формой?

- ☐ удаления
- ⊞ избыточность
- ☐ обновления
- ⊞ включения
- ⊞ никакие

Задача 7. Анализ функциональных зависимостей следующего отношения.

Код студента	Фамилия	Код экзамена	Предмет	Дата	Оценка
1	Сергеев	1	Математика	5.08.03	4
2	Иванов	1	Математика	5.08.03	5
1	Сергеев	2	Физика	9.08.03	5
2	Иванов	2	Физика	9.08.03	5

Вариант 1.
Какие из перечисленных зависимостей существуют в этом отношении?

⊞ Код студента →Фамилия
⊞ Предмет→Дата
⊞ Код экзамена →Дата
☐ Код студента→Оценка

Вариант 2.
Каких из перечисленных зависимостей не существует в этом отношении?

☐ Код студента, Фамилия→Фамилия
☐ Предмет→Дата
☐ Код экзамена →Дата
⊞ Код студента→Оценка
☐ Код студента, Предмет→Оценка

Вариант 3.
В какой зависимости определен первичный ключ отношения?

☐ Код студента, Фамилия→Фамилия, Предмет
☐ Предмет→Дата
☐ Код экзамена →Дата
☐ Код студента→Оценка
☐ Код студента, Предмет→Оценка
⊞ Код студента, Код предмета →Дата, Оценка

Задача 8. Как осуществляется поддержка целостности данных в реляционных СУБД?

Вариант 1.
Какие требования должны выполняться для поддержки целостности данных в реляционных СУБД?

- ⊞ уникальность любого кортежа отношения
- ⊞ наличие у любого отношения первичного ключа
- ⊞ для каждого значения внешнего ключа в ссылающемся отношении должен существовать кортеж с таким же значением первичного ключа в отношении, на которое ссылаются.
- ☐ для каждого значения первичного ключа в ссылающемся отношении должен существовать кортеж с таким же значением внешнего ключа в отношении, на которое ссылаются

Вариант 2.
В чем состоят ограничения целостности сущности и по ссылкам?

- ⊞ для каждого значения внешнего ключа в ссылающемся отношении должен существовать кортеж с таким же значением первичного ключа в отношении, на которое ссылаются.
- ☐ для каждого значения первичного ключа в ссылающемся отношении должен существовать кортеж с таким же значением внешнего ключа в отношении, на которое ссылаются
- ☐ должны быть экземпляры сущностей
- ⊞ экземпляры сущностей должны уникально идентифицироваться

Вариант 3.
Какие варианты поддержки ограничений целостности по ссылкам используются в современных СУБД?

- ⊞ запрещается удалять кортеж, на который существуют ссылки.
- ⊞ при удалении кортежа, на который существуют ссылки, во всех ссылающихся кортежах значение внешнего ключа заменяется на неопределенное
- ⊞ при удалении кортежа, на который существуют ссылки, из ссылающегося отношения удаляются все ссылающиеся кортежи
- ☐ при удалении кортежа, на который существуют ссылки, удаляется ссылающееся отношение

Литература

1. Карпова Т. Базы данных. Модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
2. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
3. Ульман Дж. Д., Уидом Дж. Введение в системы баз данных: Пер. с англ. – М.: Лори, 2000. – 374 с.
4. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для вузов. – СПб.: КОРОНА принт, 2000. – 416 с.
5. Толстобров А.П. Управление данными. Учебное пособие. Воронеж: Изд-во Воронежского ГУ, 2007 – 205 с.
6. Хансен Г., Хансен Дж. Базы данных: разработка и управление: Пер. с англ. – М.: ЗАО «Издательство «БИНОМ», 1999. – 704 с.
7. Дейт К.Дж. Введение в системы баз данных: Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с.
8. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Н.Новгород: Изд-во ННГУ, 2004. 271 с.

Лекция 9. Физические модели данных (внутренний уровень)

Лекция посвящена вопросам физической организации данных в памяти компьютера. Здесь описывается структура памяти компьютера, и, с учетом ее особенностей представлены структуры хранения данных в оперативной и внешней памяти.

Ключевые термины: физические модели данных, структуры хранения, представление данных в памяти ЭВМ, физическая запись, последовательный файл, списковая структура, B-дерево, хэш-функция, индексирование.

Цель лекции: дать представление об основных типовых способах организации данных в памяти ЭВМ в СУБД с оценкой соответствующих моделей по времени доступа к данным в базе данных и по объему занимаемой памяти.

Как уже отмечалось, концептуальная схема, специфицированная к СУБД, автоматически отображается в структуру хранения программами СУБД. Внешний пользователь может ничего не знать о том, как его представление о данных физически организовано в памяти вычислительной системы. Тем не менее от физического размещения данных в памяти ЭВМ существенно зависит время решения прикладных задач. В связи с этим, даже на одном из начальных этапов проектирования базы данных – этапе выбора СУБД, желательно знать возможности физических структур хранения, представляемых конкретными СУБД, и оценивать временные характеристики проектируемой базы данных с учетом этих возможностей.

Способы физической организации данных в различных СУБД, как правило, различны и определяются типом используемой ЭВМ, инструментальными средствами разработки СУБД, а также критериями, которыми руководствуются разработчики СУБД при выборе методов размещения данных и способов доступа к этим данным. Заметим, что наиболее распространенным критерием служит время доступа к данным, однако в качестве критерия может выбираться, например, трудоемкость реализации соответствующих методов.

В настоящей лекции будут рассмотрены типовые физические модели организации данных в конкретных СУБД.

Физические модели данных служат для отображения моделей данных. Основными понятиями модели данных являются поле, логическая запись, логический файл. Слово «логический» введено, чтобы отличать понятия, относящиеся к логической модели данных, от понятий, относящихся к физической модели данных. ***Основными понятиями физической модели данных, используемыми для представления логической модели данных, являются поле, физическая запись, физический файл.*** В частности, логическая запись, состоящая из полей, может быть представлена в виде физической записи (из тех же полей), логический файл – в виде физического файла. Прежде чем конкретизировать понятия, относящиеся к физической модели данных, рассмотрим структуру памяти ЭВМ.

9.1. Структура памяти ЭВМ

Важнейшей особенностью памяти ЭВМ, в значительной степени определяющей методы организации данных и доступа к ним, является её неоднородность. Существуют два разных типа памяти – оперативная (ОП) и внешняя (ВП), причем процессор работает только с данными из оперативной памяти (рис. 9.1.).

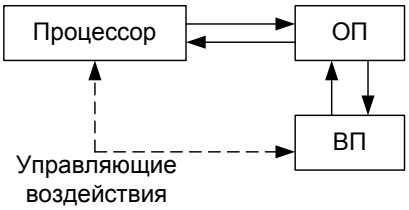


Рис. 9.1. Схема работы ЭВМ

Как уже многократно отмечалось, базы данных создаются для работы с большими объемами данных, что обусловливает необходимость использования внешней памяти. Поэтому организация данных и доступа к ним должна учитывать как специфику каждого типа памяти, так и способы их взаимодействия.

Отметим основные свойства оперативной памяти:

- единицей памяти является байт;
- память прямоадресуема (каждый байт имеет адрес);
- процессор выбирает для обработки нужные данные, непосредственно адресуясь к последовательности байтов, содержащих эти данные.

Отметим основные свойства внешней памяти:

- минимальной адресуемой единицей является физическая запись;
- для последующей обработки (например, работы с полями) запись должна быть считана в оперативную память;
- время чтения записи в ОП (занесения записи из ОП в ВП) на несколько порядков выше времени обработки процессором записи из ОП;
- организация обмена осуществляется порциями, т.к. невозможно считать сразу всю базу данных.

9.2. Представление экземпляра логической записи

Логическая запись представляется в оперативной памяти следующим образом:

Логическая запись

Поле 1	Поле 2	…	Поле *N*

→

Последовательность байтов ОП

*B*1	*B*2	...	B_N

Тип поля
Характеристика поля
Длина

B_i – последовательность байтов ОП, используемая для хранения поля *i*

Прямая адресация байтов позволяет процессору выбирать для обработки нужное поле.

Заметим, что указанное представление не делает различий для записей в сетевой, иерархической и реляционных моделях. В случае сетевой и иерархической моделей некоторые поля могут являться указателями, тогда последовательность байтов, используемая для хранения этих полей, содержит адрес начала последовательности байтов, соответствующей записи – члену отношения.

В большинстве современных СУБД используется формат записей фиксированной длины. В этом случае все записи имеют одинаковую длину, определяемую суммарной длиной полей, составляющих запись. В СУБД другие форматы записей (переменной длины, неопределенной длины) встречаются гораздо реже, поэтому в данной книге эти форматы не рассматриваются. Заметим, что поля записи, принимающие значения существенно разной длины в различных экземплярах записей, в предметной области встречаются достаточно часто. Примером может служить поле резюме в записи СОТРУДНИК. Резюме может составлять полстраницы текста, страницу и т.д. Возникает проблема – как эту информацию переменной длины представить в записи фиксированной длины. Возможным вариантом является установление размера соответствующего поля по максимальному значению. В этом случае у многих экземпляров записи указанное поле будет заполнено не полностью и, таким образом, память ЭВМ будет использоваться неэффективно. Более эффективный и часто используемый в СУБД прием организации таких записей состоит в следующем. Вместо поля (полей), принимающего значение существенно разной длины, в запись включается поле-указатель на область памяти, где будет размещаться значение исходного поля. Как правило, эта область является областью внешней памяти прямого доступа. В процессе ввода соответствующего значения в выделенной области занимается столько памяти, какова длина этого значения.

На рис. 9.2 представлен пример вышеуказанного представления экземпляров записей из *N* полей, причем поле *N* принимает значения соответственно разной длины у разных экземпляров записей.

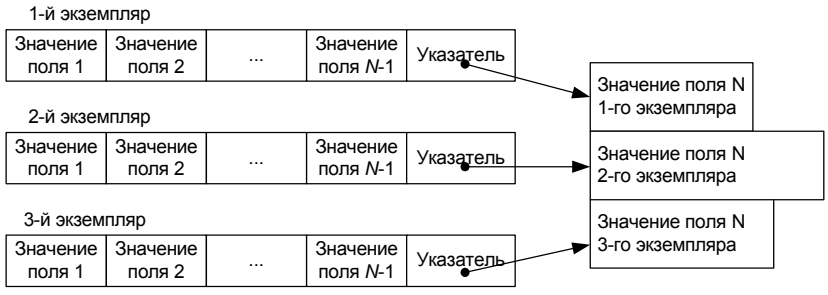


Рис. 9.2. Представление полей переменной длины

Конкретной реализацией такой схемы является поле типа МЕМО в СУБД (dBase III+, FoxPro, Access и т.д.).

9.3. Организация обмена между оперативной и внешней памятью

Единицей обмена данными между оперативной и внешней памятью является физическая запись. Физическая запись читается (записывается) за одно обращение к внешней памяти. В частности, физическая запись может соответствовать одному экземпляру логической записи. Число обращений к внешней памяти при работе с базой данных определяет время отклика системы. В связи с этим для уменьшения числа обращений к БД при работе с ней увеличивают длину физической записи (объединяют в одну физическую запись несколько экземпляров логических записей). В этом случае физическую запись называют также блоком, число k экземпляров логических записей, составляющих физическую запись, – коэффициентом блокировки.

Ввод исходных данных в БД осуществляется следующим образом:

- в ОП последовательно вводятся k экземпляров логических записей (кортежей);
- введенные k экземпляров объединяются в физическую запись (блок);
- физическая запись заносится во внешнюю память.

Ввод k экземпляров записей исходной таблицы, составляющих i-ю физическую запись, изображен на рис. 9.3.

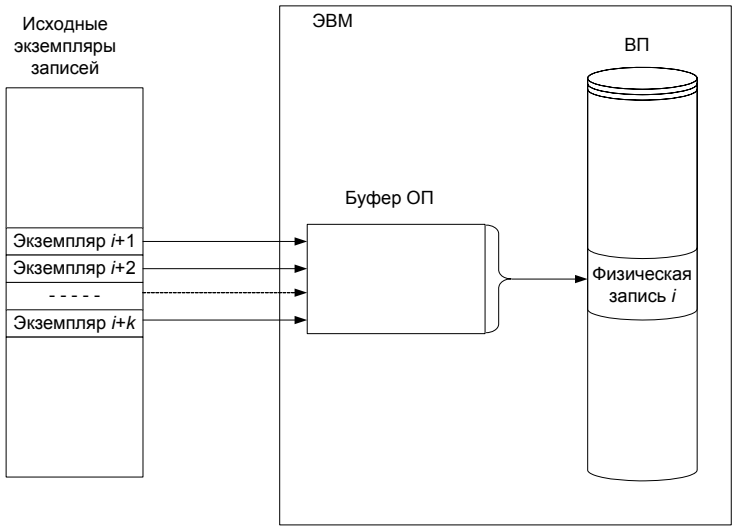


Рис. 9.3. Схема занесения записей во внешнюю память

Обработка данных, хранящихся во внешней памяти, осуществляется следующим образом:

- физическая запись (блок) считывается в оперативную память;
- обрабатываются экземпляры логических записей внутри блока (выбираются нужные поля, производится сравнение ключевого поля с заданным значением, осуществляется корректировка полей, выполняются операции удаления и т.п.).

В некоторых СУБД (например, MS SQL Server) единицей обмена между оперативной и внешней памятью является страница (вид физической записи, размер которой фиксирован и не зависит от длины логической записи). Организация обмена между оперативной и внешней памятью в этом случае аналогична описанной выше. Отличие здесь будет состоять в том, что экземпляры логических записей формируются в буфере, размером со

страницу)если сразмер страницы не кратен длине логической записи, страница может быть заполнена неполностью, физическая запись на внешнем носителе, соответственно, будет заполнена не полностью).

9.4. Структуры хранения данных во внешней памяти ЭВМ

В современных СУБД наибольшее распространение получили табличные модели данных. В связи с этим, а также для большей определенности в настоящем разделе мы будем говорить о структурах хранения для табличной модели. Однако отметим, что некоторые из рассматриваемых ниже структур хранения могут использоваться и для представления сетевых и иерархических моделей.

В качестве внешней памяти мы рассматриваем наиболее распространенную в современных ЭВМ память прямого доступа. Память прямого доступа дает возможность обращения к любой записи, если известен её адрес. Для упрощения изложения мы не будем конкретизировать ряд служебных полей, которые содержит физическая запись, и их рассмотрение опускаем.

9.4.1. Последовательное размещение физических записей

В этой структуре хранения записи в памяти размещаются последовательно друг за другом. Как уже отмечалось, считаем, что все записи имеют равную длину. Физический адрес записи может быть легко вычислен по номеру записи (для вычисления необходимо знать формат соответствующей физической записи).

Физическая запись с номером I содержит логические записи с номерами

$$(I-1)\,k+1$$
$$(I-1)\,k+2$$
$$\ldots$$
$$(I-1)\,k+k$$
$$I = 1, 2, \ldots, \lceil N/k \rceil;$$

знаком $\lceil N/k \rceil$ обозначим ближайшее целое, большее или равное N/k, – целое сверху.

Рассмотрим, как реализуются основные элементарные операции модели данных в этой структуре хранения, и оценим число этих операций. Напомним, что с точки зрения пользователя в табличной модели данных эти операции является операциями над строками (столбцами) таблицы.

Поиск записи с заданным значением ключа

При последовательной структуре хранения поиск может осуществляться только перебором. Читается первая физическая запись, в ОП она разбивается на k логических записей (разблокируется), заданное значение ключа сравнивается со значением ключа каждой логической записи. При несовпадении читается следующая физическая запись и процесс повторяется. В лучшем случае нужная запись будет найдена за одно обращение, в худшем – необходимо считать все физические записи. Среднее число обращений к внешней памяти для поиска нужной записи TP определяется следующей формулой

$$TP = (1+\lceil N/k \rceil)/2 ,$$

где N – число логических записей, k – коэффициент блокировки, $\lceil N/k \rceil$ – число физических записей.

Чтение записи с заданным значением ключа

Сначала необходимо найти нужную запись (смотри операцию «поиск»). После окончания операции «поиск» нужная запись уже считана в ОП. Число обращений к ВП равно TP.

Корректировка записи

Сначала необходимо найти нужную запись (смотри операцию «поиск»). После окончания операции «поиск» в ОП найденная логическая запись корректируется, формируется физическая запись (блок) и заносится во внешнюю память по тому адресу, откуда она была считана. Число обращений к ВП равно TP+1.

Удаление записи

Аналогична операции корректировки. Служебное поле соответствующей логической записи помечается как «удаленная запись». Число обращений к ВП равно TP+1.

Добавление записи

Рассмотрим два случая. В первом случае пользователь вводит новую логическую запись в конец таблицы. Тогда вводимая логическая запись добавляется в конец файла. Она заносится либо в последнюю физическую запись (если в ней меньше k логических записей – блок неполон), для чего эта запись должна быть считана в ОП, или формируется новая физическая запись, которая заносится в конец файла. Число обращений к ВП равно соответственно либо 2, либо 1.

Во втором случае пользователь вводит новую логическую запись в указываемую им i-ю строку таблицы (i=1, 2, …, n). В этом случае читается физическая запись с номером $\lceil (i-1)/k \rceil$, содержащая i-ю логическую запись. Если соответствующая физическая запись содержит пустые логические записи, то добавляемая запись вставляется в этот блок, блок записывается на свое место в ВП. Число обращений к ВП равно 2. Если указанная физическая запись содержит k экземпляров логических записей исходной таблицы, читается физическая запись с номером $\lceil i/k \rceil$. Если эта физическая запись содержит пустые логические записи, добавляемая запись вставляется в этот блок, блок записывается на свое место в ВП. Суммарное число обращений в этом случае будет на единицу больше и равно 3.

Если физические записи с номерами $\lceil (i-1)/k \rceil$ и $\lceil i/k \rceil$ содержат по k экземпляров исходных логических записей, необходимо формировать дополнительную физическую запись. Соответствующий блок будет содержать добавляемую логическую запись и k-1 пустых логических записей. Блоки с номерами $\lceil i/k \rceil$, $\lceil (i+1)/k \rceil$, … $\lceil N/k \rceil$ переписываются на одну позицию ниже (сдвигаются). Сформированная физическая запись заносится на освободившееся место (место записи с номером $\lceil i/k \rceil$).

В лучшем случае (i = N) ни один блок не сдвигается. В худшем случае (i = 1) сдвигаются все блоки. Среднее число обращений к ВП для перезаписи блоков (чтение + запись) соста-

вит $2\lceil N/k \rceil/2$. Тогда суммарное число обращений к ВП при добавлении записи в этом случае будет равно $3+\lceil N/k \rceil$.

Заметим, что если записи упорядочены по значениям ключа может производиться дихотомическим методом и число обращений к внешней памяти будет пропорционально не $(1+\lceil N/k \rceil)/2$, а $log_2\lceil N/k \rceil$, т.е. существенно меньше. Однако добавление записи потребует для сохранения упорядоченности, как правило, сдвига большого числа записей. Поэтому размещение физических записей с упорядочением их по значениям ключа в СУБД не используется.

9.4.2. Размещение физических записей в виде списковой структуры

Основная проблема в использовании изложенного в п. 9.4.1 способа организации записей состоит в отображении добавления логической записи в произвольное место таблицы. При этом приходится переписывать в памяти (сдвигать на одну позицию) физические записи, соответствующие логическим записям таблицы, расположенным ниже места вставки добавляемой строки. Соответствующую проблему можно устранить, используя для представления физических записей связный список (рис. 9.4.).

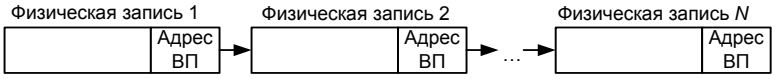


Рис. 9.4. Список физических записей

Кроме этого списка в ВП формируется список свободных элементов («пустых» физических записей), элементы которого используются при вводе новой записи с данными (рис. 9.5).

Напомним, что каждая физическая запись состоит, как и ранее, из k логических записей.

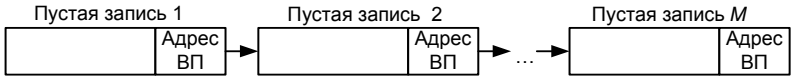


Рис. 9.5. Список свободных элементов

Рассмотрим, как реализуются основные элементарные операции модели данных в этой структуре хранения.

Поиск записи с заданным значением ключа

Заметим, что упорядочение записей по значениям ключа не дает здесь ускорения процедуры поиска. Это связано с тем, что после ряда добавлений новых записей и удаления каких-то имеющихся записей физическая и логическая последовательность записей в списке будут существенно различаться. При этом будет невозможно по номеру записи определить ее адрес и обращаться к записи, соответствующей середине таблицы, для реализации дихотомического метода поиска. Поэтому поиск можно вести только с помощью перебора. В ОП читается первая запись списка, разблокируется, значения ключевых полей логических записей этой физической записи сравниваются с заданным значением. Если значения совпали,

нужная запись найдена, если не совпали, из записи выбирается адрес следующей записи списка, читается эта запись. Далее процедура повторяется. Среднее число обращений к ВП будет равно, как и в 5.4.1, $(1+\lceil N/k \rceil) / 2$.

Чтение записи

После завершения предыдущей операции запись считана в ОП. Оценка числа обращений к ВП та же.

Корректировка записи

Считанная запись корректируется и заносится в ВП на свое место (по своему адресу). Число обращений к ВП на единицу больше, чем при чтении.

Удаление записи

Заметим, что мы говорим об операциях над логическими записями. Операция удаления логической записи аналогична операции корректировки. Служебное поле соответствующей логической записи помечается как «удаленная запись». Сформированная физическая запись заносится в ВП. Число обращений к ВП равно *TP*+1.

Добавление записи

Для определенности будем считать, что задан ключ логической записи, после которой должна быть добавлена новая запись. Осуществляется операция поиска и чтения физической записи, в которой расположена запись с ключом *PK*. Если в этом блоке есть логическая запись, помеченная как удаленная, добавляемая запись заносится на ее место. Блок записывается в ВП. Число обращений к ВП равно TP+1. Если в этом блоке нет логических записей, помеченных как удаленные, необходимо добавлять новую физическую запись, выбираемую из списка свободных элементов. С этой целью адрес связи найденной ранее физической записи заменяется на адрес начала списка свободных элементов.

Читается первая физическая запись списка свободных элементов. Адрес связи этой записи заменяет адрес начала пустого списка. В ОП формируется новая физическая запись, содержащая добавляемую логическую запись. В качестве ее адреса связи заносится адрес связи из физической записи, предшествующей добавляемой. Каждая из этих записей заносится в ВП. Число обращений к ВП при добавлении записи будет примерно равно *TP*+3.

Рассмотренный метод организации структуры хранения достаточно эффективно решает проблемы добавления и удаления записей, но не уходит от перебора при поиске нужной записи.

9.4.3. Использование индексов (индексирование)

Как уже отмечалось, упорядочение записей позволяет использовать дихотомический метод поиска нужной записи и тем самым существенно сократить одну из основных составляющих времени поиска – число обращений к ВП. Однако при этом возникают проблемы с добавлением записей, связанные с необходимостью перезаписи части физических записей (сдвига).

Для того чтобы использовать дихотомический поиск и не перемещать физические записи при добавлении новых записей, используется так называемое логическое упорядочение физических записей (индексирование). Основная структура хранения содержит запи-

си исходной таблицы и представлена в виде неупорядоченной последовательности физических записей (см. п. 5.4.1). ***Для возможной реализации дихотомического поиска по определенному ключу создается дополнительная структура хранения (так называемый индекс)***. Число записей в индексе равно числу записей исходной таблицы (числу физических записей в основной структуре хранения). Каждая запись индекса имеет два поля: ключевое поле записи основной структуры и указатель – адрес записи основной структуры с соответствующим значением ключа.

Записи индекса (индексного файла) упорядочены по значению ключа. Адреса связи этих записей определяют логическое упорядочение записей основной структуры хранения. Пример соответствующей структуры хранения приводится в предположении k=1на рис. 9.6.

Рассматриваемую структуру хранения называют еще инвертированным списком. Смысл этого термина состоит в следующем. Можно было бы упорядочить записи основной структуры хранения, не переставляя их, а объединив в соответствующий упорядоченный список. В нашем случае адреса связи как бы удаляются из списка и включаются в состав файла-индекса (инвертируются). Поэтому полученная структура интерпретируется как инвертированный список.

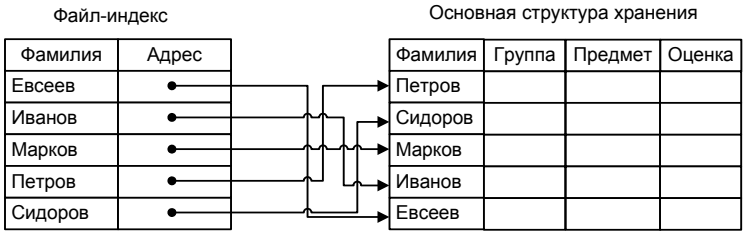


Рис. 9.6. Индексирование

Поиск нужной записи по заданному значению ключа осуществляется в индексном файле методом половинного деления. Заметим, что так как записи индекса содержат всего два поля, суммарный объем записей индекса невелик, поэтому индекс, как правило, целиком считывается для обработки в ОП за одно обращение к ВП. После того как в индексном файле обнаружена искомая запись, по адресу связи читается полная соответствующая запись основной структуры хранения. Если необходим поиск по другому ключу, строится еще один индекс по соответствующему ключу. Таким образом, по любому ключу поиск можно осуществлять дихотомическим методом.

Оценим число обращений к ВП при реализации элементарных операций. Соответствующие оценки сделаны для случая, когда физическая запись состоит из одной логической записи (коэффициент блокировки k равен 1). Расчет оценок для произвольного k производится по аналогии с расчетами пп. 9.4.1–9.4.2.

Поиск записи с заданным значением ключа

Из ВП читается индексный файл (число обращений к ВП для этого зависит от объема индексного файла, как правило, невелико и много меньше числа записей N). После нахожде-

ния нужной записи в индексном файле читается соответствующая запись основного файла (одно обращение к ВП).

Чтение записи

В ходе операции поиска искомая запись считана в ОП.

Корректировка записи

Считанная запись корректируется и заносится на свое место (еще одно обращение к ВП).

Удаление записи

Найденная запись помечается как удаленная в основном файле, соответствующая запись в индексном файле удаляется, измененный индекс записывается в ВП. Число обращений к ВП в этом случае по сравнению с числом обращений к ВП при поиске увеличивается на два.

Добавление записи

Добавляемая запись заносится в конец основного файла. Формируется новая запись индекса, соответствующая добавляемой записи. Записи индекса переупорядочиваются по значениям ключа, и индекс заносится в ВП. Число обращений к ВП в этом случае, в основном, определяется чтением-записью индекса.

Таким образом, использование индексов позволяет ценой некоторого увеличения объема используемой памяти (за счет индекса) существенно сократить время реализации основных операций. В связи с этим индексирование используется во многих современных СУБД.

9.4.4. B-дерево

Структура B-дерева (сбалансированное дерево) является следствием дальнейшего расширения концепции использования индексов (строится индекс над индексом) и представляет собой многоуровневые индексы.

B--дерево строится следующим образом. Последовательность записей, соответствующая записям исходной таблицы, упорядочивается по значениям первичного ключа. Логические записи объединяются в блоки (по k записей в блоках).

Значением ключа блока является минимальное значение ключа у записей, входящих в блок. Последовательность блоков представляет собой последний уровень B-дерева. Строится индекс предыдущего уровня. Записи этого уровня содержат значение ключа блока следующего уровня и указатель-адрес связи соответствующего блока; записи этого уровня также объединяются в блоки (по k записей). Затем аналогично строится индекс более высокого уровня и т.д., пока количество записей индекса на определенном уровне будет не более k.

Рассмотрим процедуру работы с B-деервом на примере. Пусть имеется файл экземпляров логических записей, ключи которых принимают значения 2, 7, 8, 12, 15, 27, 28, 40, 43, 50. Для определенности возьмем k=2 (в блок объединяем по 2 экземпляра записей). Построенное для этого примера B-дерево изображено на рис. 9.7 (для упрощения рисунка на уровне 4 представлены только ключи логических записей и не представлены значения других полей этих записей).

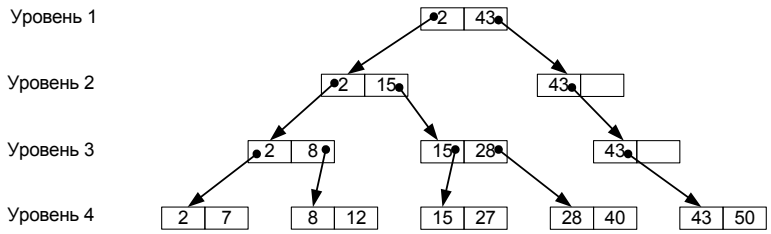


Рис. 9.7. B-дерево

В блоках указано значение ключа соответствующего блока. Значение k принято равным 2.

По построению B-дерева все исходные записи находятся на одном расстоянии от верхнего индекса (дерево является сбалансированным).

Рассмотрим реализацию основных операций.

Поиск и чтение записи с заданным значением ключа

Читается верхний индекс. Сравниваем заданное значение ключа со значением ключа записей индекса. Если заданное значение ключа больше, чем значение ключа очередной записи индекса (если такая запись имеется), или равно ему, то по адресу связи, указанному в текущей записи, читается блок записей индекса следующего уровня. Далее процесс повторяется.

Считаем, что все блоки расположены в ВП. Тогда число обращений к ВП при поиске информации будет равно числу уровней дерева. Число уровней дерева равно минимальному значению l, при котором выполняется условие $k^l \geq N$ (N – число логических записей).

Модификация (корректировка) записи

После поиска и чтения записи изменяются корректируемые поля. Если корректируется не ключ записи, то измененная запись заносится на свое место. Если изменено значение ключа, то старая запись удаляется (в соответствующем блоке появляется «пустая» запись), а измененная запись заносится так же, как вновь добавляемая.

Удаление записи

После поиска найденная запись удаляется (в соответствующий блок на место этой записи заносится «пустая» запись).

Добавление записи

Прежде всего определяется, где должна быть расположена добавляемая запись с заданным значением ключа. Процедура поиска блока, где должна быть расположена эта запись, аналогична вышеописанной процедуре поиска записей с заданным значением ключа. Если в найденном блоке низшего уровня есть «пустая» запись, добавляемая запись заносится в этот блок (с необходимым переупорядочением записей внутри блока).

Если в соответствующем блоке низшего уровня нет пустого места, блок делится на два блока. В первый из них заносится $[k/2]$ записей, во второй заносятся остальные. Значением ключа каждого из указанных блоков будет являться, как и описано ранее, минимальное значение ключей у записей, входящих в блок. Добавляемая запись заносится в тот блок, значение ключа которого меньше значения ключа добавляемой записи. Появление нового блока с

новым значением ключа обусловливает необходимость формирования соответствующей новой записи в индексе на предыдущем уровне. Эта запись содержит новое значение ключа нового блока и указатель на его месторасположение. Процедура добавления такой записи аналогична описанной выше. Находится блок предыдущего уровня, куда должна быть помещена эта запись. Если в блоке есть пустое место, запись добавляется в блок, если блок полон, он делится на два блока, запись заносится в один из блоков, формируется запись индекса предыдущего уровня и т.д.

Возможен вариант, когда придется делить блок самого верхнего уровня и формировать еще один уровень дерева.

Рассмотрим для примера, изображенного на рис. 27, добавление записи с ключом 10.

1. Сравнение на первом уровне.

2<10<43

Движение по левой ветви.

2. Сравнение на втором уровне.

2<10<15

Движение по левой ветви.

3. Сравнение на третьем уровне.

2<8<10

Движение по правой ветви.

Искомый блок | 8 | 12 |

4. Блок заполнен.

Он делится на 2 блока

| 8 | | | 12 | |

Сравнение 8<10<12.

Запись с ключом 10 заносится в блок 1

| 8 | 10 | | 12 | |

На низшем уровне появилась новая запись с значением ключа 12. Необходимо добавление новой записи с ключом 12 и указателем на запись низшего уровня к индексу предыдущего уровня.

5. Запись с ключом 12 уровня 3 должна добавляться в блок | 2 | 8 |. Блок полон, он делится на два блока

| 2 | | | 8 | |

Сравнение 8<12.

Запись добавляется во второй блок | 8 | 12 |

6. На уровне 3 появился блок с новым ключом 8. Необходимо добавление новой записи с ключом 8 и указателем на соответствующий блок уровня 3 на уровне 2.
7. Запись с ключом 8 уровня 2 должна добавиться в блок | 2 | 15 |. Блок полон, он делится на два блока.

| 2 | | | 15 | |

2<8<15

Запись добавляется в блок 1 | 2 | 8 |.

8. На уровне 2 появился блок с новым ключом 15, необходимо добавление новой записи с ключом 15 и указателем на соответствующий блок уровня 2 на уровне 1.

9. Запись с ключом 15 уровня 1 должна добавляться в блок [2 | 43]. Блок полон, он делится на два блока.

[2 |] [43 |]

2<15<43

Запись с ключом 15 добавляется в первый блок

[2 | 15] [43 |]

10. Необходимо сформировать еще один уровень дерева [2 | 43].

Полученная структура будет иметь вид, представленный на рисунке 9.8.

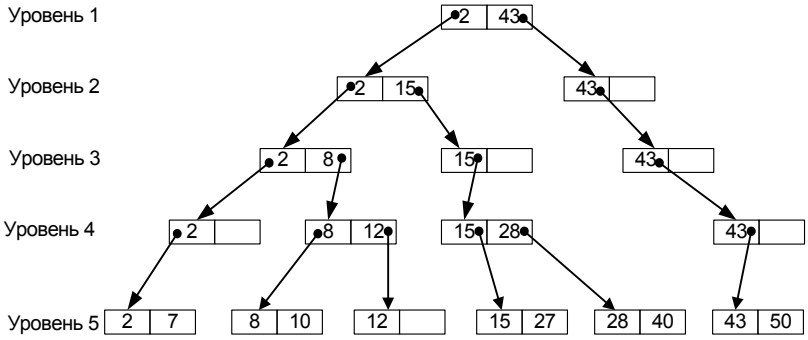


Рис. 9.8. B-дерево после добавления элемента

Необходимо заметить, что используемый прием деления пополам полностью заполненного блока при добавлении в него записи приведет к тому, что блоки будут заполнены, в среднем, наполовину. Тогда процедура добавления записи будет существенно менее трудоемкой (если в нужном блоке есть место, запись добавляется в этот блок и вышестоящие уровни не перестраиваются).

Структура хранения в виде B-дерева позволяет эффективно проводить операции поиска, чтения, удаления, модификации с оценкой числа обращений к внешней памяти числом уровней дерева l (l ≈ logkN), что существенно меньше числа обращений при переборе $\lceil N/k \rceil$.

Процедура добавления записи тоже достаточно эффективна. Соответствующая структура хранения, в частности, используется в отечественной СУБД НИКА (ранее использовалась в системе ИНЕС) и на реальных задачах показала высокую эффективность.

9.4.5. Размещение записей с использованием хэширования

Как в любом другом способе организации структур хранения, логические записи группируются в физические записи (блоки) по *k* штук. Однако в отличие от всех других способов организации структур хранения здесь выбран особенный способ группировки. ***Определенным образом выбирается так называемая хэш-функция f. Аргументом этой функции является значение x первичного ключа логической записи. Тогда f(x) указывает адрес расположения блока, в котором должна находиться логическая запись со значением ключа x.***

Функция *f* должна, по возможности, равномерно распределять значения *x* по физическим блокам. Обсуждению возможных хэш-функций посвящено достаточно много литературы, поэтому здесь мы не будем касаться этого вопроса. Можно лишь добавить, что иногда, исходя из специфики множества значений *x* первичного ключа, можно построить функцию *f*,

удовлетворяющую всем необходимым условиям. Таким образом, логическая запись таблицы со значением x первичного ключа размещается в блоке внешней памяти по адресу *f(x)*. В этом блоке может находиться не более *k* записей. Может оказаться, что выбранная функция отображает в один адрес памяти (один блок) более *k* записей. Возникает так называемая коллизия. Возможным способом разрешения коллизий является использование дополнительной области переполнения следующим образом. Если очередная запись распределяется с помощью функции хэширования в блок, а он полностью заполнен, то в области переполнения формируется список записей, соответствующих этому блоку, с включением в него указанной записи, а в сам блок заносится указатель – адрес связи на первую запись этого списка. Возможны и другие способы разрешения коллизий.

Рассмотрим реализацию основных операций и дадим оценку числа обращений к ВП при их выполнении.

Поиск записи с заданным значением ключа и чтение

По заданному значению ключа x подсчитывается значение функции f(x). Далее из ВП считывается блок, находящийся по адресу f(x). В ОП внутри этого блока перебором ищется нужная запись. Если записей в блоке нет, то по указателю в блоке (адресу связи) читается первая запись списка переполнения, относящаяся к этому блоку. Далее необходимая запись ищется по этому списку. Число обращений к ВП при этом равно:

- единице, если запись находится в блоке;
- единице плюс число записей в соответствующем этому блоку списке области переполнения (как правило, небольшое число).

Модификации записи

Осуществляется поиск и чтение записи, затем в ОП модифицируются поля записи (не являющиеся первичным ключом), запись заносится на свое место. Число обращений к ВП в этом случае на единицу больше, чем при чтении записи. Если модифицируется значение ключа, то занесение записи осуществляется как ввод новой записи (добавление).

Удаление записи

Осуществляется поиск и чтение записи. Если удаляемая запись находилась в блоке основной памяти, на ее место заносится «пустая» запись (или признак «пустой» записи). Если удаляемая запись находилась в списке области переполнения, удаление ее производится по правилам удаления элемента списка. Число обращений к ВП при удалении находится примерно в тех же пределах, что и для предыдущих операций.

Добавление записи

При добавлении записи со значением ключа x подсчитывается адрес соответствующего блока *f(x)*. Блок считывается в ОП. Если в нем есть место, запись заносится в блок, блок записывается в ВП по своему адресу. Если блок заполнен, из него выбирается адрес начала списка записей, переполняющих блок. Далее добавление записи в список производится по правилам добавления элемента в список. Число обращений к ВП при добавлении записей находится примерно в тех же пределах, что и для предыдущих операций.

Таким образом, описанная структура хранения с использованием хэширования является наиболее эффективной (из рассмотренных выше) по критерию минимизации числа обращений к ВП при реализации основных операций.

9.4.6. Комбинированные структуры хранения

Необходимо заметить, что в СУБД могут использоваться как каждая из вышерассмотренных структур в отдельности, так и их комбинация. Так, например, в ряде промышленных систем UNIBAD, БАНК для ЭВМ типа IBM 360/370 (ЕС ЭВМ), PARADOX для персональных ЭВМ ***используются следующие комбинации методов:***

- ***размещение записей по первичному ключу организовано с использованием хэширования;***
- ***последовательность записей по вторичному ключу задается с помощью списковой структуры.***

Краткие итоги: Лекция посвящена вопросам физической организации данных в памяти компьютера (организации структур хранения). Физические модели представления данных жестко заложены в структуру конкретной СУБД и различны в различных системах управления базами данных. Заметим, что в данной лекции рассматриваются не структуры хранения конкретной СУБД, а некоторые типовые структуры хранения, на основе которых и реализуются физические модели организации данных в конкретных СУБД. Здесь описывается двухуровневая структура памяти компьютера как среда размещения данных; организация обмена между внешней и оперативной памятью, определяющая специфику обработки данных. Представлены типовые физические модели (структуры хранения данных) во внешней памяти ЭВМ (последовательное размещение физических записей, размещение физических записей в виде списковой структуры, использование индексов, организация данных в виде B-дерева, размещение записей с использованием хэширования, атакже комбинированные структуры хранения). Для основных структур хранения сделана оценка числа действий при выполнении операций поиска данных, чтения, занесения данных, модификации (корректировки), удаления.

Более подробно с материалами этой лекции можно ознакомиться в [1-7].

Контрольные тесты.

Задача 1. Общая характеристика внутреннего уровня базы данных.

Вариант 1.
Что такое внутренний уровень базы данных?

☐ концептуальное представление
☐ концептуальная модель, специфицированная в терминах СУБД
⊞ структура хранения данных в памяти компьютера
⊞ отображение концептуальной модели базы данных в физическую организацию данных

Вариант 2.
Что такое физическая модель данных?

⊞ внутренний уровень базы данных
☐ концептуальная модель, специфицированная в терминах СУБД
☐ структура памяти компьютера
⊞ отображение концептуальной модели базы данных в физическую организацию данных.

Вариант 3.
В каком виде концептуальная модель базы данных представляется в памяти компьютера?

☐ в виде модели данных
⊞ в виде физической модели
⊞ в виде структуры хранения
⊞ как наборы физических записей

Задача 2. Представление экземпляра логической записи

Вариант 1.
В каком виде представляется экземпляр логической записи?

- ☐ линейная последовательность байтов переменной длины
- ⊞ линейная последовательность байтов фиксированной длины
- ☐ линейная последовательность байтов фиксированной длины с возможным указателем на другую область памяти
- ☐ линейная последовательность байтов переменной длины с возможным указателем на другую область памяти

Вариант 2.
Какие параметры характеризуют поле логической записи при его физическом представлении?

- ⊞ количество занимаемых байтов
- ⊞ тип представления данных
- ☐ наименование поля
- ☐ количество символов в значении поля

Вариант 3.
Какие параметры поля логической записи не являются характеристиками его физическом представлении?

- ☐ количество занимаемых байтов
- ☐ тип представления данных
- ⊞ наименование поля
- ⊞ количество символов в значении поля

Задача 3. Организация обмена между оперативной и внешней памятью.

Вариант 1.
Что является единицей обмена между внешней и оперативной памятью?

☐ экземпляр логической записи
☐ логический файл
☐ физический файл
⊞ физическая запись
⊞ страница

Вариант 2.
Почему обмен между оперативной и внешней памятью осуществляется страницами или физическими записями?

⊞ для сокращения времени обработки
☐ для сокращения занимаемого объема оперативной памяти
☐ для сокращения занимаемого объема внешней памяти
⊞ для сокращения числа обращений к внешней памяти

Вариант 3.
Почему обмен между оперативной и внешней памятью нецелесообразно осуществлять отдельными экземплярами логических записей?

⊞ затрачивается большое время на обработку данных
☐ используется чрезмерно много оперативной памяти
☐ используется чрезмерно много внешней памяти
☐ трудно осуществлять поиск необходимых данных

Задача 4. Последовательное размещение физических записей во внешней памяти.

Вариант 1.
Как осуществляется поиск записи с заданным значением ключа при последовательном размещении физических записей во внешней памяти?

⊞ полным перебором
☐ по заданному адресу
☐ дихотомическим методом
☐ чтением записи с заданным значением ключа

Вариант 2.
Какой формулой оценивается среднее число обращений к внешней памяти при поиске записи с заданным значением ключа при последовательном размещении физических записей во внешней памяти (N - число экземпляров логических записей, k - коэффициент блокировки)?

⊞ $(1+\lceil N/k \rceil)/2$
☐ N
☐ $log_2(1+\lceil N/k \rceil)/2$
☐ $\lceil (1+N)/k \rceil / 2$

Вариант 3.
Когда при добавлении новой физической записи при последовательном размещении физических записей во внешней памяти требуется затратить меньше действий?

⊞ при добавлении в конец физического файла
☐ при вставке в нужное место физического файла
☐ при вставке в начало физического файла
☐ при добавлении новой физической записи на место удаляемой физической записи.

Задача 5. Размещение физических записей в виде списковой структуры.

Вариант 1.
Как осуществляется поиск записи с заданным значением ключа при размещении физических записей в виде списковой структуры?

⊞ полным перебором
☐ по заданному адресу
☐ дихотомическим методом
☐ чтением записи с заданным значением ключа

Вариант 2.
Какой формулой оценивается среднее число обращений к внешней памяти при поиске записи с заданным значением ключа при размещении физических записей в виде списковой структуры? (*N* число экземпляров логических записей, *k* коэффициент блокировки)?

⊞ $(1 + \lceil N/k \rceil)/2$
☐ N
☐ $log_2(1+\lceil N/k \rceil)/2$
☐ $log_2 N/k$

Вариант 3.
Как примерно соотносится объем затрачиваемых действий при добавлении новой физической записи при размещении физических записей в виде списковой структуры?

☐ меньше при добавлении в конец физического файла
☐ больше при вставке в нужное место физического файла
☐ меньше при вставке в начало физического файла
☐ больше при добавлении новой физической записи на место удаляемой физической записи
⊞ примерно равны

Задача 6. Использование индексов

Вариант 1.
Как хранятся физические записи в памяти при использовании индексации?

☐ упорядочены по значениям ключа
⊞ в виде неупорядоченной последовательности
☐ в виде списка
⊞ используется дополнительный файл

Вариант 2.
Что такое индекс?

⊞ дополнительная таблица
☐ адрес связи у физической записи основного файла
☐ B-дерево
☐ Хэш-функция

Вариант 3.
К чему приводит использование индекса?

⊞ к сокращению времени поиска
☐ к сокращению времени добавления записи
⊞ к сокращению числа обменов между оперативной и внешней памятью
⊞ к увеличению объема занимаемой памяти
⊞ к дублированию информации

Задача 7. Использование B-дерева.

Вариант 1.
Из каких полей состоит запись всех уровней B-дерева, кроме нижнего?

⊞ из поля ключа и поля ссылки на нижележащий уровень
☐ из поля ключа и поля ссылки на вышележащий уровень
☐ из полей логической записи и поля ссылки на нижележащий уровень
☐ из полей логической записи и поля ссылки на вышележащий уровень

Вариант 2.
Что происходит при добавлении записи в B-дерево?

⊞ может увеличиться число блоков нижнего уровня
⊞ может увеличиться число блоков всех уровней
⊞ может увеличиться число уровней
☐ структура дерева не меняется

Вариант 3.
К чему приводит использование B-дерева?

⊞ к сокращению времени поиска
☐ к сокращению времени добавления записи
⊞ к сокращению числа обменов между оперативной и внешней памятью
⊞ к увеличению объема занимаемой памяти
⊞ к дублированию информации

Задача 8. Размещение физических записей с использованием хэширования

Вариант 1. Как осуществляется поиск записи с заданным значением ключа при размещении физических записей с использованием хэширования?

☐ полным перебором
⊞ по вычисленному адресу
☐ дихотомическим методом
☐ чтением записи с заданным значением ключа

Вариант 2. Как примерно оценивается среднее число обращений к внешней памяти при поиске записи с заданным значением ключа при размещении физических записей с использованием хэширования? (*N* - число экземпляров логических записей)?

☐ пропорционально *N*
⊞ небольшое число
☐ пропорционально $\log_2 N$
☐ как некоторая функция *f(N)*

Вариант 3. Как примерно соотносится объем затрачиваемых действий при добавлении новой физической записи при размещении физических записей с использованием хэширования?

☐ меньше при добавлении в конец физического файла
☐ больше при вставке в нужное место физического файла
☐ меньше при вставке в начало физического файла
☐ больше при добавлении новой физической записи на место удаляемой физической записи
⊞ примерно равны

Литература

1. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Щерса. – М.: Мир, 1980. – 664 с.
2. Дейт К.Дж. Введение в системы баз данных: Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с
3. Хансен Г., Хансен Дж. Базы данных: разработка и управление: Пер. с англ. – М.: ЗАО «Издательство «БИНОМ», 1999. – 704 с.
4. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с
5. Карпова Т. Базы данных. Модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.
6. Крёнке Д. Теория и практика построения баз данных. 8-е изд. – СПб.: Питер, 2003. – 800 с.
7. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Н.Новгород: Изд-во ННГУ, 2004. 271 с.

Лекция 10. Структура современной СУБД на примере Microsoft SQL Server 2008

В лекции рассматривается архитектура системы управления базами данных на примере одной из наиболее распространенных клиент-серверных СУБД - Microsoft SQL Server 2008 (логический и физический уровни).

Ключевые термины: структура СУБД, архитектура СУБД, логический уровень, архитектура СУБД, физический уровень СУБД, архитектура Microsoft SQL Server 2008.

Цель лекции: показать основные элементы структуры современной СУБД (архитектуры базы данных и структуры программного обеспечения) на примере Microsoft SQL Server 2008.

10.1 Общая структура СУБД

Для лучшего понимания принципов работы современных СУБД рассмотрим структуру одной из наиболее распространенных клиент-серверных СУБД - Microsoft SQL Server 2008. Несмотря на то, что каждая коммерческая СУБД имеет свои отличительные особенности, информации о том, как устроена какая-то из СУБД, обычно бывает достаточно для быстрого первоначального освоения другой СУБД. Краткий обзор возможностей Microsoft SQL Server - 2008 был приведен в разделе, посвященном краткому обзору современным СУБД. В данном разделе рассмотрим основные моменты, связанные со структурой соответствующей СУБД (архитектурой базы данных и структурой программного обеспечения)

Под архитектурой (структурой) базы данных конкретной СУБД будем понимать основные модели представления данных, используемые в соответствующей СУБД а также взаимосвязи между этими моделями.

В соответствии с рассмотренными в лекции 3 различными уровнями описания данных различают разные уровни абстракции архитектуры базы данных

Логический уровень (уровень модели данных СУБД) - средство представления концептуальной модели). Здесь каждая СУБД имеет некоторые отличия, но они являются не очень значительными. Отметим, что у разных СУБД существенно отличаются механизмы перехода от логического к физическому уровню представления.

Физический уровень (внутреннее представление данных в памяти ЭВМ - физическая структура базы данных). Данный уровень рассмотрения подразумевает изучение базы данных на уровне файлов, хранящихся на жестком диске. Структура этих файлов – особенность каждой конкретной СУБД, в т.ч. и Microsoft SQL Server.

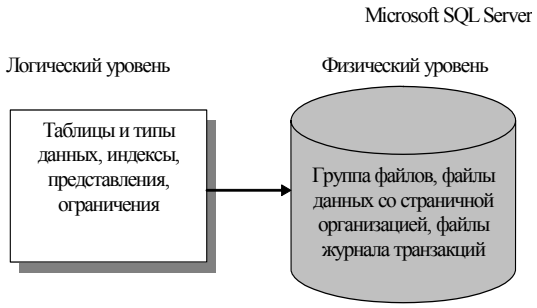


Рис. 10.1. Архитектура базы данных в Microsoft SQL Server 2008

10.2. Архитектура базы данных. Логический уровень

Рассмотрим логический уровень представления базы данных (http://msdn.microsoft.com). Microsoft SQL Server 2008 представляет собой реляционную СУБД (данные представляются в виде таблиц). Таким образом, основной структурой модели данных этой СУБД являются таблицы.

Таблицы и типы данных

Таблицы содержат данные о всех сущностях концептуальной модели базы данных. При описании каждого столбца (поля) пользователь должен определить тип соответствующих данных. Microsoft SQL Server 2008 поддерживает как уже ставшие традиционными типы данных (символьная строка с разным представлением, число с плавающей точкой длиной 8 или 4 байта, целое число длины 2 или 4 байта, дата и время, поле примечаний, булево значение и т. д.), так и новые типы данных. Кроме этого Microsoft SQL Server 2008 предоставляет специальный аппарат для создания пользовательских типов данных.

Рассмотрим краткую характеристику некоторых новых типов данных, значительно расширяющих возможности пользователя (www.oszone.net).

Тип данных hierarchyid

Тип данных hierarchyid позволяет создавать отношения между элементами данных в таблице, для того, чтобы задать позицию в иерархии связей между строками таблицы. В результате использования этого типа данных в таблице строки таблицы могут отображать определенную иерархическую структуру, соответствующую связям между данными этой таблицы.

Пространственные типы данных

Пространственные данные – это данные, определяющие географические расположения и формы, преимущественно на Земле. Это могут быть ориентиры, дороги и даже расположение фирмы. В SQL Server 2008 есть географические (geography) и геометрические (geometry) типы данных для работы с этой информацией. Тип данных ***geography*** работает с инфорацией для шарообразной земли. Модель шарообразной земли использует при расчетах кривизну земной поверхности. Информация о положении задается широтой и долготой. Эта модель хорошо годится для приложений, связанных с морскими перевозками, военным планирова-

нием и краткосрочными приложениями, имеющими привязку к земной поверхности. Эту модель нужно использовать, если данные хранятся в виде широт и долгот.

Тип данных **geometry** работает с планарной моделью или моделью плоской земли. В этой модели земля считается плоской проекцией из определенной точки. Модель плоской земли не принимает в расчет кривизну поверхности земли, поэтому используется, в первую очередь, для описания коротких расстояний, например, в базе данных приложения, описывающего внутреннюю часть строения.

Типы **geography** и **geometry** создаются из векторных объектов, заданных в форматах Well-Known Text (WKT) или Well-Known Binary (WKB). Это форматы для перенесения пространственных данных, описанные в простых функциях открытого геопространственного консорциума (Open Geospatial Consortium (OGC) Simple Features) для спецификаций SQL (SQL Specification).

Ключи

Для каждой таблицы должен быть определен ***первичный ключ – минимальный набор атрибутов, уникально идентифицирующий каждую запись в таблице***. Для реализации связи между таблицами в одну из связанных таблиц включается дополнительное поле (несколько полей) – первичный ключ другой таблицы. Дополнительно включенные поле или поля в этом случае называются внешним ключом соответствующей таблицы.

Кроме таблиц, в модель данных Microsoft SQL Server 2008 входит еще целый ряд компонентов. Дадим краткую характеристику основным из них.

Индексы

В лекции 8 рассматривалось понятие индекса. Здесь понятие индекса вынесено на логический уровень для удобства пользователя. ***Индексы создаются для ускорения поиска нужной информации и содержат информацию об упорядоченности данных по различным критериям.*** Индексирование может быть выполнено по одному или нескольким столбцам. Индексирование может быть произведено в любой момент. Индекс содержит ключи, построенные из одного или нескольких столбцов в таблице или представлении. Эти ключи хранятся в виде структуры сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям в SQL Server.

Представления

Представление — это виртуальная таблица, содержимое которой определяется запросом. Представление формируется на основе SQL-запроса SELECT, формируемого по обычным правилам. Таким образом, представление есть поименованный запрос SELECT.

Как и настоящая таблица, представление состоит из совокупности именованных столбцов и строк данных. Пока представление не будет проиндексировано, оно не существует в базе данных как хранимая совокупность значений. Строки и столбцы данных извлекаются из таблиц, указанных в определяющем представление запросе и динамически создаваемых при обращениях к представлению. Представление выполняет функцию фильтра базовых таблиц, на которые оно ссылается. Определяющий представление запрос может быть инициирован в одной или нескольких таблицах или в других представлениях текущей или других баз дан-

ных. Кроме того, для определения представлений с данными из нескольких разнородных источников можно использовать распределенные запросы. Это полезно, например, если нужно объединить структурированные подобным образом данные, относящиеся к разным серверам, каждый из которых хранит данные конкретного отдела организации.

Сборки

Сборки являются файлами динамической библиотеки, которые используются в экземпляре SQL Server для развертывания функций, хранимых процедур, триггеров, определяемых пользователем статистических вычислений и определяемых пользователем типов.

Ограничения

Ограничения позволяют задать метод, с помощью которого компонент СУБД Database Engine автоматически обеспечивает целостность базы данных. Ограничения задают правила допустимости определенных значений в столбцах и представляют собой стандартный механизм обеспечения целостности. Рекомендуется использовать ограничения, а не триггеры , правила и значения по умолчанию. Оптимизатор запросов также использует определения ограничений для построения высокопроизводительных планов выполнения запросов.

Правила

Правила – еще один специальный механизм, предназначенный для обеспечения целостности базы данных, по функциональности напоминающие некоторые типы ограничений. Microsoft отмечает, что при соответствующей возможности использование ограничений по ряду причин предпочтительнее и , возможно, в будущей версии эта возможность будет удалена.

Значения по умолчанию

Значения по умолчанию определяют, какими значениями заполнять столбец, если при вставке строки для этого столбца значение не указано. Значение по умолчанию могут быть любым выражением, результат которого — константа, например собственно константой, встроенной функцией или математическим выражением.

10.3. Архитектура базы данных. Физический уровень

Физический уровень это представление данных в памяти ЭВМ. Как уже отмечалось в лекции 8 основными понятиями, используемыми для представления структуры хранения (физического уровня) являются понятия файла (физического) и единицы обмена между внешней и оперативной памятью (физической записи или страницы). Рассмотрим, как представлены соответствующие понятия в СУБД Microsoft SQL Server 2008 (http://msdn.microsoft.com).

Файлы и файловые группы

На физическом уровне база данных в Microsoft SQL Server 2008 представляется набором файлов операционной системы. Данные и сведения журналов транзакций всегда размещаются в разных файлах. Отдельные файлы используются только одной базой данных. Фай-

ловые группы представляют собой именованные коллекции файлов и используются для упрощения размещения данных и выполнения задач администрирования, например резервного копирования и восстановления.

Базы данных SQL Server содержат файлы трех типов:

- **Первичные файлы данных.**

Первичный файл данных является отправной точкой базы данных. Он указывает на остальные файлы базы данных. В каждой базе данных имеется один первичный файл данных. Для имени первичного файла данных рекомендуется использовать расширение MDF.

- **Вторичные файлы данных.**

Ко вторичным файлам данных относятся все файлы данных, за исключением первичного файла данных. Базы данных могут вообще не содержать вторичных файлов данных, или содержать один или несколько вторичных файлов данных. Для имени вторичного файла данных рекомендуется использовать расширение NDF.

- **Файлы журналов.**

Файлы журналов содержат все сведения журналов, используемые для восстановления базы данных. В каждой базе данных должен быть по меньшей мере один файл журнала, но их может быть и больше. Для имен файлов журналов рекомендуется использовать расширение LDF, MDF, NDF и LDF. Однако эти расширения помогают пользователю идентифицировать различные виды файлов и правильно их использовать.

В SQL Server расположение всех файлов базы данных записывается в первичный файл базы данных и в специальную служебную структуру СУБД SQL Server, называемою базой данных master. В большинстве случаев при работе с базой данных компонент СУБД (SQL Server Database Engine) использует сведения о размещении файлов, хранимые в базе данных master. Однако в некоторых случаях (например, при восстановлении базы данных master из копии, при определенным образом проводимым присоединении базы данных) компонент Database Engine использует сведения о расположении файлов из первичного файла, чтобы инициализировать записи о расположении файлов в базе данных master.

Файлы SQL Server имеют два имени:

- logical_file_name — имя, используемое для ссылки на физический файл во всех инструкциях Transact-SQL. Логическое имя файла должно соответствовать правилам для идентификаторов SQL Server и быть уникальным среди логических имен файлов в соответствующей базе данных.
- os_file_name — это имя физического файла, включая путь к каталогу. Оно должно соответствовать правилам для имен файлов операционной системы.

Изначально можно указать максимальный размер каждого файла. Если максимальный размер файла не указан, файлы SQL Server могут автоматически увеличиваться в размерах, превосходя первоначально заданные показатели, пока не займет все доступное место на диске. При определении файла пользователь может указывать требуемый шаг роста. Каждый раз при заполнении файла его размер увеличивается на указанный шаг роста. Если в файловой группе имеется несколько файлов, их автоматический рост начинается лишь по заполне-

нии всех файлов. Затем файлы увеличиваются в размерах по кольцевому списку. Эта функция особенно полезна в случаях, когда SQL Server используется в качестве базы данных, внедренной в приложение, где пользователь не имеет удобного доступа к системному администратору. По мере необходимости пользователь может предоставить файлам возможность увеличиваться в размерах автоматически, тем самым снимая с администратора часть забот по наблюдению за свободным пространством базы данных и по распределению дополнительного пространства вручную.

Из объектов баз данных и файлов можно формировать **файловые группы**, используемые для решения задач распределения и административного управления. Файлы журналов не могут входить в состав файловых групп. Управление пространством журнала отделено от управления пространством данных. Файл не может входить в состав нескольких файловых групп. Таблицы, индексы и данные больших объектов могут быть ассоциированы с указанной файловой группой. В этом случае все их страницы будут размещены внутри файловой группы; либо таблицы и индексы могут быть секционированы. Данные секционированных таблиц и индексов разделяются на блоки, каждый из которых может быть помещен в отдельную файловую группу базы данных. В каждой базе данных одна файловая группа назначается файловой группой по умолчанию. Если при создании таблицы или индекса файловая группа не указывается, предполагается, что все страницы будут распределяться из файловой группы по умолчанию. В каждый момент времени лишь одна файловая группа может быть файловой группой по умолчанию.

Страницы и экстенты

Основной единицей хранилища данных и обмена информацией между внешней и оперативной памятью в SQL Server является страница. Место на диске, предоставляемое для размещения файла данных (MDF- или NDF-файл) в базе данных, логически разделяется на страницы с непрерывным перечислением от 0 до n. Дисковые операции ввода-вывода выполняются на уровне страницы. А именно, SQL Server считывает или записывает целые страницы данных. В SQL Server размер страницы составляет 8 КБ. Это значит, что в одном мегабайте базы данных SQL Server содержится 128 страниц. Каждая страница начинается с 96-байтового заголовка, который используется для хранения системных данных о странице. Эти данные включают номер страницы, тип страницы, объем свободного места на странице и идентификатор единицы распределения объекта, которому принадлежит страница. В файлах данных базы данных SQL Server используется 8 типов страниц (данные с типами данных небольших размеров, данные с типами данных больших размеров, записи индекса, сведения о размещении экстентов, сведения о размещении страниц и доступном на них свободном месте и т. д.).

Для эффективного управления памятью страницы объединяются в экстенты , которые являются основными единицами организации пространства.

Экстент — это коллекция, состоящая из восьми физически непрерывных страниц или 64 Кб; они используются для эффективного управления страницами. Все страницы хра-

нятся в экстентах. Таким образом, в одном мегабайте базы данных SQL Server содержится 16 экстентов.

Чтобы сделать распределение места эффективным, SQL Server не выделяет целые экстенты для таблиц с небольшим объемом данных. SQL Server имеет два типа экстентов:

- Однородные экстенты принадлежат одному объекту (определенной таблице, индексу и т. д.); все восемь страниц могут быть использованы только этим владеющим объектом.
- Смешанные экстенты могут находиться в общем пользовании у не более восьми объектов. Каждая из восьми страниц в экстенте может находиться во владении разных объектов.

Новая таблица или индекс — это обычно страницы, выделенные из смешанных экстентов. При увеличении размера таблицы или индекса до восьми страниц эти таблица или индекс переходят на использование однородных экстентов для последовательных единиц распределения. При создании индекса для существующей таблицы, в которой содержится достаточно строк, чтобы сформировать восемь страниц в индексе, все единицы распределения для индекса находятся в однородных экстентах. Пример размещения объектов в смешанном и однородном экстентах приводится на рис. 10.2.

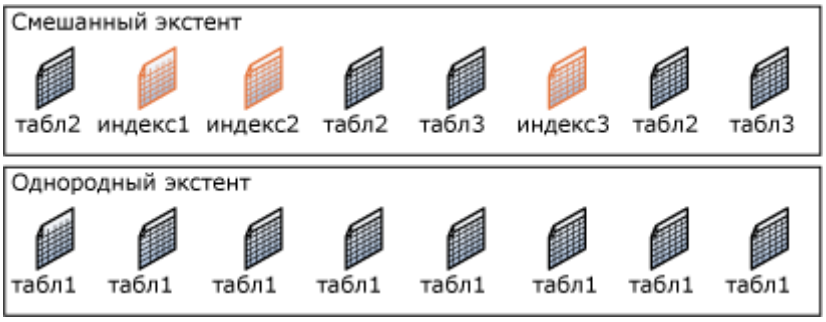


Рис. 10.2.Размещение объектов в смешанном и однородном экстентах

Страницы файлов данных

Страницы файлов данных SQL Server нумеруются последовательно; первая страница файла получает нулевой номер (0). Каждый файл базы данных имеет уникальный цифровой идентификатор. Чтобы уникальным образом определить страницу базы данных, необходимо использовать как идентификатор файла, так и номер этой страницы. На рис.10.3. показаны номера страниц базы данных, содержащей первичный файл данных объемом в 4 МБ и вторичный файл данных объемом в 1 МБ.

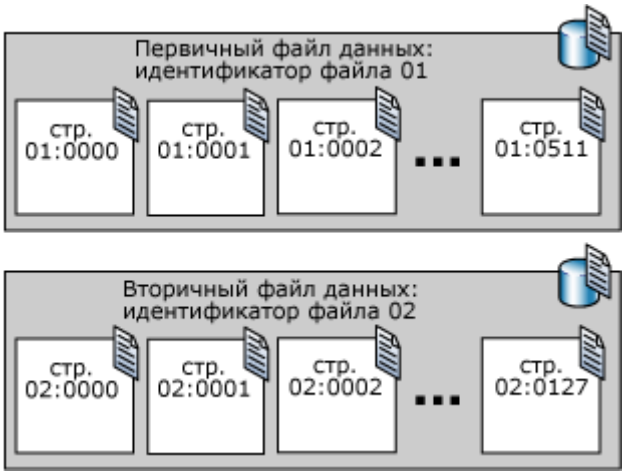


Рис. 10.3. Пример нумерации страниц файлов базы данных.

Первая страница каждого файла (страница с номером 0) — это страница заголовка файла; она содержит сведения об атрибутах данного файла. Страницы с номерами 1,2,3 будут описаны ниже.

Организация таблиц и индексов

Таблицы и индексы хранятся в виде коллекции страниц размером 8 КБ.

Страницы таблиц и индексов содержатся в одной или нескольких секциях. ***Секция — это пользовательская единица организации данных.*** По умолчанию таблица или индекс имеет единственную секцию, которая содержит все страницы таблицы или индекса. Секция располагается в одной файловой группе. Таблица или индекс, имеющие одну секцию, эквивалентны организационной структуре таблиц и индексов предыдущих версий SQL Server.

Если таблица или индекс используют несколько секций, данные секционируются горизонтально, так что группы строк сопоставляются отдельным секциям, основываясь на указанном столбце. Секции могут храниться в одной или нескольких файловых группах в базе данных. Таблица или индекс рассматриваются как единая логическая сущность при выполнении над данными запросов или обновлений. Секция состоит из фрагментов одного или нескольких файлов. Данные внутри фрагмента файла представляются в виде кучи (строки данных хранятся без определенного порядка – последовательное размещение) или сбалансированного дерева. Фрагмент файла может иметь один из трех видов: данные с типами небольших размеров (данные IN_ROW_DATA), данные с типами больших размеров (LOB_DATA), данные переменной длины (переполнение строки ROW_OVERFLOW_DATA).

В каждой секции кучи или индекса содержится по крайней мере одна единица распределения IN_ROW_DATA. Кроме того, в зависимости от схемы кучи или индекса, там могут содержаться единицы распределения LOB_DATA или ROW_OVERFLOW_DATA.

Следующая иллюстрация показывает организацию таблицы (рис. 10.4).

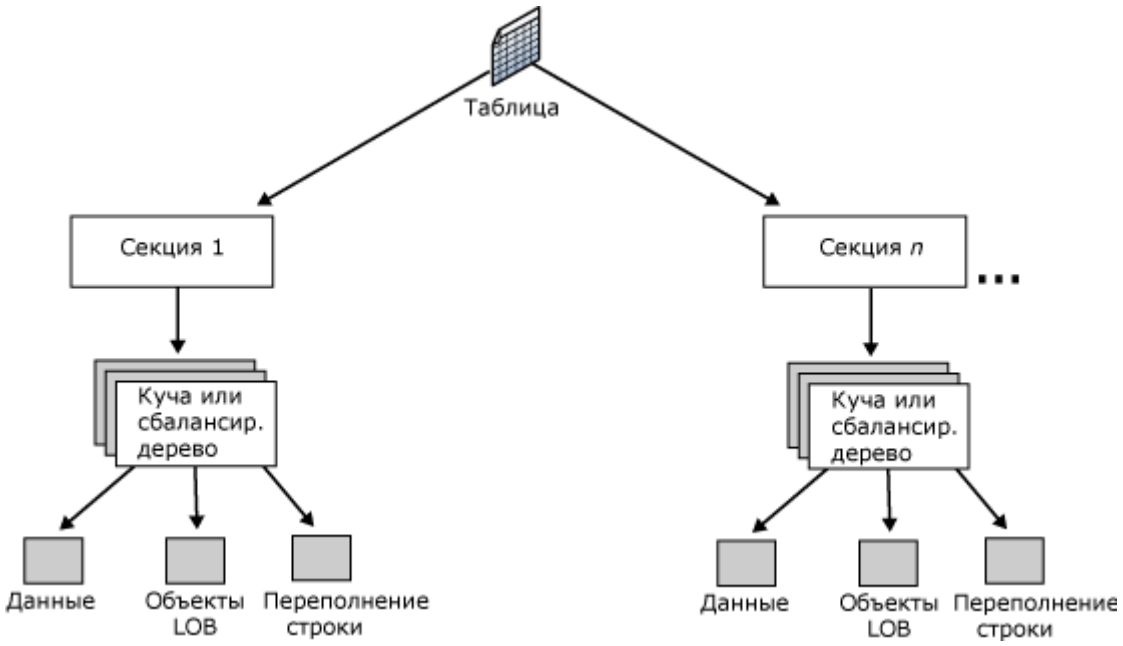


Рис.10.4. Физическая структура таблицы в базе данных SQL Server

Каждая секция содержит строки данных либо в куче, либо в структуре кластеризованного индекса. Кластеризованный индекс реализуется в виде структуры индекса сбалансированного дерева, которая поддерживает быстрый поиск строк по их ключевым значениям. Страницы в каждом уровне индекса, включая страницы данных на конечном уровне, связаны в двунаправленный список. Однако перемещение из одного уровня на другой выполняется при помощи ключевых значений.

Куча — это последовательность строк таблицы, которые не имеют кластеризованного индекса. Строки данных хранятся без определенного порядка, и какой-либо порядок в последовательности страниц данных отсутствует. Страницы данных не связаны в связный список.

Управление работой с экстентами и свободным местом

Структуры данных SQL Server, управляющие использованием экстента и отслеживанием свободного места, обладают относительно простой структурой. Сведения о свободном месте плотно упакованы, поэтому эти данные содержат относительно небольшое количество страниц. Это приводит к увеличению скорости из-за уменьшения необходимых операций чтения диска для получения сведений о размещении. Также увеличивается вероятность того, что страницы размещения будут оставаться в памяти и повторных операций чтения не потребуется. Большая часть сведений о размещении не связана по цепочке друг с другом. Это упрощает управление сведениями о размещении. Каждое действие по размещению или освобождению страницы может выполняться быстро. Это сокращает конфликты между одновременными задачами использования и освобождения страниц.

SQL Server использует два типа карт для записи сведений об использовании экстентов:

- **Глобальная карта распределения (GAM)**

На GAM-страницах записано, какие экстенты были задействованы. В каждой карте GAM содержится сведения об использовании 64 000 экстентов или о размещении почти 4

ГБ данных. В карте GAM приходится по одному биту на каждый экстент в покрываемом им интервале. Если бит равен 1, то экстент свободен; если бит равен 0, то экстент задействован.

- **Общая глобальная карта распределения (SGAM)**

На SGAM-страницах записано, какие экстенты в текущий момент используются в качестве смешанных экстентов и имеют как минимум одну неиспользуемую страницу. В каждой карте SGAM содержится сведения об использовании 64 000 экстентов или о размещении почти 4 ГБ данных. В карте SGAM приходится по одному биту на каждый экстент в покрываемом им интервале. Если бит равен 1, то экстент используется как смешанный экстент и имеет свободную страницу. Если бит равен 0, то экстент не используется как смешанный экстент, или он является смешанным экстентом, но все его страницы используются.

Это дает простые алгоритмы управления экстентами страниц. Для использования для хранения объекта однородного экстента компонент СУБД Database Engine производит на карте GAM поиск бита 1 и заменяет его на бит 0. Для поиска смешанного экстента со свободными страницами компонент Database Engine производит поиск на карте SGAM бита 1. Для размещения смешанного экстента компонент Database Engine производит на карте GAM поиск бита 1 и заменяет его на бит 0, а затем устанавливает значение соответствующего бита на карте SGAM равным 1. Для освобождения экстента компонент Database Engine устанавливает бит GAM равным 1, а соответствующий бит SGAM равным 0. Внутренние алгоритмы, которые на самом деле используются компонентом Database Engine, более сложны, чем это описано в данном подразделе, так как компонент Database Engine распространяет данные в базе данных равномерно. Однако даже настоящие алгоритмы упрощаются из-за того, что отпадает необходимость управления цепочками сведений о размещении экстентов.

Отслеживание свободного места

На страницы PFS (Page Free Space) записывается состояние размещения каждой страницы, информация о том, была ли отдельная страница использована или нет, а также количество свободного места на каждой странице. В PFS на каждую страницу приходится по одному байту, хранящему информацию о том, была ли страница использована или нет, а если была — то пустая она, или ее заполнение находится в промежутке от 1 до 50 процентов, от 51 до 80 процентов, от 81 до 95 процентов или от 96 до 100 процентов.

После размещения объекта в экстенте компонент Database Engine использует PFS-страницы для записи информации о том, какие страницы в экстенте использованы, а какие свободны. Эти сведения используются компонентом Database Engine при выборе новой страницы для размещения объектов. Количеством свободного места на странице можно управлять только в случае кучи и страниц с типами данных «Текст» и «Примечание». Это используется при поиске страницы, обладающей свободным местом, достаточным для сохранения в ней новой добавляемой строки. Для индексов не требуется, чтобы отслеживалось свободное место на странице, так как место, в которое будет вставляться новая строка, назначается значениями ключа индекса.

PFS-страница является первой страницей после страницы заголовка файла в файле данных (страница номер 1). Потом следует GAM-страница (страница номер 2), а затем SGAM-

страница (страница номер 3). После первой PFS-страницы находится PFS-страница размером примерно 8 000 страниц. После первой GAM-страницы на странице 2 находится другая GAM-страница с 64 000 экстентов и другая SGAM-страница с 64 000 экстентов находится после первой SGAM-страницы на странице номер 3. На рис. 10.5. показана последовательность страниц, используемая компонентом Database Engine, для размещения и управления экстентами.

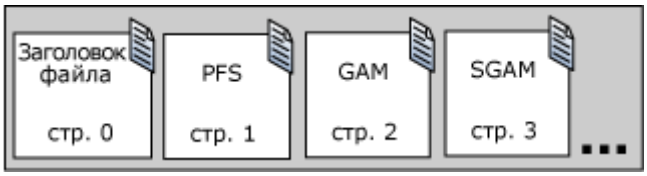


Рис.10.5. Страницы файла, используемые для размещения и управления экстентами.

Краткие итоги: В лекции рассмотрена архитектура одной из наиболее распространенных клиент-серверных СУБД - Microsoft SQL Server. Описаны основные составляющие архитектуры соответствующей СУБД на разных уровнях абстракции. Рассмотрен логический уровень (уровень модели данных СУБД как средство представления концептуальной модели), включающий следующие понятия: таблицы и типы данных, первичные и внешние ключи, индексы, представления, сборки, ограничения, правила, значения по умолчанию). Рассмотрен физический уровень (внутреннее представление данных в памяти ЭВМ - физическая структура базы данных), включающий следующие понятия: файлы и файловые группы, файлы журналов, страницы и экстенты, физическую организацию таблиц и индексов, управление работой с экстентами и памятью).

Более подробно с материалами данной лекции можно ознакомиться в [1-3]

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Общие представления об архитектуре базы данных Microsoft SQL Server .

Вариант 1.
Какие уровни описания данных различают в СУБД Microsoft SQL Server?

☐ концептуальный
☐ внешний
⊞ логический
⊞ физический
⊞ внутренний

Вариант 2.
Какие уровни описания данных не представлены в СУБД Microsoft SQL Server ?

⊞ концептуальный
⊞ внешний
☐ логический
☐ физический
☐ внутренний

Вариант 3.
Из какого уровня в какой и кто осуществляет отображение в СУБД Microsoft SQL Server ?

☐ из внешнего во внутренний, пользователь
☐ из внешнего во внутренний, СУБД
☐ из логического в физический, пользователь
⊞ из логического в физический, СУБД
⊞ из логического во внутренний, СУБД
☐ из логического во внутренний, пользователь

Задача 2. Как определяется логический уровень представления данных в СУБД Microsoft SQL Server ?

Вариант 1.
Какие понятия являются понятиями логического уровня СУБД Microsoft SQL Server?

☐ файлы
☐ группы файлов
⊞ таблицы
☐ страницы
⊞ представления

Вариант 2.
Какие понятия не являются понятиями логического уровня СУБД Microsoft SQL Server?

⊞ файлы
☐ группы файлов
☐ таблицы
⊞ страницы
☐ представления
⊞ экстенты
☐ ограничения целостности
☐ типы данных

Вариант 3.
Какие понятия логического уровня используются для обеспечения целостности базы данных?

☐ страницы
☐ представления
☐ индексы
⊞ ограничения
⊞ триггеры
☐ экстенты
⊞ правила

Задача 3. Как определяется физический уровень представления данных в СУБД Microsoft SQL Server?

Вариант 1.
Какие понятия являются понятиями физического уровня СУБД Microsoft SQL Server?

⊞ файлы
⊞ группы файлов
☐ таблицы
⊞ страницы
☐ представления

Вариант 2.
Какие понятия не являются понятиями физического уровня СУБД Microsoft SQL Server?

☐ файлы
☐ группы файлов
⊞ таблицы
☐ страницы
⊞ представления
☐ экстенты
⊞ ограничения целостности
⊞ типы данных

Вариант 3.
Какое понятие используется в качестве минимальной физической составляющей файла при обменах между оперативной и внешней памятью?

☐ группа файлов
☐ экстент
⊞ страница
☐ индекс
☐ представление

Задача 4. Как выглядит база данных SQL Server на физическом уровне?

Вариант 1.
Какие типы файлов содержит база данных SQL Server на физическом уровне?

⊞ первичный файл
⊞ вторичный файл
⊞ файл журналов транзакций
☐ страница
☐ секция

Вариант 2.
Как формируется размер файла?

⊞ пользователь задает максимальный размер файла
⊞ пользователь может предоставить файлам возможность автоматически увеличиваться по мере ввода данных
☐ пользователь не дает никаких данных
☐ пользователь указывает максимально доступное место на диске

Вариант 3.
Что может входить в состав файловых групп?

⊞ первичный файл
⊞ вторичный файл
☐ файл журналов транзакций
☐ секция

Задача 5. Как осуществляется поиск данных в физической базе данных?

Вариант 1.
Что является идентификатором поиска на уровне дисковой памяти?

☐ первичный ключ
⊞ идентификатор файла
⊞ идентификатор файла, номер страницы
☐ идентификатор файла, номер страницы, номер записи

Вариант 2.
Что не используется в качестве идентификатора при поиске на уровне дисковой памяти?

⊞ первичный ключ
☐ идентификатор файла
☐ идентификатор файла, номер страницы
⊞ идентификатор файла, номер страницы, номер записи

Вариант 3.
Как выглядит номер страницы файла?

☐ порядковый номер
⊞ номер файла, порядковый номер
☐ номер файла, номер экстента, порядковый номер
☐ номер файла, порядковый номер, значение ключа

Задача 6. В виде каких структур хранятся таблицы в базе данных SQL Server?

Вариант 1.
Какие понятия используются при описании представления таблиц в базе данных SQL Server?

⊞ секция
☐ куча
⊞ коллекция страниц
⊞ сбалансированное дерево

Вариант 2.
Какие понятия используются при описании представления индексов в базе данных SQL Server?

☐ секция
☐ куча
☐ коллекция страниц
⊞ сбалансированное дерево

Вариант 3.
В каких структурах используются двунаправленные списки?

☐ секция
☐ куча
☐ коллекция страниц
⊞ сбалансированное дерево

Задача 7. Как осуществляется управление использованием экстентов и свободной памяти?

Вариант 1.
Какие структуры используются для управления использованием экстентов?

☐ секции
⊞ глобальная карта распределения
⊞ общая глобальная карта распределения
☐ сбалансированное дерево

Вариант 2.
Какие структуры используются для управления использованием смешанных экстентов?

☐ секции
☐ глобальная карта распределения
⊞ общая глобальная карта распределения
☐ сбалансированное дерево

Вариант 3.
Какие структуры используются для управления использованием однородных экстентов?

☐ секции
⊞ глобальная карта распределения
☐ общая глобальная карта распределения
☐ сбалансированное дерево

Задача 8. Как происходит отслеживание свободного места?

Вариант 1.
Как происходит отслеживание свободного экстента?

⊞ выбирается бит, равный 1 в глобальной карте распределения
⊞ выбирается бит, равный 1 в общей глобальной карте распределения
☐ выбирается бит, равный 0 в общей глобальной карте распределения
☐ выбирается бит, равный 0 в глобальной карте распределения
☐ выбирается бит, равный 1 на странице PFS
☐ выбирается бит, равный 0 на странице PFS

Вариант 2.
Как выбирается свободная страница?

☐ выбирается бит, равный 1 в глобальной карте распределения
☐ выбирается бит, равный 1 в общей глобальной карте распределения
☐ выбирается бит, равный 0 в общей глобальной карте распределения
☐ выбирается бит, равный 0 в глобальной карте распределения
⊞ выбирается бит, равный 1 на странице PFS
☐ выбирается бит, равный 0 на странице PFS

Вариант 3.
В каком случае можно добавлять данные в неполностью заполненную страницу?

⊞ если она заполнена от 1 до 50%
⊞ если она заполнена от 51 до 80%
⊞ если она заполнена от 81 до 95%
⊞ если она заполнена до 96 %

Литература

1. http://msdn.microsoft.com
2. Реализация баз данных Microsoft SQL Server 7.0. Учебный курс: официальное пособие Microsoft для самостоятельной подготовки/ Пер с англ. – М.: Издательско-торговый дом «Русская Редакция», 2000.
3. Мамаев Е. Microsoft SQL Server 2000 в подлиннике. СПб.: Изд-во BHV, 2001.

Лекция 11. Программное обеспечение работы с современными базами данных

В лекции рассматриваются общие принципы организации программного обеспечения работы с реляционными базами данных, включающего

- **создание и ведение базы данных;**
- **создание пользовательских приложений, включающих разработку пользовательского интерфейса по работе с базой данных.**

Ключевые слова: программное обеспечение баз данных, средства работы с данными, средства создания интерфейса с базой данных, язык SQL, интерпретируемый язык запросов.

Цель лекции: дать общее представление об основных задачах программного обеспечения баз данных, существующих подходов к решению этих задач, в том числе и о структурированном языке запросов SQL.

11.1. Основные задачи программного обеспечения баз данных

При работе с реляционными базами данных можно условно выделить две основные задачи:

- собственно работа с базой данных , включающая создание и ведение базы данных (создание структур таблиц, добавление записи в таблицу, удаление записи, обновление, выборка нужной записи);
- создание пользовательских приложений, включающих разработку пользовательского интерфейса по работе с базой данных.

Для решения указанных задач современные СУБД в своем составе могут содержать следующие программные средства: языки процедурного пошагового программирования, средства визуального программирования (графический интерфейс, диспетчер проекта, мастера и построители), средства создания объектно-ориентированных приложений. Кроме этого, при разработке пользовательских программ во многих СУБД допускается использование других языков программирования, а также использование библиотек разного рода. Так, например, при работе с СУБД ACCESS можно использовать язык программирования ACCESS , мастер ACCESS и язык программирования VISUAL BASIC.

При работе с клиент-серверными системами ситуация немного сложнее. Здесь в работе участвуют два типа компьютеров (сервер и клиент) и, соответственно, различают клиентское и серверное программное обеспечение. Серверное программное обеспечение включает язык программирования, поддерживающий создание и ведение базы данных, также реализацию поступающих от клиентов запросов пользователей к базе данных. Пользовательские приложения создаются и работают на компьютерах-клиентах. Именно эти компьютеры должны иметь, наряду со средствами формирования запросов к базе данных, средства разработки интерфейса. В связи с этим, для клиент-серверных СУБД программное обеспечение разделяется на две части: программное обеспечение – клиент и программное обеспечение – сервер. Заметим, что наряду с программным обеспечением – клиент, при разработке пользователь-

ских программ в конкретной СУБД могут использоваться другие языки программирования, специальные библиотеки, другие системы программирования (определенные для этой СУБД). В качестве примера в таблице приводятся возможные варианты использования программного обеспечения для организации клиент-серверного взаимодействия в СУБД Microsoft SQL Server.

Таблица 11.1

Возможные варианты использования программного обеспечения в СУБД MS SQL Server

Средства ведения баз данных на сервере MS SQL	Средства разработки клиентских приложений
Службы SQL-сервер (MS SQL server и др.)	Программное обеспечение клиента SQL-сервер (Transact SQL, SQL Server Query Analyzer и др.) MS Access (ODBC) MS Visual Basic MS Visual Studio MS Visual FoxPro Java (JDBC) Borland Delphi Borland C++Builder и др. Библиотеки функций (API, ODBC и др.)

Полное рассмотрение всего спектра программного обеспечения работы СУБД очень обширно и выходит за рамки данного пособия. Поэтому в данной работе будет рассмотрены только средства создания и ведения базы данных.

11.2. Проблемы создания и ведения реляционных баз данных

При создании базы данных и организации работы с ней возникают три основные проблемы:

- собственно создание базы данных (создание таблиц, индексов, ограничений целостности);
- обеспечение безопасности и разграничения доступа;
- организация доступа к элементам таблицы (выборка, редактирование, удаление, добавление).

Первая проблема может быть решена посредством создания в каждой конкретной СУБД некоторой утилиты, позволяющей пользователю в определенный момент осуществлять все необходимые действия по созданию базы данных. Однако это не полностью решает проблему. Такая утилита не позволяет создать таблицу динамически во время работы прикладной программы, не позволяет, например, добавить в таблицу сформированнный во время работы пользовательской программы столбец. Необходимы средства, дающие возмож-

ность формирования во время работы прикладной программы запроса на изменение структуры и содержания базы данных. То же самое можно сказать и о решении второй проблемы

Более подробно рассмотрим возможный путь решения третьей проблемы. Организация доступа к базе данных является важнейшей функцией информационной системы. Пользователи постоянно работают с данными. Рассмотрим простой пример. Пусть у нас есть таблица СТУДЕНТ, хранящая информацию следующего рода:

СТУДЕНТ (Код студента, Фамилия, Имя, Отчество, Дата поступления).

Теперь мы хотим выполнить некоторый запрос к базе данных, результатом которого должны стать те строки таблицы СТУДЕНТ, для которых дата поступления окажется больше 01.06.2006. Рассмотрим последовательность действий для реализации данного запроса.

1. Получаем доступ к таблице СТУДЕНТ и устанавливаем указатель текущей строки на первую строку таблицы.
2. Анализируем поле «Дата поступления » в текущей строке.
3. Если значение «Дата поступления» > «01.06.2006», распечатываем на экране данные об абитуриенте.
4. Если таблица не кончилась, перемещаем указатель текущей строки на следующую строку и переходим к шагу 2, иначе заканчиваем работу.

Любой человек, знакомый с программированием, легко представит себе реализацию подобного алгоритма на любом языке программирования высокого уровня. Вот, в частности, пример реализации на Object Pascal:

```
Table.First;
while (not Table.Eof) do
begin
      if FieldByName(«Дата поступления»).Value >
         «01.06.2006»
      then List.Add(FieldByName(«Фамилия»).AsString);
      Table.Next;
end;
```

В этом случае разработчик приложения сам организует работу по выборке данных, программируя каждое движение по таблице (осуществляет навигацию по таблице). ***Такой подход к обработке данных, ориентированный на последовательную работу с отдельными записями, называется навигационным.*** Здесь на конкретном языке программирования мы описываем процедуру - последовательность действий, необходимых для получения результата. Языки, в которых используется такой подход, называются процедурными. Очевидно, что с возрастанием сложности запроса существенно возрастает сложность процедуры и, соответственно, объем текста программы.

При работе с информационной системой пользователь реализует свои запросы к базе данных с помощью разработанных программистами прикладных программ. При навигационном подходе должны быть запрограммированы все возможные запросы. Очевидно, что заранее предугадать все запросы, потребность в которых может возникнуть и запрограмми-

ровать их, невозможно. С учетом того, что подавляющее большинство пользователей не владеет навыками программирования, это означает, что объем их действий будет ограничен рамками написанных программ, а именно, теми запросами, реализация которых предусмотрена заранее.

Кроме того, должен учитываться механизм взаимодействия в рамках архитектуры «клиент – сервер». Пользовательская программа выполняется на компьютере – клиент. Запрос к базе данных реализуется компьютером – сервером. Необходим механизм формирования в пользовательской программе клиента запроса к базе данных сервера. В этом случае навигационный подход неприемлем. ***В связи с этим, для работы с базами данных разработан и используется другой подход, основанный на использовании так называемых языков запросов, которые задают не последовательность необходимых действий, а условия, которым должен удовлетворять результат*** (при добавлении столбца, выборке записи, добавлении записи и т. п.). Такой подход решает все три вышеперечисленные проблемы.

С этой целью разработан и активно используется во всех базах данных – специальный язык запросов SQL. Особо отметим, что основой языка являются операции реляционной алгебры.

Язык SQL (Structured Query Language – структурированный язык запросов) применяется для общения пользователя с реляционной базой данных **и состоит из трех частей** [2]:

- DDL (Data Definition Language) – язык определения данных. Предназначен для создания базы данных (таблиц, индексов и т.д.) и редактирования ее схемы.
- DCL (Data Control Language) – язык управления данными. Содержит операторы для разграничения доступа пользователей к объектам базы данных.
- DML (Data Manipulation Language) – язык обработки данных. Содержит операторы для внесения изменений в содержимое таблиц базы данных.

Как видно из написанного выше, SQL решает все рассмотренные ранее вопросы, предоставляя пользователю достаточно простой и понятный механизм доступа к данным, не связанный с конструированием алгоритма и его описанием на языке программирования высокого уровня. Так, вместо указания того, **как** необходимо действовать, пользователь при помощи операторов SQL объясняет СУБД, **что** ему нужно сделать. Далее СУБД сама анализирует текст запроса и определяет, как именно его выполнять.

В архитектуре «клиент – сервер» язык SQL занимает очень важное место. Именно он используется как язык общения клиентского программного обеспечения с серверной СУБД, расположенной на удаленном компьютере. Так, клиент посылает серверу запрос на языке SQL, а сервер разбирает его, интерпретирует, выбирает план выполнения, выполняет запрос и отсылает клиенту результат.

Посмотрим, как выглядит запрос на языке SQL, решающий задачу о выборке студентов по дате поступления.

```
SELECT Фамилия
FROM  Студент
WHERE Дата поступления > «01.06.2006»
```

Может сложиться ложное впечатление, что появление языка SQL является альтернативой языков программирования высокого уровня. Это не соответствует действительности. Выполнение запроса средствами SQL все равно сводится к работе с отдельными записями, и от этого никуда не уйти. Важно понимать, что появление языка SQL дало, по крайней мере, две новые возможности.

1. Появился новый уровень абстракции между пользователем и СУБД. Этот уровень находится ближе к пользователю, чем уровень программирования на языке высокого уровня, что снижает требования к квалификации пользователей.
2. Многие типовые задачи, возникающие при работе с базами данных, и ранее решаемые каждым программистом по-своему (зачастую дублируя действия другого программиста) решены реализацией языка SQL. Таким образом, отпала необходимость самостоятельного решения многих проблем, решенных в СУБД соответствующим образом. Язык SQL предоставляет средства для доступа к этим типовым возможностям СУБД.

11.3. Понятие языка SQL и его основные части

11.3.1. История возникновения и стандарты языка SQL

История возникновения языка SQL восходит к 1970 году [1], когда доктор Е.Ф. Кодд предложил реляционную модель в качестве новой модели базы данных. Для доказательства жизнеспособности новой модели данных внутри компании IBM был создан мощный исследовательский проект, получивший название System/R. Проект включал разработку собственно реляционной СУБД и специального языка запросов к базе данных. Так в начале 70-х годов появился первый исследовательский прототип реляционной СУБД. Для этого прототипа разрабатывались и опробовались разные языки запросов, один из которых получил название SEQUEL (Structured English Query Language). С момента создания и до наших дней этот язык претерпел массу изменений, но идеология осталась неизменной

Период с 1979 года (окончание проекта System/R) до настоящего времени характеризуется развитием и совершенствованием языка SQL и его постоянно увеличивающейся ролью в индустрии, связанной с созданием и эксплуатацией баз данных. Совершенно очевидно, что язык никогда не получил бы мирового признания, если бы на него не было никаких стандартов. Стандартизация – важная часть технологических процессов конца XX века. Именно наличие разработанных и официально признанных стандартов позволило утвердиться многим современным технологиям (не только в индустрии разработки программного обеспечения, но и в многих других сферах человеческой деятельности). Как обстоит дело со стандартами языка SQL и их поддержкой в распространенных СУБД?

Когда ведут речь о стандартах в области, связанной с разработкой программного обеспечения, обычно подразумевают две организации:

- ANSI (American National Standards Institute) – Американский национальный институт стандартов;

- ISO (International Standards Organization) – Международную организацию по стандартизации.

Работа над официальным стандартом языка SQL началась в 1982 году [8] в рамках комитета ANSI. В 1986 году (обратите внимание, сколько времени ушло на разработку стандарта и согласование деталей!) был утвержден первый вариант стандарта ANSI, а в 1987 году этот стандарт был утвержден и ISO. В 1989 году стандарт претерпел незначительные изменения, но именно этот вариант получил название SQL-1 или SQL-89. В чем особенность SQL-89? За время разработки стандарта (1982–1989 гг.) были созданы, представлены на рынке и активно использовались несколько различных СУБД, в которых в том или ином виде был реализован некоторый диалект языка SQL. С учетом того, что разработкой стандартов занимались те же люди, кто внедрял SQL в СУБД, стандарт SQL-89 представлял собой плод множества компромиссов, приведших к наличию в нем большого количества «белых пятен», т.е. мест, которые не были описаны, а отданы на усмотрение разработчиков диалекта. В результате чуть ли не все имеющиеся диалекты стали совместимыми со стандартом, но особой пользы это не принесло.

Следующая реализация стандарта была призвана решить эту проблему. В результате длительных обсуждений и согласований в 1992 году был принят новый стандарт ANSI SQL-2 или SQL-92. SQL-92, который заполнил многие «белые пятна», впервые добавив в стандарт возможности, еще не реализованные в существующих коммерческих СУБД.

Работа над стандартизацией продолжается и далее. Появились стандарты SQL-1999, SQL-2003. Тем не менее, все эти стандарты не решили всех проблем, связанных с наличием нескольких диалектов языка. Как правило, разработчики как игнорировали, так и игнорируют некоторые положения стандарта, с одной стороны, отказываясь реализовывать некоторые его части и, с другой стороны, реализуя то, что отсутствует в стандарте. Несмотря на имеющиеся отличия, все коммерческие СУБД поддерживают некоторое ядро языка, описанное в стандарте, одинаково. Отличий не очень много, они не носят слишком принципиального характера. Хотя каждая СУБД по-прежнему поддерживает свой диалект языка.

В систему управления базами данных Microsoft SQL Server входит язык Transact-SQL, разработанный на основе одного из стандартов SQL. .

11.3.2. Достоинства языка SQL

Для ознакомления с достоинствами языка обратимся к соответствующей литературе [1]. Вот некоторые из них:

- межплатформенная переносимость;
- наличие стандартов;
- одобрение и поддержка компанией IBM (СУБД DB2);
- поддержка со стороны компании Microsoft (СУБД SQL Server, протокол ODBC и технология ADO);
- реляционная основа;
- высокоуровневая структура;

- возможность выполнения специальных интерактивных запросов;
- обеспечение программного доступа к базам данных;
- возможность различного представления данных;
- полноценность как языка, предназначенного для работы с базами данных;
- возможность динамического определения данных;
- поддержка архитектуры клиент/сервер;
- поддержка корпоративных приложений;
- расширяемость и поддержка объектно-ориентированных технологий;
- возможность доступа к данным в Интернете;
- интеграция с языком Java (протокол JDBC);
- промышленная инфраструктура.

11.3.2. Общая характеристика SQL

Язык запросов SQL основан на операциях реляционной алгебры и, таким образом ориентирован на работу с множествами (отношениями), а не с отдельными записями. Как и в реляционной алгебре, операндами языка являются отношения (таблицы), результатами выполнения операции также являются отношения (таблицы). Таким образом, язык SQL предназначен для выполнения операций над таблицами, причем как над таблицами в целом (создание, удаление, изменение структуры), так и над данными таблиц (выборка, изменение, добавление и удаление). Отметим, что в явном виде язык SQL не является универсальным языком программирования в обычном понимании. В нем отсутствуют операторы условного перехода, организации циклов, позволяющие управлять ходом выполнения программы. Поэтому язык SQL относится к классу непроцедурных языков программирования. Это именно язык запросов к базе данных, который служит исключительно для организации базы данных и работы с ней. Как уже отмечалось выше, для разработки прикладных программ необходимо использовать другие базовые средства программирования, в который операторы языка SQL будут встраиваться. Языку SQL посвящено большое количество литературы, в том числе и учебников. Подробное изучение языка SQL не входит в задачи настоящего курса, это может занимать отдельный курс. Заметим, что этому языку посвящено большое количество литературы, в том числе и учебников. В связи с этим , здесь будут изложены только общие сведения о языке, как фундаментальном инструменте работы с базами данных.

Терминология

Под запросом, реализуемым с помощью языка SQL- запросов к базе данных, понимается команда, предназначенная для выполнения (и выполняемая) системой управления базами данных определяемого этой командой действия с базой данных.

Запрос реализуется с помощью операторов языка SQL. Операторы состоят из отдельных логических частей, называемых предложениями. Стандарты языка SQL регламентируют синтаксис операторов. Несмотря на то, что язык SQL работает с реляционной

базой данных, вместо термина «отношение» здесь используется термин «таблица», вместо терминов «кортеж» и «атрибут» используются соответственно термины «строка» и «столбец».

Разновидности SQL

Как отмечалось выше, в отличие от «обычных» языков программирования в SQL отсутствует возможность объявления переменных, нет инструкции IF, нет цикла FOR и т.д. Собственно программирование (разработка прикладных программ) на подобном языке практически невозможно. Поэтому к настоящему моменту используются следующие технологии (режимы) работы с базой данных на языке SQL (в некоторых источниках эти технологии называют разновидностями языка SQL):

- формирование непосредственно пользователем запроса на языке SQL в интерактивном режиме (**интерактивный SQL**);
- формирование запроса на языке SQL в прикладной программме (программный или встроенный SQL:
  - статическое формирование запроса (**статический SQL**);
  - динамическое формирование запроса (**динамический SQL**);
  - формирование запроса с помощью библиотек (**API – интерфейсы вызова подпрограмм**).

В интерактивном режиме работы с базой данных: пользователь работает с базой данных в прямом диалоге: вводит запрос на языке SQL – получает результат, вводит другой запрос – получает другой результат и т.д.

Встроенный SQL представляется операторами языка SQL, встроенные в прикладные программы, написанные на других языках программирования (в других программных средах). Это дает возможность работы с базой данных с помощью прикладных программ, написанных на других алгоритмических языках, но требует включения дополнительных средств, обеспечивающих интерфейс между операторами языка SQL и соответствующим языком программирования.

При статическом использовании языка (**статический SQL**) в текст прикладной программы включаются конкретные операторы SQL, и после компиляции исходной программы в выполняемый модуль жестко включаются соответствующие эти операторам функции SQL. Изменения в вызываемых функциях могут здесь определяться только изменениями параметров операторов SQL, инициируемых с помощью переменных языка программирования.

При динамическом использовании языка (**динамический SQL**) формирование SQL-запросов, соответствующие вызовы SQL-функций для обращения к базе данных осуществляется динамически в ходе выполнения программы.

Еще одним способом динамического формирования SQL-запросов в прикладной программе является обращение к соответствующим SQL-функциям с помощью специальных **интерфейсов программирования приложений** (библиотек функций, разработанных для связи прикладной программы и СУБД посредством SQL-запросов).

В настоящем пособии для всех указанных технологий (разновидностей SQL) будут приведены основные идеи и рассмотрены ключевые концепции. Интерактивный SQL будет рассмотрен более подробно, чем программный. Детальное рассмотрение статического, динамического SQL и различных API-интерфейсов (ODBC, JDBC, DB Library и др.) выходит за рамки нашего курса.

Краткие итоги: В лекции рассматриваются общие принципы организации программного обеспечения работы с реляционными базами данных, включающего

- создание и ведение базы данных;
- создание пользовательских приложений, включающих разработку пользовательского интерфейса по работе с базой данных.

Рассматриваются подходы к организации доступа к данным (навигационный подход и подход, основанный на использовании интерпретируемых языков запросов). Дается общее представление о языке SQL (история возникновения и стандарты языка SQL, достоинства языка SQL, основная терминология, технологии работы).

По языку SQL написано достаточно много литературы. Для более подробного знакомства можно указать, в частности [1-6].

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Какие основные направления использования программного обеспечения клиент-серверных СУБД?

Вариант 1.
Какие основные задачи программного обеспечения можно выделить при разработке прикладных программ, работающих с базой данных?

⊞ создание базы данных
⊞ организация работы с базой данных
⊞ создание пользовательского интерфейса
☐ разработка программ СУБД
☐ разработка вычислительных процедур

Вариант 2.
Что можно создавать с помощью программного обеспечения компьютера-сервера?

☐ прикладную программу
☐ интерфейс пользователя
⊞ базу данных
⊞ запросы к базе данных

Вариант 3.
Что можно делать с помощью программного обеспечения компьютера-сервера и компьютера-клиента?

⊞ Прикладную программу
⊞ Интерфейс пользователя
☐ Базу данных
⊞ Запросы к базе данных

Задача 2.

Вариант 1.
Какие средства программирования могут быть использованы для ведения баз данных на сервере SQL-Server?

⊞ MS SQL-Server
☐ библиотеки функций
⊞ Transact SQL
☐ MS Acces
☐ MS Visual Basic

Вариант 2.
Какие средства программирования могут быть использованы для разработки клиентской части?

⊞ библиотеки функций
⊞ Transact SQL
⊞ MS Acces
⊞ MS Visual Basic
⊞ MS Visual Studio

Вариант 3.
Операторы каких средств программирования могут быть использованы как в серверной так и в клиентской части?

☐ библиотеки функций
⊞ Transact SQL
☐ MS Acces
☐ MS Visual Basic
☐ MS Visual Studio

Задача 3. В чем состоит отличие процедурного языка программирования от языка запросов при работе с таблицами?

Вариант 1.
Как формулируется алгоритм работы с таблицей с помощью процедурного языка программирования?

⊞ как последовательность движений по таблице с выполнением других необходимых действий
☐ как указание выбрать данные, удовлетворяющие заданным условиям
⊞ осуществлением навигации по таблице
⊞ как последовательная работа с отдельными записями таблицы

Вариант 2.
Как формулируется алгоритм работы с таблицей с помощью языка запросов?

☐ как последовательность движений по таблице с выполнением других необходимых действий
⊞ как указание выбрать данные, удовлетворяющие заданным условиям
☐ осуществлением навигации по таблице
☐ как последовательная работа с отдельными записями таблицы

Вариант 3.
Как осуществляется работа с таблицами при взаимодействии компьютера-клиента с сервером?

☐ путем навигации по таблице
☐ с помощью процедур
⊞ с помощью запросов
☐ с помощью языков программирования

Задача 4. Почему необходимо иметь стандарт языка?

Вариант 1.
Что достигается введением стандарта языка ?

⊞ единообразие возможностей работы с базами данных в разных СУБД
⊞ создание прототипа описания языка запросов к базе данных
⊞ создание условий для устранения разных реализаций аналогичных операций с базой данных в разных СУБД
☐ полная унификация языков запросов в разных СУБД

Вариант 2.
Что реализовано в современных СУБД?

☐ стандарт языка запросов
⊞ диалект языка запросов
⊞ основные положения стандарта языка запросов
⊞ ядро стандарта языка запросов

Вариант 3.
Как связаны диалект языка и стандарт языка?

⊞ диалект языка игнорирует некоторые положения стандарта
⊞ имеются положения диалекта языка, не входящие в стандарт языка
⊞ имеются положения стандарта языка, не входящие в диалект языка
☐ диалект языка является конкретной реализацией всех положений стандарта языка

Задача 5. Основные свойства языка запросов SQL

Вариант 1.
Каковы основные достоинства языка SQL?

Универсальный язык программирования
⊞ наличие стандарта
⊞ реляционная основа
⊞ поддержка архитектуры клиент-сервер
⊞ использование во многих СУБД
☐ использование для разработки прикладных программ

Вариант 2.
Что является операндами в операторах языка SQL?

⊞ отношение
☐ кортеж
☐ домен
☐ атрибут

Вариант 3.
Что является результатами выполнения операторов языка SQL?

⊞ отношение
☐ кортеж
☐ домен
☐ атрибут

Задача 6. Как осуществляется работа с интерактивным SQL?

Вариант 1.
Как пользователь работает с интерактивным SQL?

☐ вставляет текст на языке SQL в прикладную программу
⊞ вводит непосредственно запрос на языке SQL
☐ обращается к языку SQL из прикладной программы
⊞ вводит последовательно несколько запросов к базе данных на языке SQL

Вариант 2.
Как пользователь получает результат запроса к базе данных при работе с интерактивным SQL?

☐ результат получает прикладная программа
⊞ результат выдается непосредственно пользователю после выполнения каждого оператора
☐ результат выводится в нужном пользователю виде
☐ результат выдается непосредственно пользователю после выполнения всей последовательности операторов

Вариант 3.
Как пользователь не может работать с интерактивным SQL?

⊞ Вставлять текст на языке SQL в прикладную программу
☐ Вводить непосредственно запрос на языке SQL
⊞ Обращаться к языку SQL из прикладной программы
☐ Вводить последовательно несколько запросов к базе данных на языке SQL

Задача 7. Как осуществляется работа с встроенным статическим SQL?

Вариант 1.
Как пользователь работает с встроенным статическим SQL?

⊞ вставляет текст на языке SQL в прикладную программу
☐ вводит непосредственно запрос на языке SQL
☐ текст запроса формируется прикладной программой
☐ вводит последовательно несколько запросов к базе данных на языке SQL

Вариант 2.
Как пользователь получает результат запроса к базе данных при работе с встроенным статическим SQL?

⊞ результат получает прикладная программа
☐ результат выдается непосредственно пользователю после выполнения каждого оператора
⊞ результат обрабатывается прикладной программой и выводится в нужном пользователю виде
☐ результат выдается непосредственно пользователю после выполнения всей последовательности операторов запроса

Вариант 3.
Как пользователь не может работать с встроенным статическим SQL?

☐ вставлять текст на языке SQL в прикладную программу
⊞ вводить непосредственно запрос на языке SQL
☐ обращаться к языку SQL из прикладной программы
⊞ формировать текст запроса работой прикладной программы

Задача 8. Как динамически осуществляется работа с SQL?

Вариант 1.
Как пользователь работает с встроенным динамическим SQL?

☐ вставляет текст на языке SQL в прикладную программу
☐ вводит непосредственно запрос на языке SQL
⊞ текст запроса формируется прикладной программой
☐ вводит последовательно несколько запросов к базе данных на языке SQL

Вариант 2.
Как пользователь получает результат запроса к базе данных при работе с встроенным динамическим SQL?

⊞ результат получает прикладная программа
☐ результат выдается непосредственно пользователю после выполнения каждого оператора
⊞ результат обрабатывается прикладной программой и выводится в нужном пользователю виде
☐ результат выдается непосредственно пользователю после выполнения всей последовательности операторов запроса

Вариант 3.
Как пользователь может работать с SQL, в условиях необходимости динамического формирования запроса во время выполнения прикладной программы?

☐ вставлять текст на языке SQL в прикладную программу
⊞ использовать интерфейсы вызова подпрограмм
☐ обращаться к языку SQL из прикладной программы
⊞ формировать текст запроса работой прикладной программы

Литература

1. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е изд. СПб.: Питер, 2003.
2. Грабер М. SQL. Справочное руководство. – М: Лори, 1997. – 291с.
3. Грофф Дж., Вайнберг П. SQL: полное руководство: Пер. с англ. – К.: Издательская группа BHV, 2000. – 608 с.
4. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для вузов. – СПб.: КОРОНА принт, 2000. – 416 с.
5. Реализация баз данных Microsoft SQL Server 7.0 Учебный курс: официальное пособие Microsoft для самостоятельной подготовки. М.: Издательско-торговый дом «Русская редакция», 2000.
6. Карпова Т. Базы данных. Модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.

Лекция 12. Основные операторы языка SQL. Интерактивный SQL

В лекции дается общая характеристика операторов языка SQL, используемых, в частности, для работы с базой данных в интерактивном режиме (создание таблиц, выбор информации из таблиц, добавление, удаление и модификация элементов). Приводятся примеры запросов к базе данных на языке SQL.

Ключевые термины: язык SQL, интерактивный SQL, интерактивный режим работы с SQL, операторы SQL, select, insert, update,delete.

Цель лекции: дать общую характеристику операторов языка SQL и показать, как записываются основные запросы к базе данных на языке SQL (в интерактивном режиме).

12.1. Общее представление об основных операторах языка SQL

Как уже отмечалось в лекции 11, все операторы языка SQL разделяются на три составные части: DDL – язык определения данных, DCL – язык управления данными, DML – язык обработки данных.

Приведем примеры основных операторов из вышеуказанных частей (без описания синтаксиса). Описание синтаксиса операторов SQL можно посмотреть в многочисленных книгах по языку SQL, в меню «Справка» конкретных СУБД.

Операторы разграничения доступа пользователей к объектам базы данных (DCL).

GRANT – создание в системе безопасности записи, разрешающей пользователю работать с данными или выполныять определенные операции SQL

DENY - создание в системе безопасности записи, заперщающей доступ для определенной учетной записи.

Операторы определения данных (язык DDL).

Соответствующие операторы предназначены для создания, удаления, изменения основных объектов модели данных релюционных СУБД: таблиц, представлений, индексов.

CREATE TABLE <имя> - создание новой таблицы в базе данных.

DROP TABLE <имя> - удаление таблицы из базы данных.

ALTER TABLE<имя> - изменение структуры существующей таблицы или ограничений целостности, задаваемых для данной таблицы.

При выполнении аналогичных операций с представлениями или индексами в указанных операторах вместо служебного слова TABLE записывается слово VIEW (представление) или слово INDEX (индекс)

Операторы манипулирования данными (язык DML).

Операторы DML работают с базой данных и используются для изменения данных и получения необходимых сведений.

SELECT – выборка строк, удовлетворяющих заданным условиям. Оператор реализует, в частности, такие операции реляционной алгебры как «селекция» и «проекция».

UPDATE – изменение значений определенных полей в строках таблицы, удовлетворяющих заданным условиям.

INSERT – вставка новых строк в таблицу.

DELETE – удаление строк таблицы, удовлетворяющих заданным условиям. Примерение этого оператора учитывает принципы поддержки целостности, поэтому он не всегда может быть выполнен корректно.

12.2 Интерактивный режим работы с SQL (интерактивный SQL)

Соответствующий режим предусматривает непосредственную работу пользователя с базой данных по следующему алгоритму: используя прикладную программу (клиентское приложение) или стандартную утилиту, входящую в СУБД, пользователь:

- устанавливает соединение с БД (подтверждая наличие прав доступа);
- вводит соответствующий оператор SQL, при необходимости в режиме диалога вводит дополнительную информацию;
- инициирует выполнение команды.

Текст запроса поступает в СУБД, которая:

- осуществляет синтаксический анализ запроса (проверяет, является ли запрос корректным);
- проверяет, имеет ли пользователь право выполнять подобный запрос (например, пользователь, у которого определены права только на чтение, пытается что-то удалить);
- выбирает, каким образом осуществлять выполнение запроса – план выполнения запроса;
- выполняет запрос;
- результат выполнения отсылает пользователю.

Схема взаимодействия пользователя и СУБД с использованием интерактивного SQL приводится на рис. 12.1.

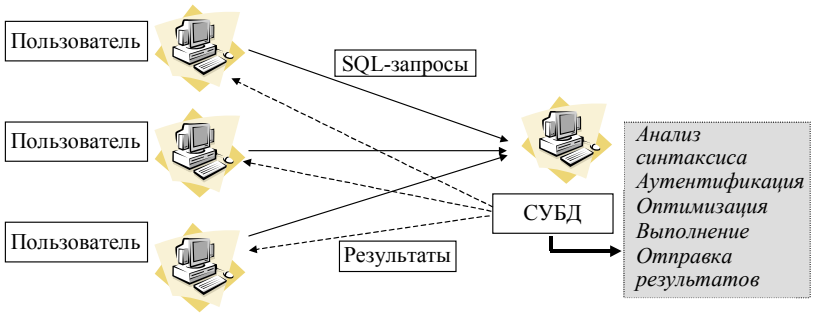


Рис. 12.1. Схема работы интерактивного SQL

12.3. Использование языка SQL для выбора информации из таблицы

Выборка данных осуществляется с помощью оператора SELECT, который является самым часто используемым оператором языка SQL. Синтаксис оператора SELECT имеет следующий вид:

```
SELECT [ALL/DISTINCT] <список атрибутов>/*
FROM <список таблиц>
[WHERE <условие выборки>]
[ORDER BY <список атрибутов>]
[GROUP BY <список атрибутов>]
[HAVING <условие>]
[UNION<выражение с оператором SELECT>]
```

В квадратных скобках указываются элементы, которые могут в запросе отсутствовать.

Ключевое слово ALL означает, что результатом будут все строки, удовлетворяющие условию запроса, в том числе и одинаковые строки. DISTINCT означает, что в результирующий набор не включаются одинаковые строки. Далее идет список атрибутов исходной таблицы, которые будут включены в таблицу-результат. Символ* означает, что в таблицу-результат включаются все атрибуты исходной таблицы.

Обязательным ключевым словом является слово FROM, за ним следуют имена таблиц, к которым осуществляется запрос.

В предложении с ключевым словом WHERE задаются условия выборки строк таблицы. В таблицу-результат включаются только те строки, для которых условие, указанное в предложении WHERE, принимает значение истина.

Ключевое слово ORDER BY задает операцию упорядочения строк таблицы-результата по указанному списку атрибутов.

В предложении с ключевым словом GROUP BY задается список атрибутов группировки (разъяснение этого и последующего ключевого слова будет представлено немного позднее).

В предложении HAVING задаются условия, накладываемые на каждую группу.

Отдельно отметим, что ключевые слова FROM, WHERE ORDER BY используются аналогичным образом и в других операторах манипулирования данными языка SQL.

Рассмотрим реализацию запросов для конкретного примера, представленного в лекции 8 (см. рис. 8.1)

Выдать список всех студентов.

```
SELECT *
FROM student
```

или

```
SELECT id_st, surname
FROM student
```

Заметим, что если добавить к данному запросу предложение ORDER BY surname, то список будет упорядочен по фамилии. По умолчанию подразумевается, что сортировка про-

изводится по возрастанию. Если необходимо упорядочение по убыванию, после имени атрибута добавляется слово DESC.

Выдать список оценок, которые получил студент с кодом «1».

```
SELECT id_st, mark
FROM mark_st
Where id_st = 1
```

Выдать список кодов студентов, которые получили на экзаменах хотя бы одну двойку или тройку.

В предложении WHERE можно записывать выражение с использованием арифметических операторов сравнения (<, >, и т.д.) и логических операторов (AND, OR, NOT) как и в обычных языках программирования.

```
SELECT id_st, mark
FROM mark_st
WHERE ( MARK >= 2 ) AND ( MARK <= 3 )
```

Наряду с операторами сравнения и логическими операторами для составления условий в языке SQL (из-за специфики области применения) существуют ряд специальных операторов, которые, как правило, не имеют аналогов в других языках. Вот эти операторы:

IN – вхождение в некоторое множество значений;

BETWEEN – вхождение в некоторый диапазон значений;

LIKE – проверка на совпадение с образцом;

IS NULL – проверка на неопределенное значение.

Оператор IN используется для проверки вхождения в некоторое множество значений. Так, запрос

```
SELECT id_st, mark
FROM mark_st
WHERE mark IN (2,3)
```

дает тот же результат, что и вышеуказанный запрос (выведет идентификаторы всех абитуриентов, получивших хотя бы одну двойку или тройку на экзаменах).

Того же результата можно добиться, используя оператор BETWEEN:

```
SELECT id_st, mark
FROM mark_st
WHERE mark BETWEEN 2 AND 3
```

Выдать список всех студентов, фамилии которых начинаются с буквы А.

В этом случае удобно использовать оператор LIKE.

Оператор LIKE применим исключительно к символьным полям и позволяет устанавливать, соответствует ли значение поля образцу. Образец может содержать специальные символы:

_ (символ подчеркивания) – замещает любой одиночный символ;

% (знак процента) – замещает последовательность любого числа символов.

```
SELECT id_st, surname
FROM student
WHERE surname LIKE 'A%'
```

Очень часто возникает необходимость произвести вычисление минимальных, максимальных или средних значений в столбцах. Так, например, может понадобиться вычислить средний балл. Для осуществления подобных вычислений SQL предоставляет специальные агрегатные функции:

MIN – минимальное значение в столбце;
MAX – максимальное значение в столбце;
SUM – сумма значений в столбце;
AVG – среднее значение в столбце;
COUNT – количество значений в столбце, отличных от NULL.

Следующий запрос считает среднее среди всех баллов, полученных студентами на экзаменах.

```
SELECT    AVG(mark)
FROM  mark_st
```

Естественно, можно использовать агрегатные функции совместно с предложением WHERE:

```
SELECT AVG(mark)
FROM  mark_st
WHERE id_st = 100
```

Данный запрос вычислит средний балл студента с кодом 100 по результатам всех сданных им экзаменов.

```
SELECT  AVG(mark)
FROM  mark_st
WHERE id_ex = 10
```

Данный запрос вычислит средний балл студентов по результатам сдачи экзамена с кодом 10.

В дополнение к рассмотренным механизмам язык SQL предоставляет мощный аппарат для вычисления агрегатных функций не для всей таблицы результатов запроса, а для разных значений по группам. Для этого в SQL существует специальная конструкция GROUP BY, предназначенная для указания того столбца, по значениям которого будет производиться группировка. Так, например, мы можем вычислить средний балл по всем экзаменам для каждого студента. Для этого достаточно выполнить следующий запрос:

```
SELECT  id_st, AVG(mark)
FROM mark_st
GROUP BY  id_st
```

Все это, как обычно, может быть совмещено с предложением WHERE. При этом, не вдаваясь в тонкости выполнения запроса внутри СУБД, можно считать, что сначала выполняется выборка тех строк таблицы, которые удовлетворяют условиям из предложения WHERE, а потом производится группировка и агрегирование.

Приведем запрос, который вычисляет средний балл по оценкам, полученным на экзамене с кодом 100, для каждого студента.

```
SELECT  id_st, AVG(mark)
FROM  mark_st
WHERE id_ex = 100
GROUP BY  id_st
```

Заметим, что группировка может производиться более чем по одному полю.

Для запросов, содержащих секцию GROUP BY существует важное ограничение: такие запросы могут включать в качестве результата столбцы, по которым производится группировка, и столбцы, которые содержат собственно результаты агрегирования.

Для того чтобы форматировать вывод, существуют различные возможности SQL. Так, например, допустимым является включение текста в запрос. Рассмотрим пример того, как это делается:

```
SELECT  'Средний балл=', AVG(mark)
FROM  mark_st
WHERE  id_ex = 10
```

В результате данного запроса пользователь увидит не просто некоторое число, а число, сопровожденное поясняющим текстом.

12.4. Использование SQL для выбора информации из нескольких таблиц

До сих пор мы рассматривали выбор информации из единственной таблицы. Можно запрашивать информацию из нескольких таблиц, реализуя описанные в соответствующем разделе учебника реляционные операции. Стоит упомянуть, что полное рассмотрение темы выходит за рамки данного учебника. Подробно этот вопрос можно изучить при помощи, например, [1, 2]. Рассмотрим некоторые примеры того, как это делается.

Как правило, в тех случаях когда возникает необходимость выбирать информацию из разных таблиц, они тем или иным образом связаны друг с другом, например отношениями один к многим или один к одному по некоторому полю.

Еще раз вернемся к примеру из лекции 8. Рассмотрим соответствующую ER-диаграмму (рис. 12.2.).

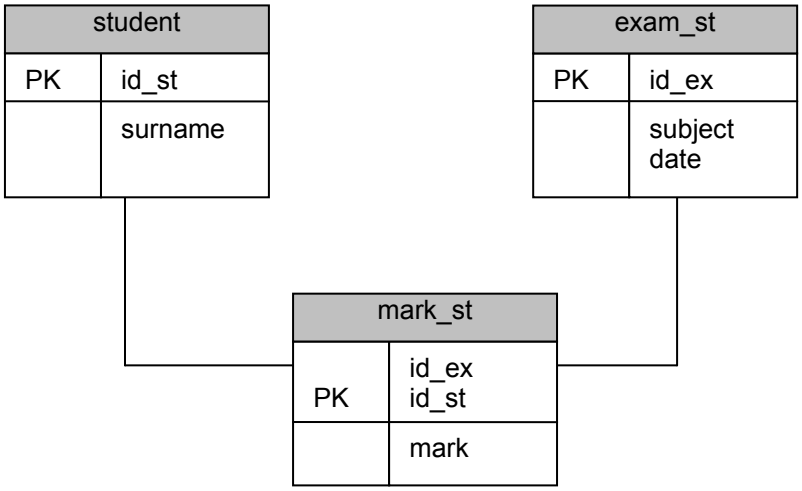


Рис. 12.2. Пример связанных таблиц

В этом примере тоже присутствуют связанные таблицы. Рассмотрим таблицы student, mark_st и exam_st.

Таблица mark_st связана с таблицей exam_st по полю id_ex.

Таблица mark_st связана с таблицей student по полю id_st.

Допустим, требуется распечатать список студентов с оценками, которые они получили на экзаменах. Для этого необходимо выполнить следующий запрос:

```
SELECT student.surname, mark_st.id_ex, mark_st.mark
FROM student, mark_st
WHERE  student.id_st = mark_st.id_st
```

Отметим следующие изменения по сравнению с запросами к одной таблице.

1. В секции FROM указаны две таблицы.
2. Так как таблиц стало больше одной, появилась некоторая неоднозначность при упоминании полей. Так, во многих случаях неизвестно, из какой таблицы из списка FROM брать поле. Для устранения неоднозначности имена полей указываются с префиксом – именем таблицы. Имя таблицы от имени поля отделяется точкой.
3. В предложении WHERE указано условие соединения таблиц.

Нетрудно заметить, что использование префиксов-имен таблиц сильно загромождает запрос. Для того чтобы избежать подобного загромождения, используются псевдонимы. Так, можно переписать предыдущий запрос следующим образом:

```
SELECT E.surname, M.id_ex, M.mark
FROM student E, mark_st M
WHERE E.id_st = M. id_st
```

12.5. Использование SQL для вставки, редактирования и удаления данных в таблицах

Для добавления данных в таблицу в стандарте SQL предусмотрена команда INSERT. Рассмотрим ряд примеров запросов.

```
INSERT INTO mark_st
VALUES (1, 2, 5)
```

Данный запрос вставляет в таблицу mark_st строку, содержащую значения, перечисленные в списке VALUES. Если не нужно указывать значение какого-то поля, можно присвоить ему NULL:

```
INSERT INTO mark
VALUES (1, 2, NULL)
```

В случае если необходимо использование для некоторых полей значений по умолчанию, SQL позволяет явно указать, какие поля необходимо заполнить конкретными данными, а какие – значениями по умолчанию:

```
INSERT INTO mark_st (id_st, id_ex)
VALUES (1, 2)
```

Для удаления данных из таблицы существует команда DELETE:

```
DELETE
FROM student
```

Этот запрос удаляет все данные из таблицы student.

Можно ограничить диапазон удаляемой информации следующим образом:

```
DELETE
FROM student
WHERE surname > 'И'
```

Для обновления данных используется команда UPDATE.

```
UPDATE    mark_st
SET mark = '5'
WHERE id_st = 100 AND id_ex = 10
```

При помощи этого запроса изменится на «5» оценка у студентас кодом 100 по экзамену с кодом 10.

12.5. Язык SQL и операции реляционной алгебры

Язык SQL является средством выражения мощного математического аппарата теории множеств и реляционной алгебры. В данном разделе рассматривается связь операторов языка SQL с операциями реляционной алгебры и теории множеств.

Операция объединения

Средствами языка SQL операция объединения представляется следующим образом:

```
SELECT *
FROM A
UNION
SELECT *
FROM B
```

Операция разности

Средствами языка SQL операция разности представляется следующим образом:

```
SELECT *
FROM A
EXCEPT
SELECT *
FROM B
```

Операция проекции

```
SELECT Field_i1, … , Field_in
FROM A
```

Операция выборки (селекции)

```
SELECT *
FROM A
WHERE (<condition>)
```

Операция пересечения

```
SELECT *
FROM A
INTERSECT
SELECT *
FROM B
```

Операция соединения, эквисоединения

```
SELECT A.Field_1, … , A.Field_n, B.Field_1, … , B.Field_m
FROM A, B
WHERE (A.Field_i Θ B.Field_1)
```

Если ***Θ*** – операция «=», то это эквисоединение.

Операция естественного соединения

Пусть есть отношения A (X_1, … , X_n, A_1, …, A_m) и B (X_1, … , X_n, B_1, …, B_r).

```
SELECT A.X_1, … , A.X_n, A.A_1, … , A.A_m,  B.B_1, … , B.B_r
FROM A, B
WHERE (A.X_1 = B.X_1) AND … AND (A.X_n = B.X_n)
```

Краткие итоги: В лекции дается общая характеристика операторов языка SQL, используемых, в частности, для работы с базой данных в интерактивном режиме (создание таблиц, выбор информации из таблиц, добавление, удаление и модификация элементов). Дается понятие интерактивного режима работы с SQL. Рассматриваются основные операторы SQL, используемые для манипулирования данными (выбор информации из таблиц, добавление, удаление и модификация элементов). Приводятся примеры записи запросов к базе данных на языке SQL с использованием операторов select, insert, update,delete. Рассматривается связь между операциями реляционной алгебры и операторами языка SQL.

Более подробно материалы лекции рассматриваются в [1-7].

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Основные группы операторов SQL?

Вариант 1.
Какие из перечисленных операторов относятся к языку управления данными (DCL)?

☐ Update - изменение значений в полях таблицы
⊞ Grant – создание в системе безопасности разрешающей записи для пользователя
☐ Select –выборка строк, удовлетворяющих заданным условиям
☐ Create – создание таблицы, индекса
☐ Drop – удаление таблицы
☐ Alter – изменение структуры таблицы
☐ Insert – вставка строк в таблицу
☐ Delete – удаление строк из таблицы
⊞ Deny - создание в системе безопасности запрещающей записи для пользователя

Вариант 2.
Какие из перечисленных операторов относятся к языку определения данными (DDL)?

☐ Update - изменение значений в полях таблицы
☐ Grant – создание в системе безопасности разрешающей записи для пользователя
☐ Select –выборка строк, удовлетворяющих заданным условиям
⊞ Create – создание таблицы, индекса
⊞ Drop – удаление таблицы
⊞ Alter – изменение структуры таблицы
☐ Insert – вставка строк в таблицу
☐ Delete – удаление строк из таблицы
☐ Deny - создание в системе безопасности запрещающей записи для пользователя

Вариант 3.
Какие из перечисленных операторов относятся к языку манипулирования данными (DML)?

☐ Update - изменение значений в полях таблицы
☐ Grant – создание в системе безопасности разрешающей записи для пользователя
⊞ Select –выборка строк, удовлетворяющих заданным условиям
☐ Create – создание таблицы, индекса
☐ Drop – удаление таблицы
☐ Alter – изменение структуры таблицы
⊞ Insert – вставка строк в таблицу
⊞ Delete – удаление строк из таблицы
☐ Deny - создание в системе безопасности запрещающей записи для пользователя

Задача 2. Какие служебные слова используются в операторе select?

Вариант 1.
Какие служебные слова обязательно присутствуют в операторе SELECT?

⊞ FROM
☐ WHERE
☐ ORDER BY
☐ GROUP BY
☐ HAVING

Вариант 2.
Какие служебные слова могут отсутствовать в операторе SELECT?
☐ FROM
⊞ WHERE
⊞ ORDER BY
⊞ GROUP BY
⊞ HAVING

Вариант 3.
После каких служебных слов указывается список атрибутов в операторе SELECT?

☐ FROM
☐ WHERE
⊞ ORDER BY
⊞ GROUP BY
☐ HAVING

Задача 3. Как формируется условие выборки записей?

Вариант 1.
Какие служебные слова определяют условие выборки записей?

☐ FROM
⊞ WHERE
☐ ORDER BY
☐ GROUP BY
☐ HAVING
☐ SELECT

Вариант 2.
Какие служебные слова не определяют условие выборки записей?

⊞ FROM
☐ WHERE
⊞ ORDER BY
⊞ GROUP BY
⊞ HAVING
⊞ SELECT

Вариант 3.
Какие операторы и операнды могут использоваться при формировании условия выборки записей?

☐ названия таблиц
⊞ имена атрибутов
⊞ имена атрибутов с указанием имен соответствующих таблиц
⊞ арифметические операторы сравнения
⊞ логические операторы
⊞ числовые константы
⊞ символьные константы

Задача 4.

Вариант 1.
Какие элементы таблицы выбираются оператором SELECT?

☐ только строки
☐ только столбцы
⊞ строки и столбцы
☐ вся таблица

Вариант 2.
После какого служебного слова в операторе SELECT указывается выбор столбцов?

☐ FROM
☐ WHERE
☐ ORDER BY
☐ GROUP BY
☐ HAVING
⊞ SELECT

Вариант 3.
После какого служебного слова в операторе SELECT указывается выбор строк?

☐ FROM
⊞ WHERE
☐ ORDER BY
☐ GROUP BY
☐ HAVING
☐ SELECT

Задача 5. Как осуществляется выборка информации из нескольких таблиц?

Вариант 1.
В каких предложениях оператора SELECT необходимо использовать имена таблиц при выборке информации из нескольких таблиц?

⊞ FROM
⊞ WHERE
☐ ORDER BY
☐ GROUP BY
☐ HAVING
⊞ SELECT

Вариант 2.
Какие предложения оператора SELECT используются для установления связи между строками таблиц при выборке информации из нескольких таблиц?

☐ FROM
⊞ WHERE
☐ ORDER BY
☐ GROUP BY
☐ HAVING
☐ SELECT

Вариант 3.
Как указываются имена атрибутов в операторе SELECT при выборке информации из нескольких таблиц?

☐ указываются только имена атрибутов через запятую
☐ указываются имена атрибутов через запятую и имена таблиц через запятую
☐ указываются имена таблиц через запятую и имена атрибутов через запятую
⊞ указывается имя таблицы и через точку имя атрибута и т. д.

Задача 6. Характеристика оператора INSERT.

Вариант 1.
Что делает оператор INSERT?

☐ вставляет строку с заданными значениями элементов в таблицу
☐ вставляет столбец с заданными значениями элементов в таблицу
⊞ вставляет строку с заданными значениями элементов и значениями по умолчанию в таблицу
☐ вставляет столбец с заданными значениями элементов и значениями по умолчанию в таблицу

Вариант 2.
В каких предложениях оператора INSERT указываются вставляемые в таблицу значения?

☐ INSERT
⊞ VALUES
☐ FROM
☐ WHERE

Вариант 3.
Какие служебные слова могут использоваться в операторе INSERT?

☐ FROM
☐ WHERE
⊞ VALUES
☐ GROUP BY

Задача 7. Характеристика оператора DELETE.

Вариант 1.
Какие служебные слова могут использоваться в операторе DELETE?

⊞ FROM
⊞ WHERE
☐ VALUES
☐ GROUP BY

Вариант 2.
В каких случаях оператор DELETE не может быть выполнен корректно?

☐ пользователь пытается удалить не ту строку, которую нужно удалить
☐ удаляемая строка ссылается на строку другой таблицы
⊞ на удаляемую строку имеется ссылка из другой таблицы
⊞ нарушаются условия целостности

Вариант 3.
С помощью какого предложения оператора DELETE может указываться удаляемая строка?

☐ FROM
⊞ WHERE
☐ DELETE
☐ SET

Задача 8. Как связаны операторы языка SQL с операциями реляционной алгебры?

Вариант 1.
Какой оператор языка (или служебное слово языка) реализует операцию проекции реляционной алгебры?

☐ INSERT
⊞ SELECT
☐ ORDER BY
☐ GROUP BY
☐ HAVING

Вариант 2.
Какой оператор языка (или служебное слово языка) реализует операцию селекции реляционной алгебры?

☐ INSERT
⊞ SELECT
☐ ORDER BY
☐ GROUP BY
☐ HAVING

Вариант 3.
Какой оператор языка (или служебное слово языка) используются при представлении операции естественного соединения реляционной алгебры?

⊞ FROM
⊞ WHERE
☐ ORDER BY
☐ GROUP BY
☐ HAVING
⊞ SELECT

Литература

1. Горев А., Ахаян Р., Макашарипов С. Эффективная работа с СУБД. СПб.: Питер, 1997. – 700 с.
2. Грабер М. SQL. Справочное руководство. – М: Лори, 1997. – 291с.
3. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е изд. СПб.: Питер, 2003.
4. Грофф Дж., Вайнберг П. SQL: полное руководство: Пер. с англ. – К.: Издательская группа BHV, 2000. – 608 с.
5. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных: Учебник для вузов. – СПб.: КОРОНА принт, 2000. – 416 с.
6. Реализация баз данных Microsoft SQL Server 7.0 Учебный курс: официальное пособие Microsoft для самостоятельной подготовки. М.: Издательско-торговый дом «Русская редакция», 2000.
7. Карпова Т. Базы данных. Модели, разработка, реализация. – СПб.: Питер, 2001. – 304 с.

Лекция 13. Использование языка SQL в прикладных программах

В лекции рассматриваются разные технологии формирования запросов на языке SQL в прикладных программах (статическое формирование запроса, динамическое формирование запроса, использование библиотек).

Ключевые термины: язык SQL, встроенный SQL, статический SQL, динамический SQL, интерфейс программирования приложений, API, библиотеки для работы с СУБД.

Цель лекции: показать основные возможности формирования запросов к базе данных из прикладных программ.

13.1. Программный (встроенный) SQL

Основная работа с базой данных проводится с использованием прикладных программ, из которых и идут запросы к базам данных. В этом случае интерактивный режим работы не может быть использован, текст SQL- запроса должен быть либо включен в прикладную программу (если запрос полностью определен заранее), либо формироваться в процессе работы прикладной программы.

Программный SQL предназначен для того, чтобы встраивать SQL-запросы в прикладную программу, написанную на одном из языков программирования. При этом возникают следующие вопросы:

- компилятор с алгоритмического языка должен иметь возможность выделения в тексте прикладной программы последовательность операторов SQL.
- компилятор должен объединять возможности языка программирования высокого уровня (переменные, ветвления, циклы) и возможности SQL (запросы на языке, близком к естественному).

Решение этих проблем частично описано в стандарте SQL.

Рассмотрим алгоритм выполнения SQL-запросов в интерактивном режиме работы. Легко видеть, что пользователь вынужден ожидать результатов выполнения запроса в течение всего времени работы реализации SQL-запроса. Если через некоторое время пользователю снова нужно будет выполнить тот же самый запрос, СУБД вновь проделает те же самые действия, что и при предыдущем обращении. Налицо некоторое несовершенство механизма:

- одни и те же этапы выполняются каждый раз заново для одинаковых запросов;
- СУБД не может обрабатывать интерактивные запросы с опережением.

Решение подобных проблем очевидно – часть действий по обработке запроса необходимо выполнять один раз, сохранять результат в некотором виде, а потом использовать столько раз, сколько необходимо. Эта идея является одной из основных идей программного SQL. Таким образом, программный SQL позволяет:

- использовать операторы интерактивного SQL в тексте программы на языке программирования высокого уровня;

- наряду с операторами интерактивного SQL использовать новые специальные конструкции, дополняющие SQL и увеличивающие его возможности;
- для передачи параметров в запрос использовать в тексте запроса переменные, объявленные в программе;
- для возврата в программу результатов запроса использовать специальные конструкции, отсутствующие в интерактивном SQL;
- осуществлять компиляцию запросов совместно с программой, обеспечивая впоследствии согласованную работу программы и СУБД. Заранее (на этапе компиляции) выполнять действия по анализу и оптимизации запросов, экономя время, затрачиваемое на этапе выполнения программы.

На настоящий момент используются три варианта встраивания запросов на языке SQL в прикладную программу (программного SQL): статический SQL, динамический SQL и метод, основанный на различных интерфейсах программирования приложений (API). Рассмотрим соответствующие варианты.

13.2. Статический SQL

Статический SQL – разновидность программного SQL, предназначенная для встраивания SQL-операторов в текст программы на языке программирования высокого уровня.

Основная особенность статического SQL определяется его названием: встраиваемые запросы должны быть четко определены на стадии написания прикладной программы, так как именно конкретный текст запросов вставляется в прикладную программу.

Рассмотрим два основных этапа, связанных с работой статического SQL, – компиляция программы и работа (выполнение) программы.

Схема компиляции и сборки программы выглядит следующим образом (рис. 13.1):

- Программа, включающая операторы языка программирования высокого уровня (ЯПВУ) вместе с операторами SQL, подается на вход специального препроцессора, который выделяет из нее части, связанные с SQL.
- Вместо инструкций встроенного SQL препроцессор подставляет вызовы специальных функций СУБД. Библиотеки таких функций для связи с языками программирования существуют для всех распространенных серверных СУБД. Стоит особо отметить, что эти библиотеки имеют «закрытый» интерфейс, т.е. разработчики библиотеки могут менять его по своему усмотрению, соответственно обновив препроцессор. Все это говорит о том, что программист не должен вмешиваться в этот процесс.
- Сами инструкции SQL препроцессор выделяет в отдельный файл.
- Программа поступает на вход обычного компилятора языка программирования, после чего получаются объектные модули. Далее эти объектные модули вместе с библиотеками СУБД собираются в один исполняемый модуль – приложение.
- Наряду с этими операциями происходит работа с файлом, содержащим SQL-инструкции. В литературе этот модуль часто носит название «модуль запросов к ба-

зе данных» (Database Request Module, DBRM) [1]. Обработку этого модуля осуществляет специальная утилита, которая обычно носит название BIND. Для каждой инструкции SQL утилита выполняет следующие действия:

- осуществляет синтаксический анализ запроса (проверяет, является ли запрос корректным);
- проверяет, существуют ли в базе данных те объекты, на которые ссылается запрос;
- выбирает, каким образом осуществлять выполнение запроса – план выполнения запроса;

- Все планы выполнения запросов сохраняются в СУБД для последующего использования.

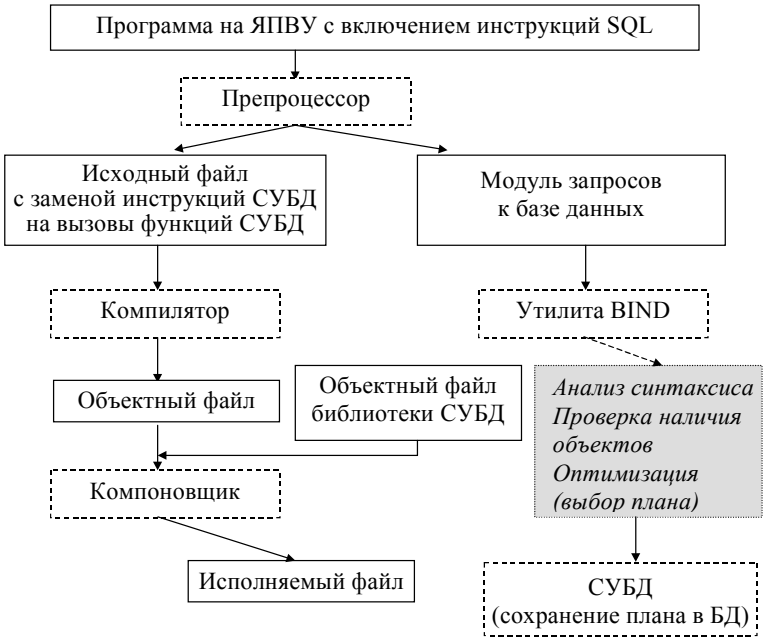


Рис. 13.1. Схема компиляции программы с встроенными инструкциями статического SQL

Схема выполнения программы выглядит следующим образом (рис. 13.2.):

Программа запускается на выполнение обычным образом. При необходимости выполнить запрос программой осуществляется вызов специальной функции СУБД, которая отыскивает уже сформированный ранее план выполнения запроса. СУБД выполняет запрос в соответствии с выбранным планом. Результат выполнения запроса поступает в приложение.

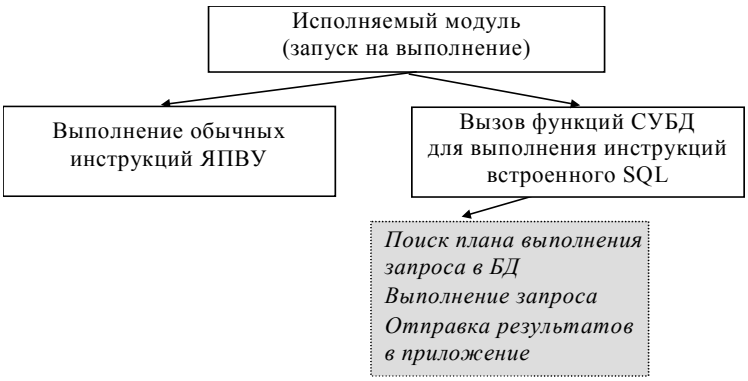


Рис. 49. Схема выполнения программы с встроенными инструкциями статического SQL

Для реализации вышеуказанных схем статический SQL должен содержать дополнительные операторы (по сравнению с интерактивным SQL), позволяющие компилятору выделить в тексте программы SQL-запросы, объявлять используемые в этих запросах таблицы, объявлять переменные для обработки ошибок, как результатов реализации запросов и т. п. Основные команды статического SQL приводятся в следующей таблице.

Таблица 13.1.

Основные команды статического SQL

EXEC SQL	Спецификатор, указывающий, что следующая за ним инструкция является инструкцией встроенного SQL
;	В языке С – признак окончания инструкции встроенного SQL
DECLARE TABLE	Объявляет таблицу, которая потом будет использоваться в инструкциях встроенного SQL
SQLCODE	Переменная для обработки ошибок
SQLSTATE	Переменная для обработки ошибок
GET DIAGNOSTICS	Инструкция для обработки ошибок
WHENEVER **SQLERROR** **SQLWARNING** **NOT FOUND** **GOTO** **CONTINUE**	Набор совместно используемых инструкций для упрощения обработки ошибок
BEGIN DECLARE SECTION **END DECLARE SECTION**	Инструкции для определения области, в которой будут объявлены переменные, впоследствии используемые в запросах SQL
INTO	Используется в операторе

	SELECT для указания переменной, в которую необходимо поместить результат выполнения запроса
DECLARE CURSOR	**Курсор** – специальный инструмент, предназначенный для обработки результатов запроса, содержащих более одной строки. Работа с курсором похожа на работу с файлами. Данная инструкция служит для создания курсора и связывания его с конкретным запросом
OPEN	Команда, открывающая курсор и побуждающая СУБД начать выполнение запроса. Устанавливает курсор перед первой строкой результата запроса
FETCH	Команда, перемещающая указатель текущей строки (курсор) на следующую строку. В некоторых СУБД и стандарте SQL-92 реализованы разные формы команды FETCH, перемещающие курсор на произвольную строку результатов запроса
CLOSE	Закрывает курсор и прекращает доступ к результатам запроса

Использование описанной выше схемы компиляции/сборки/выполнения программы позволяет:

- использовать SQL совместно с программой на языке программирования высокого уровня;
- заранее осуществлять проверку синтаксиса запросов и оптимизацию их выполнения (выбор плана). Понятно, что проверка синтаксиса выполняется быстро, но выбор плана выполнения – весьма трудоемкая процедура. Тот факт, что она выполняется один раз на этапе компиляции, позволяет говорить о существенном уменьшении накладных расходов.

Однако статическая разновидность программного SQL имеет некоторые существенные ограничения. Так, переменные в запросах могут использоваться только в тех местах, где в запросах обычно стоят константы. Например, нельзя задавать имя таблицы, из которой производится выборка, а также названия столбцов, как параметр. В связи с этим при использовании статического варианта вложенного (программного) SQL необходимо на этапе написания программы точно знать состав запросов, которые необходимо будет выполнять в прикладной программе. Во многих случаях это ограничение является существенным. Для его устранения была введена новая разновидность программного SQL – динамический SQL. Рассмотрим кратко основные идеи динамического SQL.

13.3. Динамический SQL

Динамический SQL – разновидность программного SQL, предназначенная для встраивания SQL-операторов в текст программы на языке программирования высокого уровня, допускающая динамическое формирование и выполнение запросов во время работы программы.

История возникновения динамического SQL во многом связана с компанией IBM, внедрившей этот мощный инструмент в свою СУБД DB2. Стандарты SQL, в частности SQL-1, не поддерживали динамического SQL. Лишь в 1992 году в стандарт SQL-2 были включены спецификации динамического SQL. Основной концепцией динамического SQL является следующее утверждение: встроенная инструкция SQL не записывается в исходный текст программы, вместо этого программа формирует текст инструкции во время выполнения в одной из своих областей данных, а затем передает сформированную инструкцию в СУБД для динамического выполнения [1].

Напомним, что при использовании статического SQL: схема реализазации подразумевала два этапа – компиляцию программы и выполнение программы. При этом на этап компиляции ложилась основная нагрузка. Именно здесь решались вопросы проверки, разбора и оптимизации запросов, поскольку запрос был заранее известен. Совершенно очевидно, что подобную двухэтапную схему нельзя реализовать для динамического SQL, так как на этапе компиляции программы запрос неизвестен. Поэтому проверку, разборку и оптимизацию запросов здесь приходится выполнять непосредственно во время работы программы. Таким образом, если эти операции в статическом SQL выполнялись во время компиляции один раз, то в динамическом SQL они будут выполняться столько раз для одного запроса, сколько раз он будет сформирован в процессе работы прикладной программы. Это определяет существенный недостаток динамического SQL– низкую производительность по сравнению со статическим. Достоинство динамического SQL в том, что он позволяет формировать запрос к базе данных во время работы программы, реагируя на те или иные произошедшие события. Такая возможность является жизненно важной для клиент-серверной и трехзвенной архитектур, в которых структура базы данных и деловые правила имеют тенденцию к изменению, что требует определенной гибкости при организации процесса обработки данных.

Учитывая относительно низкую производительность динамического SQL, представляется правильным, там, где только возможно, рекомендовать использование статической разновидности SQL, применяя аппарат динамического SQL где это действительно необходимо.

Динамический SQL также должен содержать дополнительные операторы (по сравнению с интерактивным SQL). Основные операторы динамического SQL приводятся в следующей таблице.

Таблица 13.2.

Основные команды динамического SQL

EXECUTE IMMEDIATE	Немедленное выполнение инструкции

PREPARE	Подготовка инструкции к выполнению
EXECUTE	Выполнение подготовленной ранее инструкции
DESCRIBE	Специальная команда, участвующая при возврате результата выполнения инструкций динамического SQL
DECLARE CURSOR	Разновидность инструкции DECLARE CURSOR, применявшейся ранее в рамках статического SQL, содержащая вместо запроса его имя (связанное с запросом при помощи инструкции PREPARE)
OPEN FETCH CLOSE	Разновидности инструкций для работы с курсором в динамическом SQL

Рассмотрим схему функционирования динамического SQL (рис.13.3). Схема предусматривает одноэтапное и двухэтапное выполнение инструкций.

Одноэтапное выполнение инструкций осуществляется командой EXECUTE IMMEDIATE.

Рис. 13.3.. Схема выполнения программы со встроенными инструкциями динамического SQL с применением одноэтапной схемы

Схема выполнения инструкции подразумевает:

- динамическое формирование команды SQL в строковом виде во время работы программы;
- передачу строкового вида инструкции в СУБД при помощи команды EXECUTE IMMEDIATE;

- выполнение инструкции системой управления БД, включающее синтаксический анализ, проверку параметров, оптимизацию (выбор плана) и выполнение этого плана.

Основные проблемы одноэтапной схемы заключаются в том, что она не позволяет выполнять инструкции SELECT (ибо нет средств для возврата в приложение результатов запроса) и приводит к нерациональному расходованию вычислительных ресурсов (т.к. при повторном выполнении той же инструкции вновь будет затрачено время на все те же действия по ее интерпретации и выполнению).

Двухэтапное выполнение инструкций основано на следующем соображении: скорее всего, команда динамического SQL в таком виде, как она поступает на выполнение, будет выполняться неоднократно. При этом могут меняться какие-то конкретные детали. А это значит, что инструкцию можно параметризовать. Использование параметризованных инструкций позволяет сделать схему выполнения двухэтапной, разделив процесс на «подготовку инструкции» и «выполнение инструкции» (рис. 13.4.).

Рис. 13.4. Схема выполнения программы со встроенными инструкциями динамического SQL с применением двухэтапной схемы

На этапе подготовки можно осуществить синтаксический анализ инструкции, интерпретировать ее и подготовиться к выполнению, выбрав план выполнения.

На этапе выполнения СУБД подставляет значения параметров (полученные из программы) и использует сформированный ранее план выполнения для достижения результата.

При этом реализуется идея однократного выполнения тех действий, которые можно выполнить один раз. Так, подготовленная один раз инструкция может быть выполнена десятки раз с разными параметрами.

13.4. Интерфейсы программирования приложений (API). DB-Library, ODBC, OCI, JDBC

Как замечено выше, программный SQL отличается от обычной, интерактивной формы наличием некоторых специальных инструкций, а также механизмом трансляции и выполне-

ния запросов. Таким образом, для применения программного SQL в тексте своих программ программистам необходимо ознакомиться с некоторым специфическим набором инструкций. Стоит заметить, что в разных СУБД эти наборы инструкций, вообще говоря, могут несколько отличаться друг от друга. В результате возникает некоторая проблема, связанная с непереносимостью программы.

Наряду с описанным выше механизмом существует и активно применяется еще один подход, связанный с наличием специальных интерфейсов – API (application programming interface – интерфейс программирования приложений). Эти API представляют собой библиотеки функций, разработанные для обеспечения связи прикладной программы с СУБД посредством выполнения SQL-запросов. Прикладная программа вызывает специальные функции библиотеки для передачи в СУБД SQL-запроса в текстовом виде и для получения результатов выполнения запросов, а также различной служебной информации.

Применение подобного подхода приводит к тому, что программистам более не требуется изучать специальные наборы инструкций SQL, а необходимо лишь изучить специальную библиотеку функций. С учетом того, что механизм использования API является широко используемым и стандартным подходом (чего только стоит использование мощного аппарата Windows API), для специалистов нет ничего нового в изучении еще одной библиотеки, в данном случае – для общения с СУБД.

Кроме этого, программа, содержащая вызовы некоторых функций специализированной библиотеки, ничем не отличается по схеме компиляции и выполнения от обычной программы. Так, подобная программа не требует применения специализированного препроцессора с механизмом раздельной компиляции. Может показаться, что подход, связанный с использованием библиотек API, является наиболее прогрессивным, на самом же деле такой вывод вряд ли верен. Так, на настоящий момент очень активно используются и динамический SQL и библиотеки API. В каждом из этих подходов существуют свои достоинства, недостатки и границы разумной применимости. Как обычно, выбор того, каким из подходов воспользоваться, лежит на административной группе разработчиков базы данных, которая принимает решения в зависимости от особенностей конкретной задачи и имеющихся специалистов.

В данном разделе рассматривается подход, основанный на интерфейсе программирования приложений.

Посмотрим, как работают прикладные программы, использующие различные API. Принципы работы разных библиотек аналогичны. Схема работы приложения совместно с SQL API выглядит следующим образом [1]:

- программа получает доступ к базе данных путем вызова одной или нескольких API-функций, подключающих программу к СУБД и к конкретной базе данных;
- для пересылки инструкций SQL в СУБД программа формирует инструкцию в виде текстовой строки и затем передает эту строку в качестве параметра при вызове API-функции;
- программа вызывает выполнение API-функции для проверки состояния переданной в СУБД инструкции и обработки ошибок;

- если инструкция SQL представляет собой запрос на выборку, то, вызывая API-функции, программа записывает результаты запроса в свои переменные; обычно за один вызов возвращается одна строка или столбец данных;
- свое обращение к базе данных программа заканчивает вызовом API-функции, отключающей ее от СУБД.

Из имеющихся для реализации SQL-запросов интерфейсов API на настоящий момент выделилось несколько библиотек, «стандартных» в том смысле, что они активно применяются множеством разработчиков по всему миру. Данное пособие не является подробным руководством по всем этим библиотекам. Более того, для их профессионального освоения необходимо серьезное изучение соответствующей литературы. В рамках данного курса мы лишь приведем обзор этих библиотек, рассмотрим основные заложенные в них идеи. Подробно эти SQL API описаны, например в [1].

Протокол ODBC

ODBC (Open Database Connectivity – открытый доступ к базам данных) – разработанный компанией Microsoft универсальный интерфейс программирования приложений для доступа к базам данных [1].

Основной целью разработки протокола ODBC считается стандартизация механизмов взаимодействия с различными СУБД. Основная проблема, связанная с разработкой приложений, взаимодействующих с базами данных на основе специальных SQL API, состояла в том, что каждая СУБД имела собственный программный интерфейс доступа, каждый из них имел свои особенности и функционировал не совсем так, как другие. В связи с этим разработка приложения существенно зависела от используемой СУБД. Компания Microsoft сделала важный шаг для решения этой проблемы. Основная идея заключалась в разработке универсального интерфейса на уровне семейства операционных систем Windows, который мог бы быть поддержан в разных СУБД.

Рассмотрим кратко структуру программного обеспечения ODBC [1]:

- **интерфейс вызовов функций ODBC**: это так называемый верхний уровень ODBC, содержащий API, который и используется непосредственно приложениями. Данный API реализован в виде библиотеки динамической компоновки Dll и входит в состав операционной системы Windows;
- **драйверы ODBC**: это так называемый нижний уровень ODBC, содержащий набор драйверов для СУБД, поддерживающих протокол ODBC. В рамках технологии для каждой СУБД может быть разработан соответствующий ODBC-драйвер, который будет являться промежуточным звеном между прикладной программой и СУБД, транслируя вызовы функций СУБД в вызовы внутренних специализированных функций СУБД. Таким образом решается проблема стандартизации. Для многих современных СУБД существуют специализированные драйверы ODBC, отдельно устанавливаемые в операционную систему;
- **диспетчер драйверов ODBC**: данный программный механизм представляет средний уровень ODBC, управляя процессом загрузки необходимых драйверов.

Схема выполнения программы с использованием протокола ODBC для доступа к данным приводится на рис.13.5.

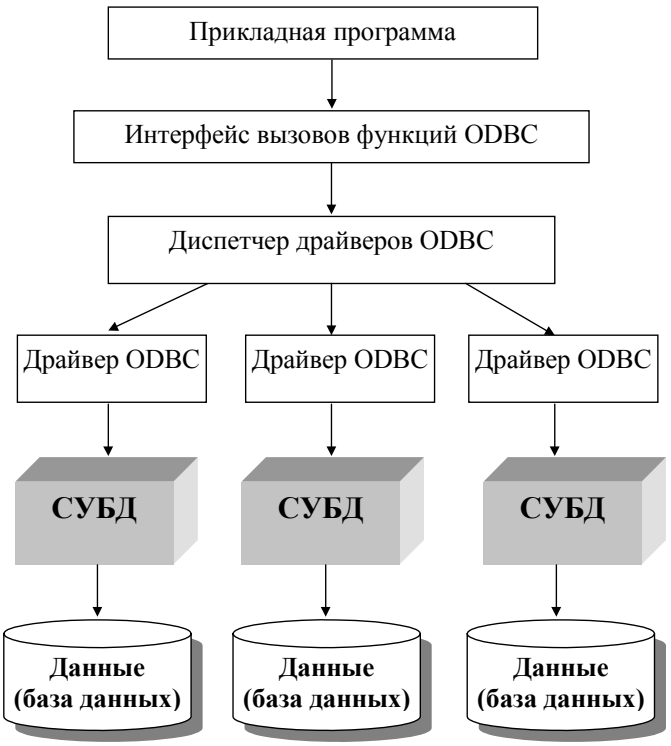


Рис. 13.5. Схема выполнения программы с использованием протокола ODBC для доступа к данным

Перечень некоторых базисных функций ODBC API приводится в следующей таблице.

Таблица 13.3.

Базисные функции ODBC API

Назначение	**Функция**	**Описание**
Соединение с источником данных	SQLAlocEnv	Получает указатель окружения. Одно окружение может служить для создания нескольких соединений.
	SQLAlIoc Connect	Получает указатель соединения.
	SQLConnect	Соединяется с указанным драйвером, используя имя источника данных, идентификатор пользователя и пароль.
Подготовка SQL запросов	SQLAllocStmt	Размещает указатель выражения.
	SQLPrepare	Подготавливает SQL выражение для дальнейшего использования.
	SQLGet CursorName	Возвращает имя, связанное с указателем выражения.

	SQLSet CursorName	Устанавливает имя курсора.
Выполнение запросов	SQLExecute	Выполняет заранее подготовленный запрос.
	SQLExec Direct	Выполняет запрос.
Выборка результатов и информации о результатах	SQLRow Count	Возвращает количество записей, задействованных в операциях вставки, удаления, модификации.
	SQLNum ResultCol	Возвращает количество колонок в выбранном наборе данных.
	SQLDescribe Col	Описывает колонку в выбранном наборе данных.
	SQLCol Attributes	Описывает атрибуты колонки в выбранном наборе данных.
	SQLBindCol	Присваивает место в памяти для колонки в выбранном наборе данных и указывает ее тип данных.
	SQLFetch	Возвращает несколько наборов данных.

Протокол JDBC

JDBC (Java Database Connectivity) представляет собой API для выполнения SQL-запросов к базам данных из программ, написанных на языке Java [1].

Рассмотрим основные принципы JDBC.

С развитием глобальных сетей, в частности Интернета, и всех сопутствующих технологий стали появляться новые языки, специально предназначенные для работы в новых условиях. Одним из таких языков является язык программирования Java. В настоящее время Интернет-приложения занимают существенное место на рынке, работая в рамках 2-, 3- и многозвенной архитектуры. При этом значение языка Java как средства создания приложений, работающих с базами данных, существенно возрастает. Именно это и явилось одной из основных причин разработки нового программного интерфейса – JDBC. Первоначально интерфейс JDBC был разработан компанией Sun Microsystems, в настоящий момент этот API поддерживается всеми ведущими коммерческими СУБД.

Известно несколько различных версий JDBC. Так, версия 1.0 содержала некоторые средства доступа к данным:

- диспетчер драйверов (для подключения к разным СУБД);
- механизм управления сеансами (для одновременной работы с несколькими СУБД);
- механизм передачи инструкций SQL на выполнение в СУБД;
- механизм работы с курсорами (для передачи результатов выполнения запросов из СУБД в приложение).

Этот перечень определенным образом напоминает аналогичный функциональный аппарат протокола ODBC.

Версия JDBC 2.0 содержит существенные отличия. Так, вследствие увеличения возможностей интерфейса было проведено его идеологическое разделение на две основные части: Core API (основные возможности) и Extensions API (так называемые расширения).

В [1] указаны следующие возможности JDBC:

- **Пакетные операции**. Программа на Java может осуществить обновление базы данных в пакетном режиме, т.е. одна функция JDBC может добавить в базу данных несколько записей, что положительно сказывается на производительности программ.
- **Курсоры с произвольным доступом**. В JDBC 2.0 существует средство, позволяющее перемещаться по результатам запроса произвольным образом.
- **Обновляемые курсоры**. В JDBC 2.0 курсоры, наряду с функцией возврата результата запроса, используются и при обновлении базы данных. Обновления производятся при добавлении или изменении одной из строк в результатах запроса.
- **Организация связного пула.** Несколько программ на языке Java могут пользоваться совместным доступом к базе данных, уменьшая затраты на подключения к базе данных и отключения от нее. Данный перечень можно продолжить (распределенные транзакции, поддержка JNDI и т.д.).

Версия JDBC 3.0 появилась совсем недавно и содержит такие новации, как объектно-реляционные расширения SQL и улучшенные механизмы обработки транзакций. Архитектура JDBC берет свое начало от ODBC и в существенной части повторяет ее, поэтому схема выполнения программы на Java с использованием протокола JDBC для доступа к данным полностью аналогична схеме на рис. 13.5 (слова ODBC заменяются на слова JDBC). В отличие от ODBC, драйверы JDBC подразделяются на четыре типа. Основные отличия между этими типами связаны с местонахождением API СУБД (на клиентской или серверной СУБД) и способом доступа к базе данных (через собственный API СУБД или через ODBC).

Библиотека DB-Library

Библиотека DB-Library реализует интерфейс программирования приложений для совместной работы с широко распространенной СУБД Microsoft SQL Server. Данная библиотека является весьма обширной и содержит более 100 функций. Основными из них являются:

dblogin(); dbopen() – подключение к БД;

dbopen(); dbexit() – установка/разрыв соединения с БД;

dbcmd() – передача инструкции (пакета инструкций) SQL в СУБД в текстовом виде;

dbSQLexec() – требование к СУБД выполнить текущий пакет инструкций;

dbcancel() – прекращение выполнения пакета инструкций SQL;

dbresults() – получение результатов выполнения очередной инструкции SQL в текущем пакете;

dbbind(), dbdata(), dbnextrow(), dbnumcols(), dbdatlen() и др. – обработка результатов запросов на выборку данных.

Логика работы прикладной программы, обрабатывающей данные, хранящиеся в базе данных под управлением Microsoft SQL Server, выглядит следующим образом:

- при помощи указанных выше функций (dblogin(), dbopen()) прикладная программа формирует сведение об авторизации и пытается установить соединение с СУБД;
- при помощи СУБД программа открывает конкретную базу данных, с которой будет происходить работа (dbopen());
- при помощи специальной функции (dbcmd()) программа передает в СУБД текст SQL-инструкции, которую далее необходимо будет выполнить; в библиотеке DB-Library поддерживается так называемый пакетный режим работы. Данный режим подразумевает возможность создания пакетов инструкций. Так, вызывая функцию dbcmd() несколько раз, вы можете передать в СУБД текст нескольких команд SQL, которые впоследствии будут выполнены как одна команда;
- используя функцию dbSQLexec(), программа вызывает выполнение инструкций, переданных ранее при помощи вызовов функций dbcmd();
- вызывая функцию dbresults(), программа может определить, удалось ли СУБД выполнить очередную инструкцию (как правило, число вызовов dbresults() соответствует числу инструкций в очередном пакете);
- в случае если мы имеем дело с запросом, возвращающим набор строк в качестве результата (запросом на выборку), программа при помощи вызовов функции dbbind() осуществляет связывание каждого поля результатов запроса с некоторой областью оперативной памяти. Далее при помощи функции dbnextrow() программа выполняет переход к следующей строке результатов запроса, что приводит к помещению в буфер новых данных;
- при помощи функции dbexit() программа разрывает соединение с базой данных. Библиотека DB-Library представляет собой большой и сложный механизм. Так, в библиотеке предусмотрены специальные механизмы обработки ошибок, разные способы передачи результатов выполнения запросов в прикладную программу и т.д.

Краткие итоги: В лекции рассматриваются разные технологии формирования запросов на языке SQL в прикладных программах (программный SQL). Дается понятие статического SQL, динамического SQL и приводятся соответствующие основные операторы. Рассматриваются интерфейсы программирования приложений (API) (протокол ODBC, протокол JDBC, библиотека DB-Library).

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Что такое программный SQL?

Вариант 1.
Особенности программного SQL по сравнению с интерактивным

☐ используются принципиально другие операторы
☐ пользователь пишет программу на языке SQL
⊞ могут использоваться те же операторы SQL
⊞ запрос на языке SQL встраивается в программу на алгоритмическом языке

Вариант 2.
Какие специальные конструкции (дополнительные операторы) должны быть в программном SQL?

☐ оператор цикла
☐ оператор ветвления
⊞ оператор, определяющий начало запроса на языке SQL
⊞ оператор, определяющий окончание запроса на языке SQL
⊞ операторы, указывающие дополнительные переменные для обработки результатов запроса
☐ операторы, определяющие передачу управления от программы на алгоритмическом языке программе на языке SQL

Вариант 3.
За счет чего последовательность запросов, встроенных в прикладную программу будет выполняться быстрее, чем та же последовательность запросов, выполняемая в интерактивном режиме?

⊞ запросы могут выполняться с опережением
⊞ запросы компилируются один раз
⊞ заранее выполняются действия по анализу и компиляции запросов
⊞ компьютер не ожидает ответных действий пользователя

Задача 2. Как происходит работа с программой при использовании статического SQL?

Вариант 1.
Как должен выглядеть запрос на языке в прикладной программе в этом случае?

⊞ должно быть указано фактическое имя таблицы
☐ имя таблицы может быть указано как параметр
⊞ должны быть указаны фактические имена атрибутов
☐ имена атрибутов могут быть указаны как параметры
☐ в запросах могут широко использоваться переменные
⊞ переменные могут использоваться в запросах только на месте констант

Вариант 2.
Как компилируется и компонуется прикладная программа с встроенным запросом на языке SQL?

☐ прикладная программа компилируется вместе с текстом запроса
⊞ текст запроса компилируется отдельно
☐ сформированный модуль запроса вставляется в модуль прикладной программы
⊞ в модуль прикладной программы вставляется вызов функции СУБД

Вариант 3.
Как происходит выполнение прикладной программы?

⊞ переход из прикладной программы к запросу осуществляется вызовом специальной функции
☐ скомпилированная вместе с текстом запроса прикладная программа автоматически выполняется
⊞ при неоднократном выполнении одного и того же запроса используется один и тот же программный модуль
☐ при каждом выполнении одного и того же запроса используются разные программные модули

Задача 3. Характеристика команд статического SQL

Вариант 1.
Какие операторы могут быть использованы в статическом SQL?

⊞ SELECT
⊞ DELETE
⊞ INSERT
⊞ DECLARE TABLE
⊞ EXEC SQL
⊞ OPEN

Вариант 2.
Какие специальные операторы могут быть использованы в статическом SQL?

☐ SELECT
☐ DELETE
☐ INSERT
⊞ DECLARE TABLE
⊞ EXEC SQL
⊞ GET DIAGNOSTIC
⊞ DECLARE CURSOR

Вариант 3.
Какие специальные операторы могут быть использованы в статическом SQL для указания начала и конца SQL- запроса?

☐ DECLARE TABLE
⊞ EXEC SQL
☐ GET DIAGNOSTIC
⊞ ; (точка с запятой)
☐ END DECLARE

Задача 4. Особенности динамического SQL по сравнению со статическим ?

Вариант 1.
Как должен выглядеть запрос на языке в прикладной программе в этом случае?

- ☐ запрос должен быть определен в тексте прикладной программы
- ⊞ запрос формируется во время работы прикладной программы
- ⊞ имя таблицы может быть указано как параметр
- ⊞ имена атрибутов могут быть указаны как параметры
- ⊞ в запросах могут широко использоваться переменные
- ☐ переменные могут использоваться в запросах только на месте констант

Вариант 2.
Как компилируется и компонуется прикладная программа при использовании динамического SQL?

- ☐ прикладная программа компилируется вместе с текстом запроса
- ☐ текст запроса компилируется отдельно один раз
- ☐ сформированный модуль запроса вставляется в модуль прикладной программы
- ⊞ в модуль прикладной программы вставляется вызов функции СУБД
- ⊞ текст запроса компилируется столько раз, сколько раз запрос формируется прикладной программой

Вариант 3.
Как происходит выполнение прикладной программы при использовании динамического SQL?

- ⊞ переход из прикладной программы к запросу осуществляется вызовом специальной функции
- ☐ скомпилированная вместе с текстом запроса прикладная программа автоматически выполняется
- ☐ при неоднократном выполнении одного и того же запроса используется один и тот же программный модуль
- ⊞ при каждом выполнении одного и того же запроса используются разные программные модули

Задача 5. Характеристика команд динамического SQL

Вариант 1.
Какие операторы могут быть использованы в динамическом SQL?

⊞ SELECT
⊞ DELETE
⊞ INSERT
☐ DECLARE TABLE
☐ EXEC SQL
⊞ OPEN
⊞ DECLARE CURSOR
⊞ PREPARE
⊞ EXECUTE

Вариант 2.
Какие специальные операторы могут быть использованы в динамическом SQL?

☐ SELECT
☐ DELETE
☐ INSERT
☐ DECLARE TABLE
☐ EXEC SQL
☐ GET DIAGNOSTIC
⊞ DECLARE CURSOR
⊞ PREPARE
⊞ EXECUTE

Вариант 3.
Какие специальные операторы могут быть использованы в динамическом SQL для подготовки и выполнения SQL- запроса?

☐ DECLARE TABLE
☐ EXEC SQL
☐ GET DIAGNOSTIC
⊞ PREPARE
⊞ EXECUTE

Задача 6. Характеристика интерфейсов программирования приложений (API).

Вариант 1.
Чем удобны интерфейсы программирования приложений?

☐ не требуется изучать алгоритмический язык программирования
⊞ не требуется изучать специальные инструкции статического и динамического SQL
⊞ соответствующий подход может применяться с использованием разных языков программирования
☐ не требуется изучать язык SQL

Вариант 2.
Как компилируется прикладная программа, использующая интерфейсы программирования приложений?

⊞ прикладная программа компилируется вместе с вызовом функций библиотек
☐ вызов функций библиотек компилируется отдельно
☐ сформированный модуль запроса вставляется в модуль прикладной программы
⊞ в модуль прикладной программы вставляется вызов функции библиотеки

Вариант 3.
Как выполняется программа с использованием интерфейсов программирования приложений?

☐ параметрами функций библиотеки интерфейсов программирования приложений являются имена таблиц, атрибутов и константы
⊞ параметрами функций библиотеки интерфейсов программирования приложений являются тексты SQL- запросов
⊞ переход из прикладной программы к запросу осуществляется вызовом специальной функции
☐ скомпилированная вместе с текстом запроса прикладная программа автоматически выполняется
⊞ при неоднократном выполнении одного и того же запроса используется один и тот же программный модуль
☐ при каждом выполнении одного и того же запроса используются разные программные модули

Задача 7. Что такое протокол ODBC?

Вариант 1.
Какова цель создания протокола ODBC?

- ☐ создание интерфейса с конкретной СУБД
- ☐ создание универсального интерфейса с СУБД
- ⊞ создание универсального интерфейса с СУБД на уровне конкретной операционной системы
- ☐ создание библиотеки функций для обеспечения связи прикладной программы и СУБД

Вариант 2.
Что такое драйверы ODBC?

- ☐ программа- интерфейс между прикладной программой на алгоритмическом языке и вызовом функции API
- ⊞ программа- интерфейс между вызовом функции API и программой, реализующей функции конкретной СУБД
- ☐ программа- интерфейс между прикладной программой на алгоритмическом языке и программой, реализующей функции конкретной СУБД
- ☐ программа- интерфейс между прикладной программой на алгоритмическом языке и программой, реализующей функции любой СУБД

Вариант 3.
Для чего в этом протоколе используются драйверы?

- ☐ для сокращения времени реализации запроса
- ⊞ для создания возможности использования этого протокола в разных СУБД
- ☐ для удобства разработки прикладных программ
- ☐ для упрощения текста запроса к базе данных

Задача 8. Что такое протокол JDBC?

Вариант 1.
Какова цель создания протокола JDBC?

- ☐ создание интерфейса с конкретной СУБД
- ☐ создание универсального интерфейса с СУБД
- ☐ создание универсального интерфейса с СУБД на уровне конкретной операционной системы
- ⊞ создание библиотеки функций для обеспечения связи прикладной программы и СУБД
- ⊞ создание интерфейса программы, написанной на определенном алгоритмическом языке, с СУБД

Вариант 2.
Что такое драйверы JDBC?

- ☐ программа-интерфейс между прикладной программой на определенном алгоритмическом языке и вызовом функции API
- ⊞ программа-интерфейс между вызовом функции API и программой, реализующей функции конкретной СУБД
- ☐ программа-интерфейс между прикладной программой на алгоритмическом языке и программой, реализующей функции конкретной СУБД
- ☐ программа-интерфейс между прикладной программой на алгоритмическом языке и программой, реализующей функции любой СУБД

Вариант 3.
Для чего в этом протоколе используются драйверы?

- ☐ для сокращения времени реализации запроса
- ⊞ для создания возможности использования этого протокола в разных СУБД
- ☐ для удобства разработки прикладных программ
- ☐ для упрощения текста запроса к базе данных
- ☐ для создания возможности обращения к функциям API из программы, написанной на языке Java

Литература

1. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е изд. СПб.: Питер, 2003.
2. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Н.Новгород: Изд-во ННГУ, 2004. 271 с.

ЛЕКЦИЯ 14. НАПРАВЛЕНИЯ РАЗВИТИЯ БАЗ ДАННЫХ

В лекции рассматриваются перспективные направления в теории и практике создания баз данных – объектно-ориентированные и распределенные базы данных, а также новое направление в аналитической обработке данных = хранилища данных.

Ключевые термины: направления развития баз данных, объектно-ориентированные базы данных, системы управления объектно-ориентированными базами данных, объект, класс, методы класса, наследование, системы управления объектно-реляционными базами данных, распределенные базы данных, системы управления распределенными базами данных, хранилища данных.

Цель лекции: выделить основные черты в новых направлениях развития теории и практики создания баз данных (новые свойства, присущие объектно-ориентированным и распределенным базам данных) и хранилищ данных.

14.1. Объектно-ориентированный подход к организации баз данных

В начале 90-х годов XX века начались активные попытки по внедрению объектно-ориентированных технологий в отрасль проектирования и разработки баз данных. Бытовала точка зрения о том, что соответствующие технологии быстро вытеснят все остальные, так же как и во многих других программистских отраслях, но ничего подобного не произошло.

Объектно-ориентированное программирование

Рассмотрим термин «объектно-ориентированное программирование». Заметим, что это термин, принятый преимущественно в российской литературе. В западной литературе [2] под этим понимается сразу три аспекта:

- Объектно-ориентированный анализ – OOA, object-oriented analysis.
  Объектно-ориентированный анализ – это методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.
- Объектно-ориентированное проектирование – OOD, object-oriented design.
  Объектно-ориентированное проектирование – это методология проектирования, соединяющая в себе процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.
- Объектно-ориентированное программирование – OOP, object-oriented programming.
  Объектно-ориентированное программирование – это методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Здесь и далее по тексту условимся не отступать от традиций и понимать под объектно-ориентированным программированием (ООП) сразу три указанных выше аспекта.

Основой объектно-ориентированной технологии является так называемая объектная модель, которая возникает как результат объектно-ориентированной декомпозиции. Она выделяет основные абстракции предметной области, определяет классы абстракций и выясняет, какими данными (атрибутами) описывается каждая абстракция, какую функциональность эти абстракции должны обеспечивать. В отличие от традиционных технологий программирования объектно-ориентированная технология представляет программу как совокупность классов и объектов, взаимодействующих друг с другом.

Объект – конкретная материализация абстракции; сущность с хорошо определенными границами, в которой инкапсулированы состояние и поведение.

Объект ООП – инкапсулированная структура, имеющая атрибуты и методы.

Термин «инкапсулированная структура» означает, что объект является самодостаточным, программы, внешние по отношению к объекту, ничего «не знают» о его структуре и такое «знание» им не требуется. «Внешний» вид объекта называется его интерфейсом.

В таком понимании объект – это черный ящик, нам неизвестно, что́ у него внутри, мы лишь можем вызвать его методы и только через них взаимодействовать с ним. Кроме этого, объекты могут принадлежать иерархии «от общего к частному», которая реализуется путем наследования. Инкапсулированные состояния объекта могут быть как простыми типами данных, так и другими объектами, или даже массивами объектов. Каждый объект содержит определенную совокупность методов, классы взаимодействуют друг с другом посредством механизма сообщений. Объекты идентифицируются с помощью специальных указателей – дескрипторов. Методы объектов ООП представляют собой последовательности инструкций, выполняемых объектом. Например, у объекта может быть метод, отображающий данный объект, создающий данный объект и изменяющий его.

Предметная область моделируется как множество классов взаимодействующих объектов. Объект характеризуется набором свойств, которые являются как бы его пассивными характеристиками, и набором методов работы с этим объектом. Работать с объектом можно только с использованием его методов. Атрибуты объекта могут принимать множество допустимых значений, набор конкретных значений атрибутов определяет состояние объекта. Используя методы работы с объектом можно изменять значение его атрибутов и тем самым как бы изменить состояние самого объекта. Множество объектов с одним и тем же набором атрибутов и методов образует класс объектов. Класс, объекты которого могут служить значениями атрибута объектов другого класса, называется доменом этого атрибута.

К числу основных идей объектно-ориентированной технологии, как правило, относят [1]: **абстрагирование, инкапсуляцию, модульность, иерархичность, типизацию, полиморфизм, наследование.**

Инкапсуляция ограничивает область видимости имени атрибута пределами того объекта, в котором оно определено. Смысл этого атрибута будет определяться тем объектом, в котором оно инкапсулировано.

Полиморфизм – способность одного и того же программного кода работать с разнообразными данными. Другими словами, он допускает возможность в объектах разных типов иметь методы (процедуры или функции) с одинаковыми именами. Во время выполнения объектной программы одни и те же методы оперируют с разными объектами в зависимости от типа аргумента.

Наследование. ***Допускается порождение нового класса на основе уже существующего класса, и этот процесс называется наследованием.*** *В этом случае новый класс, называемый подклассом существующего класса, наследует все атрибуты и методы класса.* В подклассе, кроме того, могут быть определены дополнительные атрибуты и методы. Различают случаи простого и множественного наследования. В первом случае подкласс может определяться только на основе одного класса, во втором случае – на основе нескольких классов. Набор классов образует иерархическую структуру.

Объектно-ориентированные базы данных

К настоящему моменту терминология еще не устоялась, существует много разных определений и трактовок. ***Представляется, что объектно-ориентированная база данных (ООБД) – база данных, основанная на принципах объектно-ориентированной технологии.*** К основным описательным моментам, связанным с ООБД, в литературе [2] относят:

- объекты (в ООБД любая сущность – объект и обрабатывается как объект); отметим, что здесь используется понятие «объект» объектно-ориентированного программирования, которое отличается от понятия «объект», рассматриваемого в ранее в данном учебном пособии.
- классы (понятие «тип данных» реляционной модели заменяется понятиями «класс» и «подкласс»);
- наследование (классы образуют иерархию наследования, заимствуя свойства друг друга);
- атрибуты (характеристики объекта моделируются его атрибутами);
- сообщения и методы (каждый класс имеет определенную совокупность методов, классы взаимодействуют друг с другом посредством механизма сообщений);
- инкапсуляция (внутренняя структура объектов скрыта);
- идентификаторы объектов – дескрипторы.

Схема представления объекта приводится на рис. 14.1

Рис. 14.1. Схема представления объекта

Система управления объектно-ориентированной базой данных называется объектно-ориентированной СУБД (ООСУБД). Цель ООСУБД – обеспечение постоянного хранения объектов, причем в отличие от традиционной СУБД ООСУБД должна хранить в составе объекта данные и программы.

Поскольку каждый объект данного класса имеет один и тот же набор методов, методы сохраняются только один раз – как методы класса (данные каждого экземпляра объекта хранятся отдельно).

Схема представления класса объектов приводится на рис. 14.2

Рис. 14.2. Схема представления класса объектов

Используя наследование, всем объектам ПОДРАЗДЕЛЕНИЕ можно приписать свойство объекта-родителя (ФАКУЛЬТЕТ) – название факультета, номер факультета. Схема представления объектов ФАКУЛЬТЕТ и ПОДРАЗДЕЛЕНИЕ приводится на рис.14.3.

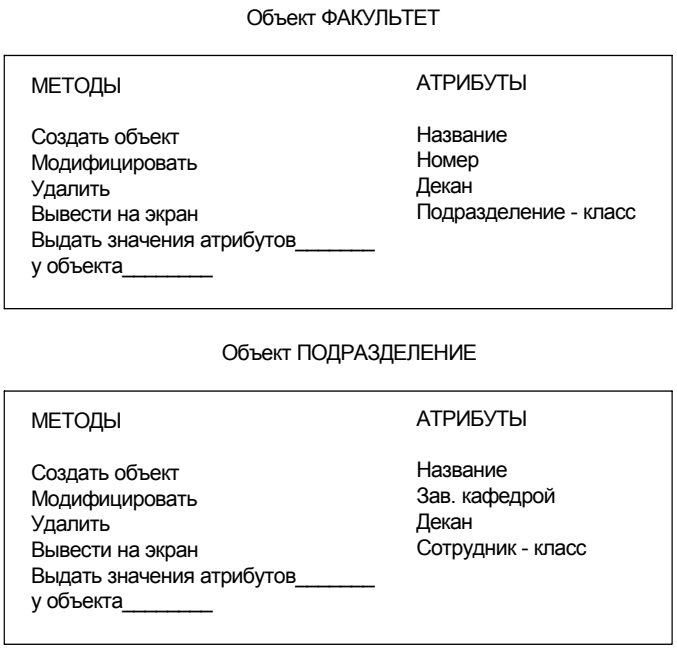


Рис. 14.3. Фрагменты представления конкретных объектов

Сравнивая объектно-ориентированный и реляционный подходы к БД, можно отметить следующие особенности. В реляционных БД (РБД) реальные объекты представляются как структуры, состоящие из набора элементарных типов данных. Такое представление имеет понятную интерпретацию – строка в плоской таблице. В том случае, когда специфика предметной области позволяет работать с такого рода приближением реальных объектов, РБД отлично справляются со своей задачей. Довольно часто реляционная модель и ее способ описания предметной области в виде набора плоских таблиц не отражают внутренней структуры для многих предметных областей, являются искусственными и становятся совершенно непонятными при увеличении количества таблиц. Основная причина несостоятельности реляционного подхода заключается в слишком сильной абстракции реального объекта, что ведет к потере семантики.

В отличие от реляционных баз данных объектно-ориентированные базы данных обладают простой и естественной связью с предметной областью, представляя ее структуру и состав, что облегчает проектирование и положительно сказывается на понимании принципов функционирования программ. Так, в сложных неоднородных предметных областях использование ООБД (в частности, там, где разные объекты имеют разные методы) должно действительно упростить процесс проектирования и разработки.

К сожалению, в ООБД существуют свои проблемы. В ООБД отсутствует универсальная модель данных, и соответственно, отсутствует мощная математическая база, как, например, в реляционной модели. В связи с этим у ООБД нет языка запросов высокого уровня, аналогичного SQL, и при доступе к данным используется мало эффективный навигационный подход. ООСУБД отличаются от реляционных СУБД тем, что программный интерфейс создания приложения либо очень слаб, либо вообще отсутствует. Это означает, для написании приложения, работающего с ООБД не существует мастеров и конструкторов (не считая, на-

пример, конструктора создания списка полей в объекте, который поставляется вместе с ООСУБД ObjectStore). Поэтому разработчик создает приложения на одном из алгоритмических языков.

По нашему мнению, существенным ограничением развития объектно-ориентированного подхода к созданию баз данных является то, что методы объекта содержатся внутри объекта и неразрывно связаны с ним. Это делает, по сути, невозможным создание для объектно-ориентированной базы данных соответствующей системы управления базой данных в традиционном понимании СУБД, функциями которой, в частности, является реализация операций обработки данных. Поэтому ООСУБД часто является не системой управления базами данных, а библиотекой программ, с помощью которой можно построить объектно-ориентированную базу данных. Примером такой библиотеки является ООСУБД ObjectStore. В связи с этим, возникает проблема реализации непредвиденных запросов.

Для перехода к объектно-ориентированным БД стандарт объектного программирования был дополнен стандартизованными средствами доступа к базам данных (стандарт ODMG 93; Object Database Management Group – группа управления объектно-ориентированными базами данных). К настоящему времени этот стандарт не реализован. Состояние проблемы подробно описано также в работах [2-6 и др.]. Отметим только, что ООБД используются, но пока не стали реальной альтернативой реляционным базам данных.

Объектно-ориентированные возможности появляются в ведущих современных СУБД, таких, как, например, Oracle. Предпринимаются попытки внесения изменений в стандарты языка SQL с целью его частичной адаптации к ООБД. Так, новый стандарт SQL-3 включает большой раздел, посвященный этому вопросу.

Объектно-реляционные СУБД

В настоящее время реляционные СУБД доминируют среди систем управления данными. Преимущества объектно-ориентированного подхода для создания сложных специализированных приложений с одной стороны, и стремление разработчиков систем управления базами данных с другой стороны расширить границы применения соответствующих СУБД обусловили ***включение объектно-ориентированных компонент (расширяемая пользователем система типов, инкапсуляция, наследование, полиморфизм и т. п.) в модель данных реляционной СУБД. Соответствующие СУБД, называемые объектно-реляционными***, соединяют в себе лучшие качества реляционных и объектно-ориентированных баз данных. Отметим, что в разных СУБД реализован разный набор из перечисленных объектно-ориентированных компонент. Таким образом, не существует общепринятой объектно-реляционной модели, а скорее имеется несколько таких моделей, поддерживающих определенный набор объектно-ориентированных компонент. Однако, основой всех таких моделей являются реляционные таблицы, используется язык запросов, включено понятие объекта, а в некоторых дополнительно реализована возможность сохранения методов в базе данных.

Соответствующие изменения реляционной модели обусловили необходимость расширения стандарта языка запросов SQL. Первый вариант такого стандарта получил название SQL3. Работа над стандартом продолжается и в настоящее время.

В качестве примера в максимальной степени объектно-ориентированной СУБД можно указать исследовательскую СУБД Postgres{3].

Отметим считающиеся объектными расширениями элементы СУБД Microsoft Server 2008.

- **Пользовательские расширения.** Пользователи имеют возможность вмешиваться в изначально предоставляемый СУБД инструментарий, создавая, в частности, новые пользовательские типы данных.
- **Хранение больших объемов данных**. Наряду с теми данными, которые хранились в БД традиционно, Microsoft SQL Server 2008 позволяет хранить в столбцах таблицы данные больших размеров (поддерживаются соответствующие типы данных).
- **Новые, ориентированные на определенные классы объектов, типы данных**. В системе определены новые типы данных (geometry, geography), характерные для тех направлений, в которых объектно-ориентированный подход весьма эффективен и часто используется (картография и соответствующие приложения, геометрическое представление объектов самой разной природы).
- **Хранимые** процедуры**.** В определенном смысле хранимые процедуры также являются объектным расширением, осуществляя необходимые пользователю воздействия на данные (стандартный для ООП процедурный подход).

14.2. Распределенные базы данных

База данных – интегрированная совокупность данных, с которой работают много пользователей. Изложение всех предыдущих разделов предполагало единую базу данных, размещаемую на одном компьютере. Напомним основные принципы, положенные в основу теории баз данных:

- централизованное хранение данных;
- централизованное обслуживание данных (ввод, корректировка, чтение, контроль целостности).

Заметим, что базы данных появились в период господства больших ЭВМ. База данных велась на одной ЭВМ, все пользователи работали именно на ЭВМ (возможные режимы работы описаны в лекции 3). Других вариантов использования вычислительной техники в то время просто не существовало. Если проанализировать работу пользователей с данными в компаниях, организациях, предприятиях в «докомпьютерное» время, то нетрудно заметить, что на отдельных участках пользователи работали со «своими» данными (осуществляли сбор определенных данных, их хранение, обработку, передачу обработанных данных на другие участки или уровни управления).

У такой технологии были существенные недостатки, которые уже отмечались в предыдущих разделах: дублирование некоторых данных, отсутствие возможности сравнительного

анализа данных всех участков. Однако у этой технологии были и существенные достоинства: данные вводились и хранились в местах их порождения; с этими данными работал пользователь, являющийся специалистом именно по этим данным, что позволяло ему вести эффективный контроль правильности данных на всех стадиях обработки; данные находились непосредственно у пользователя, что давало возможность их оперативной обработки. Централизация данных на одной ЭВМ, несомненно, дающая эффективные возможности хранения и обработки данных, не позволяла реализовывать вышеназванные достоинства.

Развитие вычислительных компьютерных сетей обусловило новые возможности в организации и ведении баз данных, позволяющие каждому пользователю иметь на своем компьютере свои данные и работать с ними и в то же время позволяющие работать всем пользователям со всей совокупностью данных как с единой централизованной базой данных. Соответствующая совокупность данных называется распределенной базой данных.

Термин «**распределенная база данных**» достаточно часто встречается в литературе [3-7]. Однако в разных источниках под этим термином понимаются совершенно разные вещи. Часть авторов понимают под распределенной базой данных то, что имеется удаленный сервер, на котором расположены данные, а также клиентские компьютеры, расположенные территориально в другом месте. Такая трактовка нам представляется неправильной. Настоящая **распределенная база данных** располагается на нескольких компьютерах. При этом часть файлов расположена на одном компьютере, часть на другом и т.д. Более того, возможна и даже часто встречается ситуация, когда информация на этих компьютерах пересекается, дублируется.

Распределенная база данных – совокупность логически взаимосвязанных разделяемых данных (и описаний их структур), физически распределенных в компьютерной сети.

Система управления распределенной базой данных – программная система, обеспечивающая работу с распределенной базой данных и позволяющая пользователю работать как с его локальными данными, так и со всей базой данных в целом.

Система управления распределенной базой данных (РаСУБД) является распределенной системой. Каждый фрагмент базы данных работает под управлением отдельной СУБД, которая осуществляет доступ к данным этого фрагмента. Пользователи взаимодействуют с распределенной базой данных через локальные и глобальные приложения. Локальные приложения дают пользователю возможность работать со своими локальными данными и не требуют доступа к другим фрагментам. Глобальные приложения дают пользователю возможность работать с другими фрагментами базы данных, расположенными на других компьютерах сети. Общая схема распределенной базы данных представлена на рис. 14.4.

Объединение данных организуется виртуально. Соответствующий подход, по сути, отражает организационную структуру предприятия (и даже общества в целом), состоящего из отдельных подразделений. Причем, хотя каждое подразделение обрабатывает свой набор данных (эти наборы, как правило, пересекаются), существует необходимость доступа к этим данным как к единому целому (в частности, для управления всем предприятием).

Одним из примеров реализации такой модели может служить сеть Интернет: данные вводятся и хранятся на разных компьютерах по всему миру, любой пользователь может получить доступ к этим данным, не задумываясь о том, где они физически расположены.

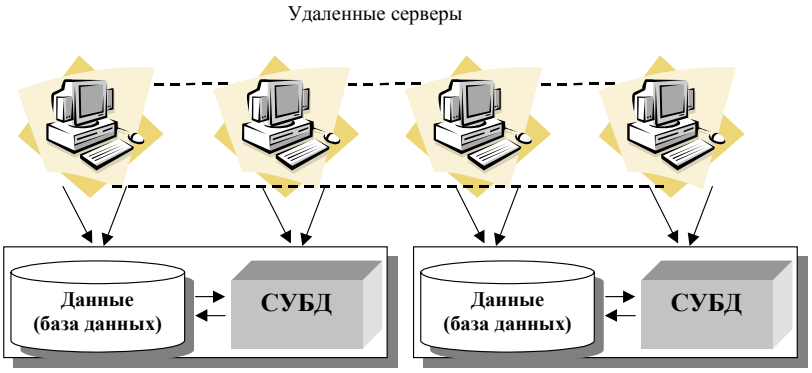


Рис. 14.4. Распределенная база данных

К.Дж. Дейт провозглашает следующий фундаментальный принцип распределенной базы данных [3]. Для пользователя распределенная система должна выглядеть точно так же, как нераспределенная. Из этого принципа следует ряд правил:

1. Локальная автономия.
2. Независимость от центрального узла.
3. Непрерывное функционирование.
4. Независимость от расположения.
5. Независимость от фрагментации.
6. Независимость от репликации.
7. Обработка распределенных запросов.
8. Управление распределенными транзакциями.
9. Независимость от аппаратного обеспечения.
10. Независимость от операционной системы.
11. Независимость от сети.
12. Независимость от СУБД.

Заметим, что понятие распределенной базы данных можно интерпретировать как следующий шаг в развитии понятий о данных (см. лекцию 1), ***обусловленный распределенностью данных в реальных предметных областях, а также новым этапом развития средств вычислительной техники – широким использованием вычислительных сетей.***

В этой интерпретации распределенную базу данных можно понимать как совокупность логически взаимосвязанных распределенных по разным компьютерам баз данных.

Перечислим основные проблемы создания распределенной базы данных.

1. Фрагментация данных и распределение по компьютерам.

2. Составление глобального каталога, содержащего информацию о каждом фрагменте БД и его местоположении в сети. (Каталог может храниться на одном узле или быть распределенным)
3. Организация обработки запросов (синхронизация нескольких запросов к одним и тем же данным, исключение аномалий удаления и обновления одних и тех же данных, расположенных на различных узлах, оптимизация последовательности шагов при обработке запроса и т.д.).

Значительным достоинством этой модели является приближение данных к месту их порождения, что позволяет существенно повысить их достоверность, недостатком – достаточно высокая сложность управления данными как единым целым.

К сожалению, процесс создания и обслуживания распределенных баз данных связан и с техническими трудностями, среди которых можно выделить жесткие требования к пропускной способности каналов связи, а также низкую производительность, обусловленную значительными затратами коммуникационных и вычислительных ресурсов при их синхронизации во время выполнения транзакций (особенно при интенсивных обращениях из разных узлов к одному фрагменту).

В задачу данного учебника не входит подробное изучение принципов построения распределенных баз данных. Интересующимся рекомендуем обратиться к соответствующей литературе, например [3-6]. Здесь мы хотим лишь обрисовать проблему и сделать некоторые выводы по перспективам ее решения. Технология, связанная с использованием распределенных баз данных, в наибольшей степени соответствует организационной человеческой деятельности (информация распределена по месту деятельности людей, и они обмениваются ей в процессе работы) и позволяет наиболее успешно решать важнейшие проблемы, ведения баз данных:

- повысить достоверность информации (информация вводится в месте ее порождения лицом, которое лучше всех понимает ее смысловое значение);
- повысить оперативность локальной обработки информации (соответствующие вопросы решаются на локальном компьютере с фрагментом базы данных).

Поэтому очевидно, что задача проектирования, создания и функционирования распределенных баз данных является весьма существенной, активно изучается в настоящее время и будет решаться и далее.

14.3. Хранилища данных

Как уже неоднократно отмечалось, технологии баз данных предназначены, как правило, для решения текущих задач обработки данных организации. В базу данных постоянно вносятся изменения, то есть база данных отражает моментальный снимок определенной области деятельности предприятия. Для эффективного принятия решений руководством при управлении организацией важно не только знать текущее положение дел, но и иметь возможность анализировать динамику (изменение во времени) основных показателей, причем,

зачастую из разных баз данных. Такую возможность дает технология так называемых хранилищ данных.

Приведем определение хранилища данных (Bill Inmon).

Хранилище данных – предметно-ориентированный, интегрированный, привязанный ко времени и неизменяемый набор данных, предназначенный для поддержки принятия решений.

Под **предметной ориентированностью** здесь понимается ориентированность на предметы (определенные группы данных), а не на конкретные приложения. Например, ориентация на данные о сотрудниках, а не только о расчете их заработной платы.

Под **интегрированностью** здесь понимается возможное объединение данных из разных источников (баз данных), имеющих разный формат и несогласованных.

Привязка ко времени предполагает, что для всех данных указан момент или промежуток времени, в который они корректны.

Данные в хранилище не изменяются, они лишь регулярно пополняются из оперативных баз данных.

Общая схема взаимодействия информационного хранилища и баз данных приводится на рис. 14.5.

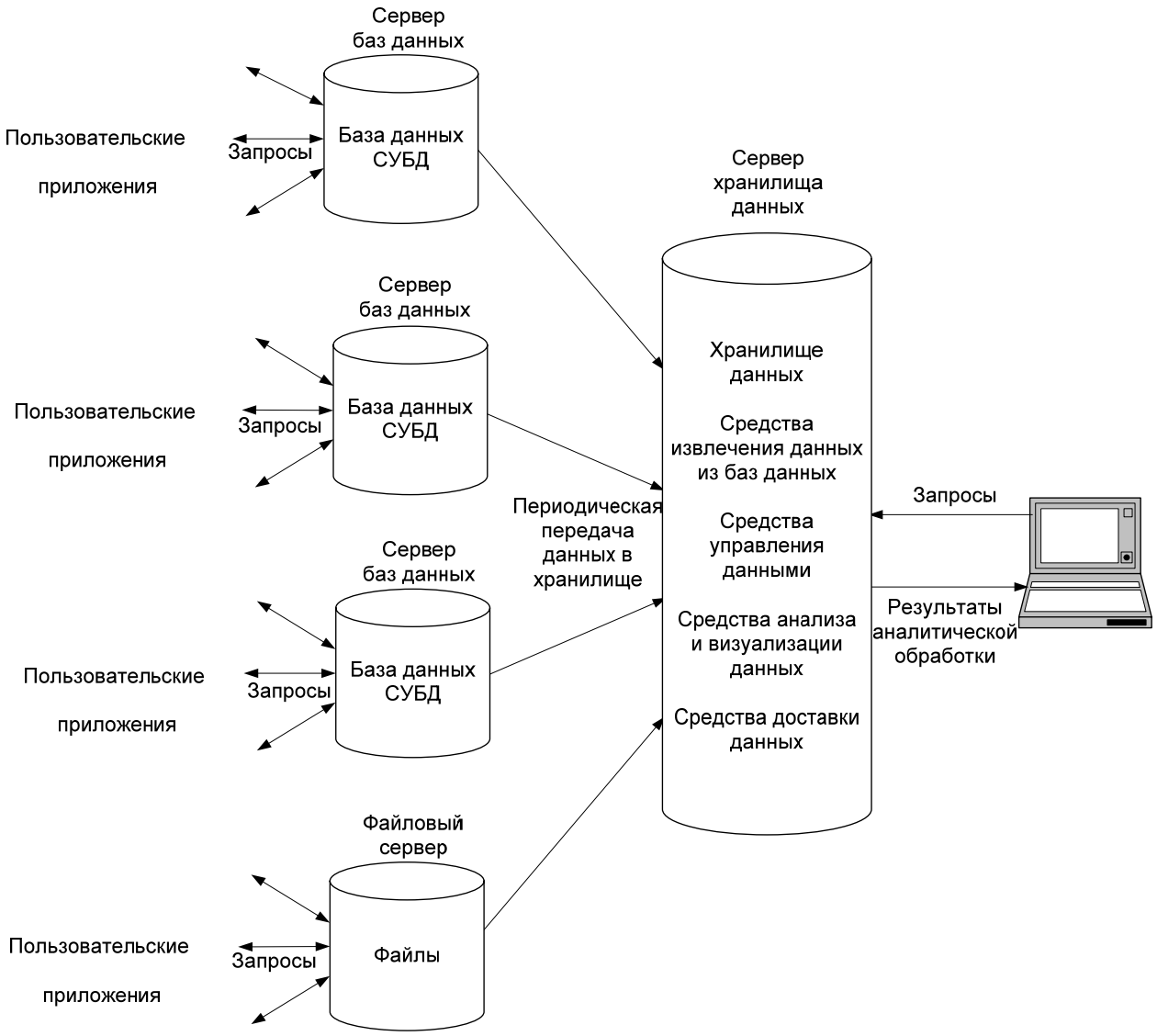


Рис. 14.5. Схема оргаизации работы хранилища данных

Еще раз подчеркнем, что основной целью хранилищ данных является бизнес-анализ или информационная поддержка принятия управленческих решений.

Для реализации всей необходимой обработки информации в соответствии с этой схемой необходимы следующие программные средства:

- средства извлечения данных из баз данных;
- средства управления данными хранилища (система управления базой данных хранилища);
- средства анализа данных хранилища (используется OLAP-технология):
- средства доставки данных;
- средства визуализации результатов обработки для конечных пользователей.

Для работы соответствующих программных средств необходимо описание структуры содержимого информационного хранилища (метаописание).

Для самого общего случая, если данные берутся из баз данных, управляемых разными СУБД, из файлов разных типов, а данные разнородны, средства управления данными хранилища пока не созданы. Однако, если данные в информационное хранилище выбираются

только из реляционных баз данных, то в качестве средств управления данными хранилища может быть взята мощная реляционная СУБД. Поэтому разработчики современных СУБД включают в состав программного обеспечения СУБД средства организации работы с хранилищами данных.

Рассмотрим в качестве **примера** ***возможности СУБД Microsoft SQL Server 2008 для организации хранилищ данных***.

Microsoft SQL Server 2008 содержит в своем составе средства извлечения, преобразования и загрузки данных (SQL Server 2008 Integration Services), способные интегрировать данные из различных источников, проверять данные на допустимость и преобразовывать перед загрузкой в хранилище. Эти средства также способствуют перемещению данных, поддерживают текстовый анализ и нечеткий поиск. Нужно отметить также среду визуальной разработки (Business Intelligence Development Studio) для создания многомерных кубов, отчетов, пакетов извлечения, преобразования и загрузки данных.

Существенной особенностью хранилищ данных является их очень большой объем. Microsoft SQL Server 2008 как средство управления данными хранилища позволяет работать с большими объемами данных, причем для сокращения времени обработки предусмотрена поддержка параллельных вычислений (путем разделения таблиц и индексов на секции о обеспечение параллельной обработки секций). В системе предусмотрена возможность сжатия данных (таблиц), что позволяет уменьшить физический размер таблиц и существенно сокращает время обмена между оперативной и внешней памятью.

В качестве средств анализа данных хранилища используется SQL Server 2008 Analysis Services, применяемый для построения многомерных кубов (многомерных моделей данных). Это средство содержит семь эффективных алгоритмов анализа данных с целью поддержки принятия управленческих решений, в том числе анализ тенденций и статистический анализ данных.

В качестве средств представления аналитических данных пользователям предлагается использовать средство генерации отчетов SQL Server 2008 Reporting Services.

Таким образом, Microsoft SQL Server 2008 является эффективным средством реализации хранилищ данных на основе реляционных баз данных.

Краткие итоги: В лекции рассмотрены перспективные направления в теории и практике создания баз данных – объектно-ориентированные и распределенные базы данных. Здесь описываются основные идеи объектно-ориентированного программирования (объект, класс, методы класса, наследование) и их приложение к теории баз данных. Отмечены основные достоинства и недостатки объектно-ориентированных баз данных.

Рассматривается понятие распределенных базы данных как следующий шаг в развитии понятий о данных. Отмечены основные достоинства распределенных баз данных и проблемы, возникающие при их разработке.

Рассматривается понятие хранилища данных, в качестве примера системы управления данными хранилища приводится СУБД Microsoft SQL Server 2008.

КОНТРОЛЬНЫЕ ТЕСТЫ

Задача 1. Какими понятиями характеризуется объектно-ориентированное программирование?

Вариант 1.
Как характеризуется объект в объектно-ориентированном программировании?

- ☐ объект в объектно-ориентированном программировании это сущность предметной области при проектировании баз данных
- ☐ объект это структура, имеющая атрибуты
- ⊞ объект это структура, имеющая свои внутренние атрибуты и методы
- ⊞ объект это сущность, характеризуемая внутренними состоянием и поведением

Вариант 2.
Какими основными понятиями характеризуется объектно-ориентированное программирование?

- ⊞ инкапсуляция
- ⊞ объект
- ⊞ наследование
- ⊞ полиморфизм
- ⊞ класс объектов
- ☐ класс связей

Вариант 3.
Какие основные принципы работы с объектами в объектно-ориентированном программировании?

- ☐ работать с объектом можно с помощью методов любых объектов
- ⊞ работать с объектом можно с помощью только его методов
- ⊞ с помощью методов можно менять значения атрибутов объекта
- ⊞ работать с классом объектов можно с помощью только методов соответствующего объекта
- ☐ работать с классом объектов можно с помощью методов любых объектов

Задача 2. Что такое объектно-ориентированная база данных?

Вариант 1.
Как характеризуется понятие объекта (сущности) в объектно-ориентированных базах данных по сравнению с традиционными базами данных?

☐ аналогично понятию объекта в традиционных базах данных
⊞ в понятие объекта включены методы объекта
⊞ используется то же понятие атрибута
☐ используется то же понятие типа данных
⊞ понятие тип данных может заменяться понятиями «класс» и « подкласс»

Вариант 2.
Какие понятия характеризуют объектно-ориентированную базу данных?

⊞ заимствование свойств класса объектов другим классом
☐ взаимодействие классов с помощью установленных связей
⊞ взаимодействие классов с помощью механизма сообщений
⊞ внутренняя структура объектов скрыта
☐ представление объекта в виде строки таблицы

Вариант 3.
Какие основные принципы работы объектно-ориентированной СУБД?

☐ те же, что и у традиционной СУБД
☐ хранит и выполняет программы обработки запросов ко всем объектам базы данных
☐ хранит и выполняет определенные программы обработки запросов к соответствующим объектам базы данных
⊞ хранит данные об объекте вместе с программами обработки этого объекта и обрабатывает соответствующие данные этими программами

Задача 3.Что такое объектно-реляционная база данных?

Вариант 1.
Что является основой объектно-реляционной базы данных?

☐ понятие объекта
⊞ реляционная таблица
☐ объектно-ориентированная реляционная таблица
☐ реляционная таблица, представляющая объект как понятие объектно-ориентированного программирования

Вариант 2.
Какие компоненты объектно-ориентированного программирования включают существующие объектно-реляционные базы данных?

⊞ ориентированные на определенные классы объектов типы данных
⊞ возможность создания новых пользовательских типов данных
☐ возможность хранения в реляционной таблице методов вместе с объектом
☐ инкапсуляцию состояния и поведения объекта

Вариант 3.
Каковы основные достоинства объектно-реляционных баз данных?

⊞ основаны на широко используемой реляционной модели
⊞ будут поддержаны стандартом языка запросов
☐ реализуют все принципы объектно-ориентированного программирования
⊞ поддерживаются известными разработчиками СУБД

Задача 4. Что такое распределенная база данных?

Вариант 1.
Как данные размещены по компьютерам в распределенной базе данных?

- ☐ общая база данных и СУБД размещены на сервере; данные, относящиеся к конкретным пользователям, размещены на их компьютерах
- ☐ общей базы данных нет, данные, относящиеся к конкретным пользователям, и СУБД размещены на их компьютерах;
- ☐ база данных разбита на части, части размещены на разных компьютерах, СУБД размещена на сервере и имеет доступ ко всем частям базы данных
- ⊞ база данных разбита на части, части базы данных и СУБД размещены на компьютерах пользователей, СУБД на каждом компьютере имеет доступ ко всем частям базы данных

Вариант 2.
Как система управления распределенной базой данных распределяется по компьютерам?

- ☐ серверная часть СУБД размещается на сервере, клиентская часть на компьютерах – клиентах
- ☐ СУБД копируется на всех компьютерах пользователей
- ☐ часть СУБД, обеспечивающая локальную работу с частью базы данных на компьютере пользователя, размещается на этом компьютере, общая часть СУБД размещается на сервере
- ⊞ часть СУБД, обеспечивающая локальную работу с частью базы данных на компьютере пользователя, размещается на этом компьютере, общая часть СУБД также размещается на этом компьютере

Вариант 3.
Как пользователь работает с распределенной базой данных?

- ☐ только с фрагментом базы данных, расположенным на его компьютере
- ☐ с любыми фрагментами базы данных, расположенных на компьютерах подразделения, в котором он работает
- ☐ только с фрагментами базы данных, расположенных на тех компьютерах, с которыми напрямую соединен его компьютер
- ⊞ с любыми фрагментами базы данных

Задача 5. Каковы основные принципы и проблемы построения распределенной базы данных?

Вариант 1.
Какие требования выдвигаются к программному обеспечению в распределенной СУБД?

☐ однотипность операционных систем всех компьютеров
☐ однотипность СУБД на всех компьютерах
⊞ управление распределенными транзакциями
⊞ возможность обработки распределенных запросов

Вариант 2.
Какие требования выдвигаются к аппаратному обеспечению в распределенной СУБД?

☐ однотипность всех компьютеров
⊞ непрерывное функционирование
⊞ независимость от компьютерной сети
⊞ независимость от расположения компьютеров

Вариант 3.
Каковы основные проблемы создания распределенной базы данных?

⊞ как распределить базу данных по компьютерам
☐ как распределить СУБД по компьютерам
⊞ как составить каталог о размещении фрагментов базы данных
⊞ как исключить одновременный доступ к одним и тем же данным
☐ как передавать данные между компьютерами

Задача 6. Что такое хранилище данных?

Вариант 1.
Какова основная цель хранилища данных?

☐ долговременное хранение данных (архив)
☐ хранение резервных копий баз данных для восстановления при машинных сбоях
⊞ хранение выборок из таблиц баз данных, привязанных к разным моментам времени, с целью их детального анализа
☐ хранение выборок из таблиц баз данных, привязанных к одному моменту времени, с целью их детального анализа

Вариант 2.
Что понимается под интегрированностью данных в хранилище?

☐ подведены итоги по разным срезам
⊞ данные объединены из разных источников
⊞ объединены данные разных форматов
⊞ объединены несогласованные данные

Вариант 3.
Как изменяются данные хранилища?

☐ корректируются
☐ частично удаляются
⊞ добавляются
☐ не изменяются

Задача 7. Как организуется работа хранилища данных?

Вариант 1.
Как загружаются данные в хранилище данных?

☐ данные вводятся пользователем в ручном режиме
☐ данные загружаются из одной базы данных один раз
⊞ данные загружаются из многих баз данных регулярно
☐ данные загружаются из одной базы данных регулярно

Вариант 2.
Как обрабатываются данные в хранилище данных?

☐ данные в хранилище обрабатываются прикладными программами пользователя
⊞ данные обрабатываются программами анализа данных хранилища и доставляются пользователю
☐ данные из хранилища доставляются пользователю и обрабатываются пользователем
☐ данные обрабатываются средствами системы управления базами данных

Вариант 3.
Какие программные средства должны поддерживать работу хранилища данных?

⊞ средства извлечения данных из баз данных;
⊞ средства управления данными хранилища
⊞ средства анализа данных хранилища
⊞ средства доставки данных
⊞ средства визуализации результатов обработки для конечных пользователей

Задача 8. Какие средства Microsoft SQL Server 2008 позволяют использовать эту СУБД для организации и работы хранилищ данных ?

Вариант 1.
Какие средства Microsoft SQL Server 2008 используются для построения многомерных кубов?

☐ SQL Server 2008 Integration Services
☐ Business Intelligence Development Studio
⊞ SQL Server 2008 Analysis Services
☐ SQL Server 2008 Reporting Services

Вариант 2.
Какие средства Microsoft SQL Server 2008 используются для извлечения данных из баз данных и их преобразования перед загрузкой в хранилище?

⊞ SQL Server 2008 Integration Services
☐ Business Intelligence Development Studio
☐ SQL Server 2008 Analysis Services
☐ SQL Server 2008 Reporting Services

Вариант 3.
Какие средства Microsoft SQL Server 2008 используются для формирования пакетов обработки данных хранилища?

☐ SQL Server 2008 Integration Services
⊞ Business Intelligence Development Studio
☐ SQL Server 2008 Analysis Services
☐ SQL Server 2008 Reporting Services

Литература

1. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. 2-е изд.: Пер. с англ. – М.: Бином, СПб.: Невский диалект, 2000
2. Горев А., Макашарипов С., Владимиров Ю. Microsoft SQL Server 6.5 для профессионалов. – СПб.: Питер, 1998. – 464 с.: ил.
3. Дейт К.Дж. Введение в системы баз данных: Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с.
4. Конноли Т., Бэгг К., Страчан А. Базы данных: проектирование, реализация и сопровождение. Теория и практика. 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
5. Саймон А.Р. Стратегические технологии баз данных: менеджмент на 2000 год: Пер. с англ. / Под ред. и с предисл. М.Р. Когаловского. – М.: Финансы и статистика, 1999. – 479 с.
6. Крёнке Д. Теория и практика построения баз данных. 8-е изд. – СПб.: Питер, 2003. – 800 с.

Оглавление