Министерство образования и науки Российской Федерации

Нижегородский государственный университет
им.Н.И.Лобачевского

В.И. Швецов, А.Н. Визгунов, И.Б. Мееров

БАЗЫ ДАННЫХ

Учебное пособие

Издательство Нижегородского госуниверситета
Нижний Новгород
2004

УДК 681.3
ББК 32.97
Ш 93

Ш 93 Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Учебное пособие. Нижний Новгород: Изд-во ННГУ, 2004. 217 с..

Учебное пособие посвящено важнейшей составляющей широко разрабатываемых и используемых информационных систем организационного уровня – базам данных (БД), создаваемых и функционирующих на основе систем управления базами данных (СУБД).

Основной целью данного 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.

Учебник может служить основой общего университетского курса по базам данных. В состав пособия входит также описание лабораторного практикума, поддерживающего соответствующий курс (он может использоваться и независимо).

Предназначено для широкого круга читателей, преподавателей высшей школы, научных работников, аспирантов и студентов, интересующихся вопросами создания и использования баз данных.

ISBN 5-85746-681-4 ББК 32.97

ОГЛАВЛЕНИЕ

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 обуславливает 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.

Отражением потребности в специалистах такого рода является включение курса по базам данных в учебный план целого ряда специальностей подготовки. Так, в Нижегородском государственном университете им. Н.И.Лобачевского такой курс читается на ряде факультетов:

- факультете вычислительной математики и кибернетики для трех специальностей и направлений подготовки («Прикладная математика и информатика», «Прикладная информатика», «Информационные технологии»);
- экономическом факультете для специальности «Прикладная информатика»;
- механико-математическом факультете для специальности «Прикладная математика и информатика».

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

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

Настоящая книга написана на основе лекций по курсу «Базы данных», ежегодно читаемому одним из авторов на факультете

вычислительной математики и кибернетики (ВМК) Нижегородского государственного университета им. Н.И.Лобачевского (ННГУ). Дисциплина преподается в течение семестра и сопровождается лабораторными занятиями. При написании книги использовался опыт чтения аналогичного курса на механико-математическом факультете ННГУ, опыт проведения лабораторных работ по аналогичному курсу на экономическом факультете ННГУ, а также опыт чтения курса по основам языка SQL на факультете вычислительной математики и кибернетики ННГУ.

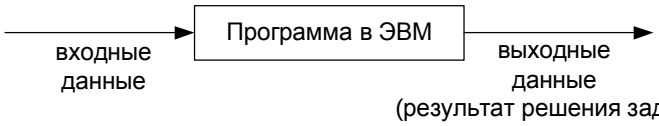
Структура данного учебника, а также его содержание в процессе подготовки пособия неоднократно обсуждались на семинарах научно-исследовательской лаборатории «Математические и программные технологии для современных компьютерных систем (информационные технологии)» факультета ВМК ННГУ, на семинаре а Санкт-Петербургском техническом университете.

Создание учебного пособия поддерживалось Фондом содействия развитию малых форм предприятий в научно-технической сфере (Фондом Бортника).

ГЛАВА 1. ВВЕДЕНИЕ В БАЗЫ ДАННЫХ. ОБЩАЯ ХАРАКТЕРИСТИКА ОСНОВНЫХ ПОНЯТИЙ ОБРАБОТКИ ДАННЫХ

1.1. Развитие основных понятий представления данных

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



Соотношение сложности представления обрабатываемых данных и алгоритма вычислений определяют 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, характеризующие представление данных при программировании, используются для представления обрабатываемых данных.

Рассмотрим простейший пример вычисления по формуле:

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

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 их значений, которые должны выбираться при программировании.

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 обуславливает 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 для решения таких задач.

Начиная с конца 60^{x} 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	Оклад	Количество отработанных дней в месяце	Начисленная сумма
FIO	O	K_o	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 невычислительных задачах вводится ряд новых понятий.

Элемент данных (поле) – наименьшая единица поименованных данных.

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

Запись – поименованная совокупность элементов данных (полей).

Экземпляр записи – текущее значение элементов записи.

Файл – поименованная совокупность всех экземпляров записей заданного типа.

Пример 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. Эти понятия нашли свое отражение в современных языках программирования, предназначенных как для вычислительных задач, так и для задач обработки 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 запись НАЧИСЛЕНИЕ представляется видом данных *RECORD*.

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

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. Данные для решения этой конкретной задачи хранится в файле MyFile.fsf, предназначенном только для решения этой задачи. Такие программные системы носят название файловых систем.

Отметим, что в этом случае описание данных включено в прикладную программу. При изменении формата записей файла необходимо изменение прикладной программы.

Задача 2. Учет кадрового состава.

Здесь обрабатываются сведения о сотруднике, представленные в карточке СОТРУДНИК:

Фамилия Имя Отчество	Должность	Год рождения	Оклад	Место жительства
FIO	D	G	O	M

Решение задачи состоит из следующих этапов:

1. Ввод исходных данных и занесение их во внешнюю память
2. Чтение исходных данных из внешней данных с целью удаления записи, корректировки записи, 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);
```

…

Разработанная программная система также является файловой системой. 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	Оклад	Количество дней на больничном листе	Невыплаченная сумма
FIO	O	$K_{дв}$	SN

$$SN = K_{дв} * O / K_r .$$

Программа решения задачи 3 аналогична программе решения задачи 1. Получается еще одна файловая система.

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

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

Устранить, по крайней мере, первые три недостатка 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*	*H*	K_o	*M*	$K_{дв}$	*S*	*SN*

Дублирование информации полностью убрано. Расход памяти минимален. Недостатки 1–3 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, длина записи, число записей i-ой 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-ой задачи (i = 1, 2, 3) при использовании файла объединенных записей, длина записи, число записей.

$$R_i \approx \mathrm{C} * \mathrm{d} * N.$$

Заметим, что $N_1 = N_2 = N$, $N_3 \ll N$

Тогда время решения i-ой задачи ($i = 1,2$) при использовании объединенного файла увеличится в $R_i/T_i \approx d/l_i$. 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 $R_3/T_3 \approx d * N / l_3 * N_3$. Т.к. для данного примера по смыслу задачи $N_3 \ll N$, то $R_3 \gg T_3$. 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 2 типа записей:

FIO	*D*	*O*	*G*	*H*	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 недостатки 1–3 и в то же время существенно увеличивает время решения всех задач. Все 3 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).

Таким образом, в этом случае, для решения вышеуказанных задач используется некоторый новый вид данных и появляется новое понятие «база данных».

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.

1.2. Системы управления базами данных

В файловых системах программы решения прикладной задачи работали с данными, предназначенными только для этой задачи. 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 приводиться 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 обуславливает необходимость отделения данных от прикладных программ.

Работа с данными должна быть организована таким образом, чтобы данные и программы не зависели друг от друга. Роль интерфейса между прикладными программами и базой данных, обеспечивающей их независимость, играет программный комплекс – система управления базами данных (СУБД) (рис. 1). СУБД – программный комплекс поддержки интегрированной совокупности данных, предназначенный для создания, ведения и использования базы данных многими пользователями (прикладными программами).

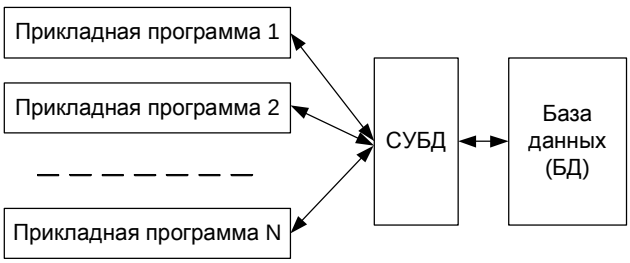


Рис.1. Обеспечение независимости прикладных программ и базы данных

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 ряд наиболее распространенных СУБД для персональных ЭВМ.

Наиболее распространенными СУБД за последние годы были 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 обуславливала 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). Система управления базами данных Access входит в пакет Microsoft Office 97 Professional. Одно время достаточно широко использовалась СУБД PARADOX (фирма Borland International). Фирма IBM представляет, в частности, СУБД DB2, которая в последние годы получает все большее распространение. Среди мощных СУБД, предназначенных для решения задач с использованием сети необходимо отметить широко распространенные системы Oracle (Oracle Corp.) MS SQL – сервер (фирма Microsoft), SYbase (фирма Sybase Inc.). Кроме вышеуказанных зарубежных систем отметим и

отечественную разработку – СУБД НИКА, преемницу широко распространенной в Советском Союзе СУБД ИНЕС для ЕС ЭВМ.

Перечислим основные функции системы управления базами данных.

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

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 и Sybase SQL Anywhere приведены на рисунках 2, 3.

Как правило, создание структуры базы данных происходит в режиме диалога. СУБД последовательно запрашивает у пользователя необходимые данные. Надо отметить, что для клиент-серверных СУБД данный диалог представляет собой графический интерфейс пользователя для формирования и выполнения соответствующих операторов языка SQL.

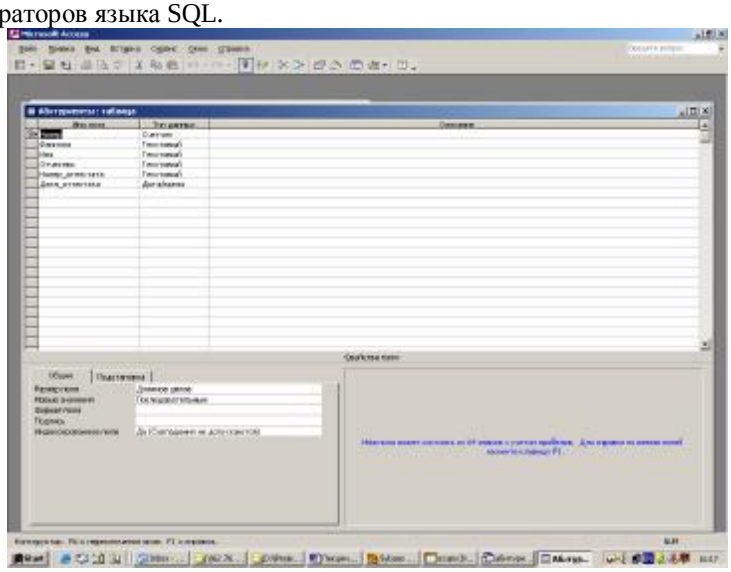


Рис. 2. Формирование структуры базы данных в СУБД Access

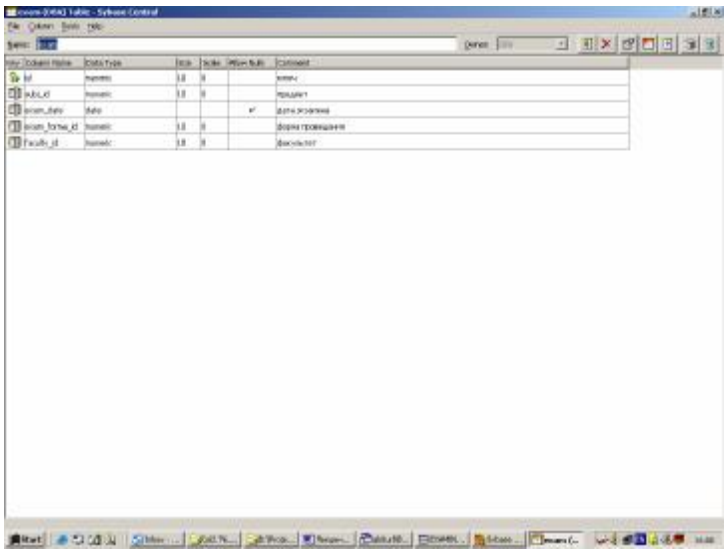


Рис.3. Формирование структуры базы данных в СУБД Sybase

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

В MS Access реализация данной функции сводится к созданию и выполнению запросов и форм ввода (рис. 4).

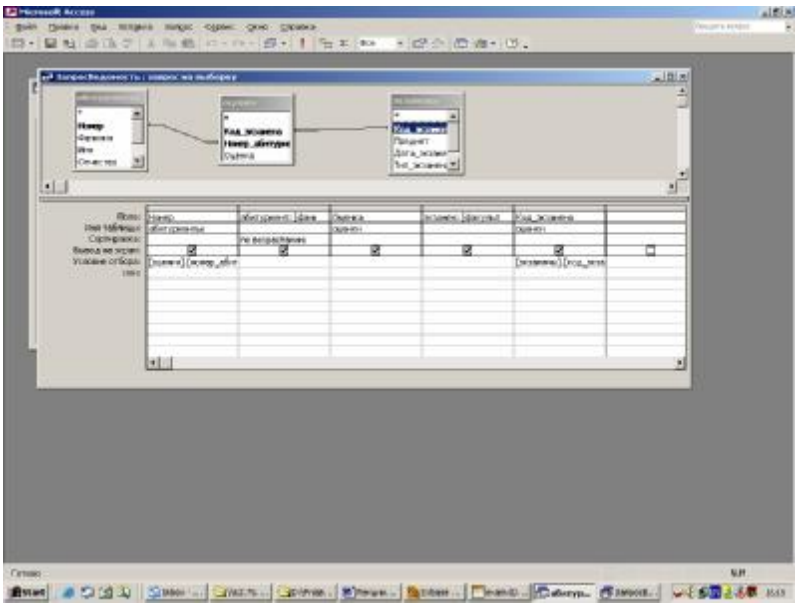


Рис.4. Формирование запроса на выборку в СУБД Access

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

Для Sybase SQL Anywhere средством для создания и выполнения запросов является программа Sybase ISQL, а средством создания GUI – Sybase Power Designer. Пример работы ISQL приведен на рис. 5. Конечно, вовсе не обязательно использовать именно эти программные продукты. В настоящее время любой современный язык программирования имеет средства для доступа к базам данных.

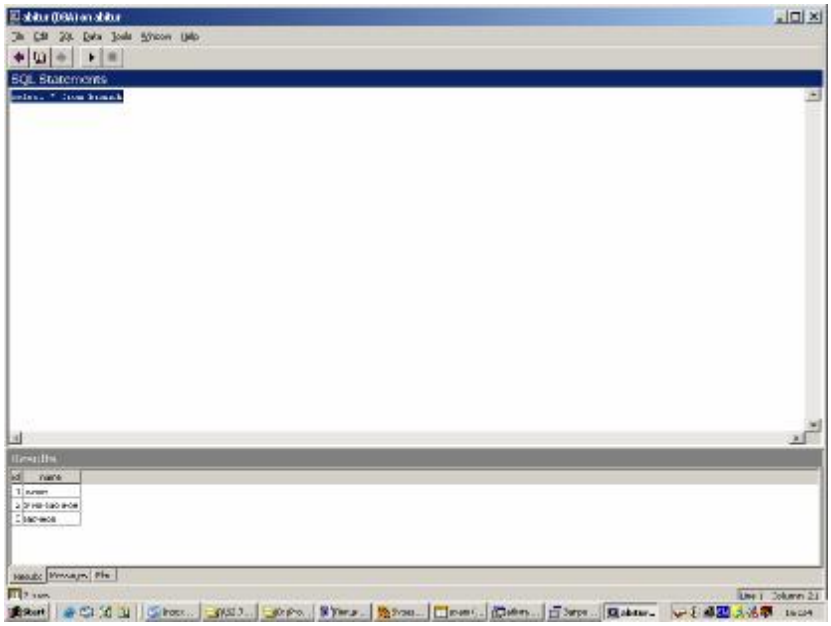


Рис.5. Пример создания и выполнения запроса с помощью ISQL

3. Обеспечение логической и физической независимости данных.

Важнейшим свойством СУБД является возможность поддерживать два независимых взгляда на базу данных – взгляд пользователя, воплощаемый в «логическом» представлении данных 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 представляет 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 вводе в БД недостоверных данных, 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.

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. 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 могли 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.

8. Управление ресурсами среды хранения.

БД располагается во внешней памяти ЭВМ. При работе в БД заносятся новые данные (занимается память), 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.

1.3. Краткий обзор литературы и других доступных источников

Началом систематизированных публикаций, в которых рассматриваются концептуальные принципы построения баз данных, по видимому, можно считать перевод технического отчета по анализу информационных систем общего назначения. Этот отчет был создан системным комитетом CODASYL на материалах сравнительного изучения десяти наиболее известных систем, разработанных в США [13]. Основная заслуга авторов состоит в попытке единого толкования понятий, реализуемых и обозначаемых в этих системах по разному, а также в изложении (впервые) с единой позиции совокупности результатов плохо совмещающихся, а иногда и противоречащих друг другу.

Классикой стала известная неоднократно переиздаваемая монография Дж. Мартина [23].

Эта монография, по сути, была первым шагом по превращению процесса разработки баз данных, основанного исключительно на основе интуиции и опыта разработчиков, в особую научную дисциплину.

Одним из первых фундаментальных учебников, где ясно и точно изложены основные принципы построения систем управления базами данных (книга входит в серию «Системное программирование», спонсируемую фирмой IBM, цель серии – накопление и широкое распространение принципов и технологий, имеющих большое значение для компьютерной индустрии) является книга К.Дж. Дейта [9].

Большой вклад в систематизации вопросов разработки баз данных и их изложении внес Дж. Ульман [28, 29], долгие годы читавший курс лекций по системам баз данных. Главное отличие его книг в том, что автор большое внимание уделяет приложениям математического аппарата для дальнейшего развития теории баз данных, а также их практической разработке.

Развитие математического аппарата реляционных баз данных целиком посвящена монография Д. Мейера [24].

Вопросы автоматизированного проектирования баз данных рассматриваются Дж. Хаббардом [30.]

В качестве недавно изданных книг, ориентированных на специалистов, занимающихся проектированием, созданием и сопровождением баз данных, и рассчитанных также на использование в качестве учебного пособия следует указать книги Г. Хансен, Дж. Хансен [31], Т. Коннолли, К. Бэгг, А.Страчан [18]. Последняя фундаментальная монография концентрирует весь опыт разработки баз данных для промышленности, бизнеса и науки, а также обучения студентов. Она является беспрецедентно полным справочным руководством по проектированию, реализации и сопровождению баз данных. Ближайшие и долговременные перспективы развития массовых технологий баз данных излагаются в монографии А. Саймона [27].

Большое количество литературы посвящено работе в среде конкретных систем управления базами данных, например [1, 4, 5, 6, 8,10, 15, 22, 25].

Детальный анализ различных аспектов технологий баз данных включающий как методологические подходы, так и программный

инструментарий приводится в книге М. Когаловского [17] (к сожалению, быстро устаревающей).

В качестве одного из первых учебников по базам данных можно указать книгу Четверикова В.П. и др. [35]. В качестве учебника по одному, но очень важному аспекту баз данных – проектированию баз данных можно указать книгу Диго С.М. [12]. К сожалению, оба вышеуказанных учебных пособия ориентированы на устаревшие программные средства и не могут быть использованы в полной мере.

Длительное время (1988–1999 гг.) учебные пособия по базам данных в центральных издательствах, к сожалению, не издавались. В последние годы эта ситуация стала меняться, можно отметить ряд учебников [16, 33], которые можно рекомендовать студентам для подготовки по отдельным аспектам баз данных.

Весьма доступным источником образовательной информации по базам данных является сеть Интернет. Так, например, на сайте FORUM www.citforum.ru выставлен учебный курс «Основы проектирования реляционных баз данных», подготовленный В.В. Кирилловым (Санкт-Петербургский Государственный институт точной механики и оптики) [39] и курс С.Д. Кузнецова «Основы современных баз данных» (информационно-аналитические материалы Центра Информационных Технологий) [40].

Можно указать учебное пособие Зеленкова Ю.А. (Ярославский государственный университет) [38]. Перспективные направления баз данных изложены в информационно-аналитических материалах С.Д. Кузнецова [41, 42].

На сайте www.eManual.ru [48] представлен обзор «Серверы корпоративных баз данных» (В.З. Шнитман, С.Д. Кузнецов), посвященный аппаратно-программным платформам СУБД и конфигурации серверов баз данных. На том же сайте содержится большое количество разнообразной документации к различным конкретным СУБД, в том числе и к рассматриваемым в данном курсе. Интересные дополнительные материалы можно найти также на сайте «Открытые системы» [49] (http://www.osp.ru). Особый интерес представляет журнал «СУБД», который, к сожалению, перестал выходить как отдельное издание.

Основным недостатком Интернет-материалов является их недолговечность и неактуальность. Зачастую прекрасные учебные материалы становятся недоступными из-за ликвидации соответствующего сайта. Отмеченных недостатков лишены материалы

фирм-производителей программного обеспечения, в частности, www.oracle.com [44]. Однако, учебные курсы на этом сайте находятся в платном доступе, что лишает студентов и преподавателей возможности пользоваться ими. В списке литературы приводятся адреса сайтов ведущих компаний-производителей соответствующего программного обеспечения [44–47].

Приведенный выше обзор не является исчерпывающим. В Интернет появляются новые курсы и пропадают уже завоевавшие популярность. Несмотря на очевидную важность и полезность информации, размещенной в Сети, она не может быть использована как основа учебного курса, а может лишь дополнять его.

По проблемам развития баз данных регулярно проводятся международные конференции, издаются ряд научных журналов. Информацию о конференциях и издаваемых научных журналах можно найти в сети Интернет. Некоторые из полезных ссылок приведены ниже.

http://www.cs.man.ac.uk/~franconi/kr.html
http://dbis-conference.informatik.tu-cottbus.de/adbis2003/topic/index.html
http://www.cs.washington.edu/homes/suciu/DBPL/
http://dke.cti.gr/SSTD03/CFP.htmhttp://www.cs.man.ac.uk/~franconi/kr.html

1.4. 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 связей между ними. Далее эти требования должны обобщаться в единое представление, которое и будет служить основой для построения единой базы данных (рис. 6).

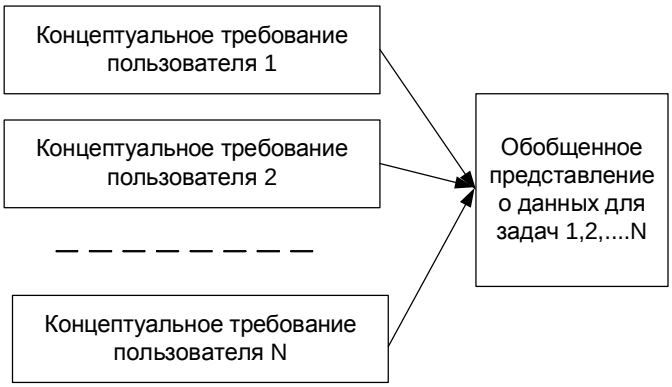


Рис.6. Обобщение представления пользователей о данных

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.

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.

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 этой прикладной программе.

Соответствующее «видение» данных прикладными программами (пользователями) представляет собой внешние представления. Взаимосвязь вышеуказанных моделей изображена на рис.7.

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 – независимость программ и данных. При изменении прикладных программ может измениться соответствующее внешнее представление, 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.

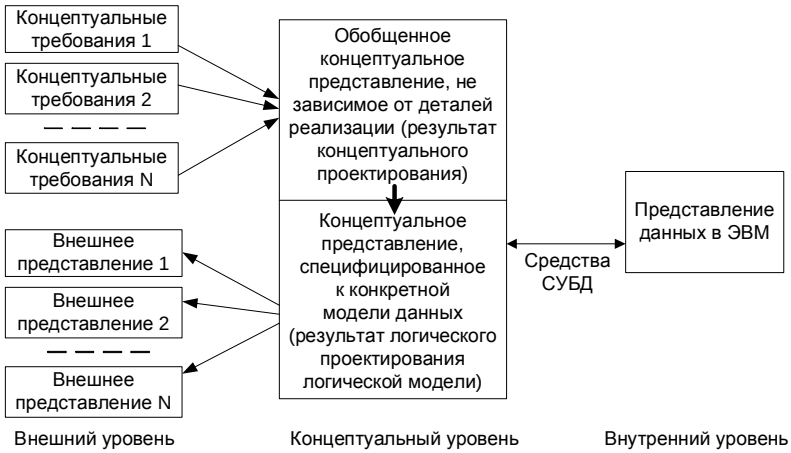


Рис.7. Различные представления о данных в БД

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.

1.5. Различные модели организации работы пользователей с базой данных

Постоянное изменение основных свойств вычислительной техники и развитие программного обеспечения обуславливало возникновение различных моделей взаимодействия пользователей с базой данных. Дадим краткую характеристику этим моделям в хронологическом порядке.

1.5.1. Модель с централизованной архитектурой

При использовании этой технологии база данных, СУБД и прикладная программа (приложение) располагаются на одном компьютере (мэйнфрейме или персональном компьютере). Для такого способа организации не требуется поддержка 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.

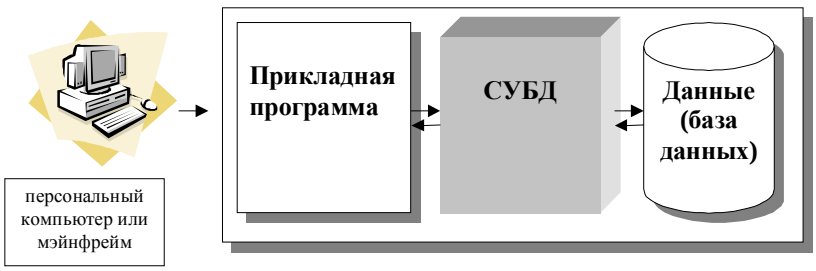


Рис. 8. Централизованная архитектура

Таким образом, в этой модели реализуется однопользовательский режим работы.

Подобная архитектура использовалась в первых версиях СУБД DB2, Oracle, Ingres [8]. Исходная идея создания и использования баз данных предполагала многопользовательское использование данных.

В данной модели с централизованной архитектурой реализуем и многопользовательский режим работы. С этой целью к мейнфрейму подключалось несколько терминалов, но тогда приходилось обслуживать в рамках ресурсов одного компьютера весь комплекс возникающих задач, начиная от собственно обработки и хранения данных, до отображения информации и приема запросов от пользователей. Модель использовалась в период «господства» больших ЭВМ (IBM-370, ЕС-1045, ЕС-1060). Все программы разных пользователей выполнялись одной ЭВМ в режиме разделения времени или мультипрограммирования. С ростом сложности задач росло количество пользователей и объемы баз данных, вследствие чего подобная архитектура более не обеспечивала удовлетворительной производительности.

Основным недостатком является резкое снижение производительности при увеличении числа пользователей.

1.5.2. Модель с автономным персональными ЭВМ

Каждый пользователь имеет свою автономную персональную ЭВМ. База данных и СУБД, копируется на компьютере каждого пользователя.

Каждый пользователь работает с базой данных на своей ЭВМ. Модель широко использовалась в начальный период появления персональных ЭВМ. Для модели характерна полная децентрализация данных. Основным недостатком модели является невозможность оперативного обновления данных на всех компьютерах при изменении из одним из пользователей.

1.5.3. Модель вычислений с сетью и файловым сервером (архитектура «файл-сервер»)

Увеличение сложности задач, появление персональных компьютеров и локальных вычислительных сетей явилось 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 [36]. В соответствие 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 [36].

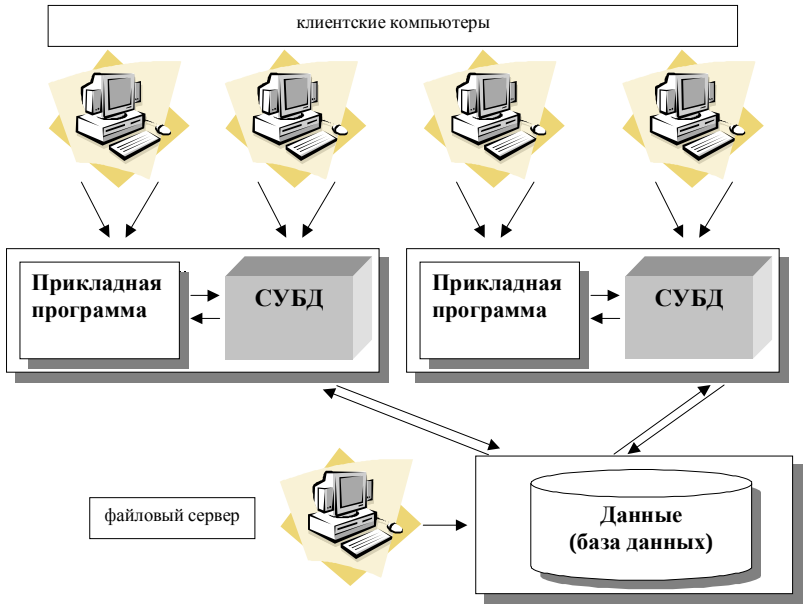


Рис. 9. Архитектура «файл-сервер»

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 т.н. настольных СУБД, таких как dBase и Microsoft Access.

В литературе 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 [36].
- 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. Т.о. не оптимально расходуются ресурсы клиентского компьютера и сети. В результате возрастает сетевой трафик и увеличиваются требования к аппаратным мощностям пользовательского компьютера [36].
- Как правило, используется навигационный подход, ориентированный на работу с отдельными записями [36].
- 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 сточки зрения внесения ошибочных изменений [36].

§ 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 [36].

1.5.4. Распределенная модель вычислений (архитектура «клиент-сервер»)

Использование архитектуры «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 [22, 36].

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 [22, 36].

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 [22, 36].

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» используются так называемые «удаленные» (или «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. Локальные СУБД предназначены для однопользовательской работы или для обеспечения работы информационных систем, рассчитанных на небольшие группы пользователей [36].

К разряду промышленных СУБД принадлежат Oracle, Gupta, Informix, Sybase, MS SQL Server, DB2, InterBase и ряд других [36].

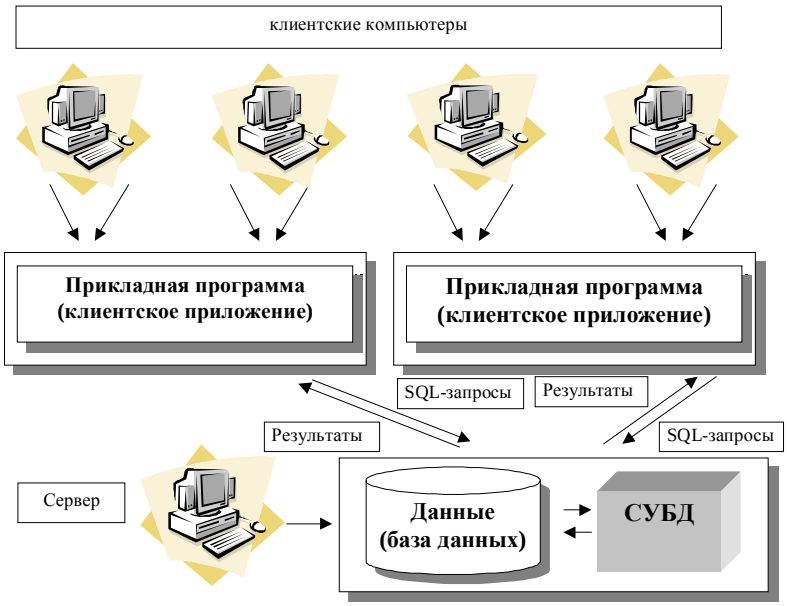


Рис. 10. Архитектура «клиент-сервер»

Как правило, SQL-сервер управляется 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 [36].

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

- Существенно уменьшает сетевой трафик.
- Уменьшает сложность клиентских приложений (большая часть нагрузки ложится на серверную часть), а, следовательно, и 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 повышает целостность и безопасность БД.

Однако нельзя не указать и некоторые недостатки и трудности, с которыми приходится сталкиваться при применении данной архитектуры. Во-первых, это стоимость SQL-сервера. Внедрение архитектуры «клиент-сервер» требует существенных финансовых ресурсов. К числу недостатков можно отнести и то, что большое количество клиентских компьютеров, расположенных в разных местах, вызывает определенные трудности со своевременным обновлением клиентских приложений на всех компьютерах-клиентах. Однако, эти недостатки не являются препятствием к использованию данных принципов построения корпоративных информационных систем. Так, архитектура «клиент-сервер» хорошо зарекомендовала себя на практике, в настоящий момент существует и функционирует большое количество БД, построенных в соответствии с данной архитектурой.

1.5.5. Распределенная модель вычислений (Клиент-сервер. Трехзвенная (многозвенная) архитектура)

Конец 90-х годов был знаменован развитием сети Интернет. Сегодня мы смело можем сказать, что Интернет, как ранее вычислительная техника, проник или активно проникает во многие сферы человеческой жизни – экономику, науку, технику, образование.

Развитие индустрии разработки программного обеспечения с одной стороны и сети Интернет, как средства коммуникации, с другой стороны привело к появлению нового взгляда на проблему построения хранилищ информации и на принципы работы с ними. Так, в современных условиях является совершенно нормальным то, что обычный человек, сидя дома, имеет возможность войти в глобальную сеть, посетить сайт компании, торгующей интересующими его товарами (например, книгами), и посмотреть, есть ли в наличии интересующая его книга, сколько она сейчас стоит, как обстоит дело с доставкой и, если все его устраивает, оформить заказ на приобретение книги.

Новое понимание действительности привело к возникновению новых технологий, новых подходов к хранению и обработке данных. А в чем, собственно, проблема, чем архитектура «клиент-сервер» плоха для организации такой деятельности, как описано выше в примере?

Один из основных моментов – снижение затрат на организацию рабочих мест пользователей. Так, для полноценной работы в Интернет не требуется каких-то запредельных мощностей. Требуется лишь возможность установки операционной системы и Web-браузера. Заметьте, что когда Вы запрашиваете информацию о книге (выполняете запросы к базе данных Интернет-магазина), на Ваш компьютер не ложиться практически никакой нагрузки. Как достигается этот эффект?

Второй момент – устранение бесконечных трудностей с обновлением клиентских приложений. Заметьте, когда Интернет-магазин меняет свое программное обеспечение, это никак не сказывается на Вас, как обычном пользователе. Вы не скачиваете никаких дополнительных файлов…

Для разрешения описанных выше проблем появилось некоторое развитие архитектуры «клиент-сервер». На настоящий момент это развитие представляет собой так называемую трехзвенную (в некоторых случаях многозвенную) архитектуру (N-tier или multi-tier). 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, выделяются 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 делового анализа [8] (бизнес-логика).
- Существует множество клиентских компьютеров, на каждом из которых установлен т.н. «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.

1.6. Краткий обзор СУБД

Первые специализированные программы для хранения обработки данных стали появляться в конце 60-х, начале 70-х годов двадцатого

века. СУБД помогала пользователям компьютеров организовывать и структурировать данные. Несмотря на то, что первые СУБД функционировали на больших ЭВМ (мэйнфреймах), с появлением персональных компьютеров активное использование БД и СУБД переместилось на них и стало доступным не только для индустриальных гигантов и исследовательских центров, но и для обычных пользователей (малый и средний бизнес, образовательные учреждения, предприятия бытовой сферы, муниципальные организации).

СУБД сыграли важную роль в развитии компьютерных сетей и Интернета [8]. Так, вследствие развития технологий представления и обмена данными в сочетании с появлением современных сетевых протоколов произошел отход от представления вычислительной системы как отдельно стоящего персонального компьютера сначала в сторону многопользовательских систем, выполненных в рамках технологий компьютерных сетей, территориально ограниченных рамками одного здания, а далее в направлении распределенных баз данных, территориально расположенных на различных компьютерах, иногда в разных частях света. В настоящее время, благодаря сети Интернет пользователю достаточно иметь лишь Web-браузер для успешной работы с базой данных, находящейся далеко за пределами здания, в котором расположен центральный офис его организации.

В литературе неоднократно можно встретить упоминания того, что сегодня рынок СУБД – это большой бизнес. Независимые компании по производству программного обеспечения и крупные поставщики продают программы для управления базами данных на миллиарды долларов ежегодно. В большинстве корпоративных приложений, обеспечивающих ежедневную деятельность крупных компаний и организаций, используются базы данных. Эти приложения подразделяются на несколько быстро развивающихся категорий [8]:

- Планирование ресурсов предприятий (Enterprise Resource Planning – ERP).
- Регулирование отношений с клиентами (Customer Relationship Management – CRM).
- Управление системой поставок (Supply Chain Management – SCM).
- Автоматизация продаж (Sales Force Automation – SFA).
- Финансовые приложения.

Производители компьютерного оборудования разрабатывают и поставляют серверы, которые специально сконфигурированы для

функционирования в качестве серверов баз данных. Коммерческое использование этих систем дает ежегодный оборот в миллиарды долларов. Базы данных обеспечивают информацией большинство Web-узлов, ориентированных на транзакции, и используются для отслеживания и анализа взаимодействия с Web-узлами. Таким образом, проблема управления базами данных затрагивает все сегменты рынка компьютерных технологий [8].

Многие авторы классифицируют СУБД на 2 большие категории: т.н. «настольные» и «серверные».

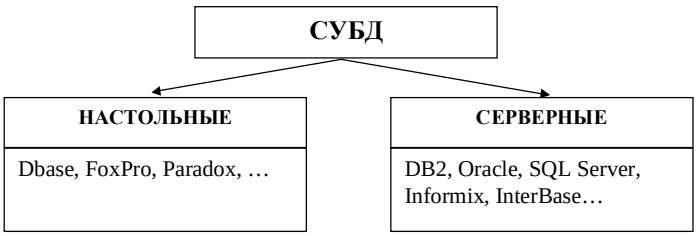


Рис. 11. Настольные и серверные СУБД

1.6.1. Настольные СУБД

Несмотря на то, что многие авторы высказывают мнение, что время этих СУБД прошло, они по-прежнему используются и некоторые из них достаточно активно. К числу подобных СУБД относятся DBase, FoxPro, Paradox, MS Access.

Конечно, настольные СУБД обладали, обладают и будут обладать всеми недостатками файл-серверной архитектуры. Не вызывают сомнения слова о плохой защищенности данных, медленной работе, трудностях с поддержкой ограничений целостности, проблемах с дублированием данных при миграции и резервном копировании, трудностях администрирования, катастрофического снижения скорости обработки при возрастании объемов данных и т.д. и т.п.

Однако, на наш взгляд используемые для решения проблемы средства должны соответствовать сложности решаемой проблемы. Так, вряд ли имеет смысл тратить на разработку и внедрение информационной системы средства, существенно большие, чем весь годовой оборот предприятия, а для многих предприятий сферы малого (а, возможно, и среднего) бизнеса дело обстоит именно так. Следует понимать, что расходы на приобретение готового программного

обеспечения (в частности, серверной СУБД), а также разработку соответствующей информационной системы, функционирующей под управлением это СУБД, составят от нескольких десятков тысяч до нескольких миллионов долларов.

Итак, где же и как используются на сегодняшний день перечисленные выше СУБД? 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 большой, 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 с успехом применяется.

Более того, последние версии настольных СУБД приобрели некоторые качества, необходимые для нормальной работы, такие, например, как поддержка ограничений целостности и механизма транзакций (подробно понятие транзакции и механизм транзакций будет описан далее).

Некоторые настольные СУБД функционируют в среде Microsoft Windows, а также «обзавелись» средствами реализации оконного пользовательского интерфейса, например, Microsoft Access (рис. 12) и Visual FoxPro.

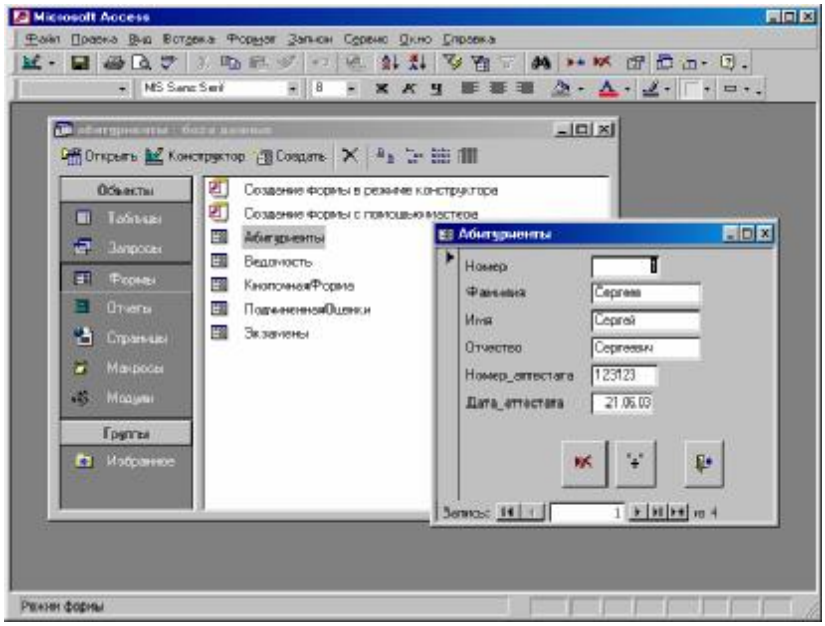


Рис. 12. Пользовательский интерфейс СУБД Access

Для тех СУБД, новые версии которых более не выпускаются, жизнь все равно не кончилась. Так, до сих пор используются базы данных, таблицы которых выполнены в формате DBase или Paradox. Совершенно понятно, что изначально эти СУБД были рассчитаны на работу в среде MSDos, однако современные средства доступа к данным позволяют с успехом использовать их под Windows, не особенно различая, какой конкретно из форматов представления таблиц используется. В частности, разработанный фирмой Borland механизм BDE (Borland Database Engine) предоставляет средства для работы с таблицами DBase, FoxPro, Paradox, Access, которые инкапсулируют внутри себя всю информацию о структуре таблиц, делая прикладного программиста «прозрачной» работу с данными. Так, текст программы, обрабатывающей базу данных, выполненную в формате одной из этих систем, вряд ли будет зависеть от того, с какой из систем действительно ведется работа.

1.6.2. Серверные СУБД

Однако, 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, разработкой которых активно занимаются компании IBM, INFORMIX, INPRISE, MICROSOFT, ORACLE, SYBASE (в алфавитном порядке). Так, в настоящий момент на рынке серверных СУБД видное место занимают DB2, Informix, MS SQL Server, Oracle, Sybase, Interbase. Вот некоторые выдержки из рекламных проспектов, взятые с сайтов www.interface.ru и www.ibm.com.ru:

«**Microsoft SQL Server 2000** – 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 сервер SQL Server 2000 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. Кроме того, SQL Server 2000 полностью использует все возможности операционной системы Windows, включая поддержку до 32 процессоров и 64 ГБ ОЗУ.

Пример пользовательского интерфейса СУБД, SQL Server приводится на рис. 13.

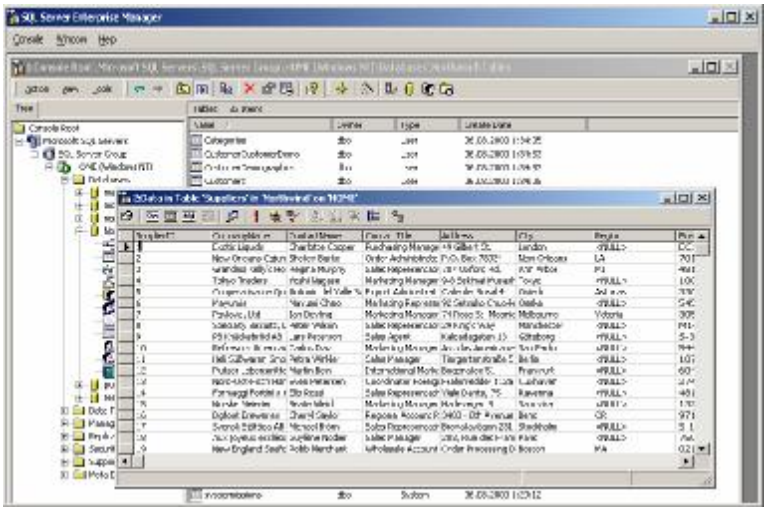


Рис. 13. Пользовательский интерфейс СУБД Microsoft SQL Server

Достоинства MS SQL Server:

- Широкая поддержка языка XML и стандартов Интернета.
- Удобный доступ к данным через веб.
- Эффективные средства анализа данных на базе веб.
- Платформа для безопасного размещения приложений.
- Масштабируемость для электронной коммерции.
- Масштабируемость для бизнес-приложений.
- Масштабируемость для хранилищ данных.
- Значительно увеличенные продолжительность бесперебойной работы и надежность.
- Интегрированные и расширяемые службы анализа.
- Упрощенное управление и настройка.
- Быстрые преобразование данных, разработка и отладка» [47]

«**Oracle9i Database** нацелена на недавно сложившийся рынок Интернет-приложений и отвечает самым строгим требованиям к качеству обслуживания. Она обладает возможностями кластеризации, мощными и экономичными средствами безопасности, исключает потери данных и позволяет интерактивно обмениваться информацией.

Oracle9i Database удовлетворит все потребности вашего электронного бизнеса в Интернете.

Основные качества:

- Масштабируемость (модуль Oracle Real Application Clusters обеспечивает масштабируемость, не зависит от используемых компонентов и позволяет масштабировать вашу систему своими силами).
- Высокая доступность (Oracle9i позволяет организовать непрерывный доступ к данным, практически исключая запланированные и аварийные задержки).
- Управление системами (встроенные в Oracle9i средства управления системами позволяют контролировать все жизненно важные компоненты, занятые в процессах электронного бизнеса).
- Безопасность в Oracle9i (Oracle9i – одна из самых безопасных платформ Интернета для защиты информационного актива вашей компании).

Новые возможности:

- Обмен деловой информацией и хранилища данных
- Oracle9i Dynamic Services
- Java и XML в Oracle9i
- Любой масштаб СУБД
- Любые компьютерные платформы и архитектуры
- Любые типы приложений
- Любые типы данных
- Переносимость приложений на платформе Oracle» [47].

«**IBM DB2 Universal Database**:

- Поддерживает новейшие стандарты Java™, а также другие технологии Web.
- Предоставляет интегрированные инструменты для интеллектуального управления бизнесом.
- Расширяет объектно-реляционные возможности.
- Сохраняет лидерство в области систем управления данными с высокой доступностью и большой производительностью.
- Обеспечивает централизованное управление гетерогенными системами.
- Позволяет выполнять более глубокую обработку данных.
- Поддерживает широкий диапазон операционных систем, в том числе Linux®» [46]

«SQL-сервер баз данных **Borland InterBase 6** объединяет простоту использования, низкие затраты на сопровождение и мощность систем корпоративного уровня. Borland гарантирует, что InterBase 6 совмещает силу мощной, апробированной архитектуры с развитыми технологиями, необходимыми для успеха прикладных систем.

Основные качества:

- Повышенная производительность за счет развитой архитектуры.
- Многопотоковая архитектура.
- Поддержка Java.
- Высокая надежность всех ваших приложений.
- Мощная поддержка различных типов данных.
- Сигнализаторы событий.
- Самонастройка и простота инсталляции.
- Реальная идентичность функциональных возможностей.
- Независимость от клиента и инструментария.
- Эффективность использования ресурсов.
- Строгое соблюдение индустриальных стандартов.
- Поддержка интернациональных требований бизнеса.
- InterBase: Embed.Deploy.Relax.
- Репликация в InterBase» [47].

Мы привели краткий обзор незначительной части существующих СУБД. При огромном разнообразии СУБД вполне естественны споры (которые возникали 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 области применимости, у каждой из них существуют свои достоинства и недостатки. Каждый из читателей может иметь личные пристрастия, которые уменьшают недостатки и увеличивают достоинства в его глазах некоторой конкретной СУБД. Мы считаем, что это нормально. Общих рекомендаций о том, какой из СУБД воспользоваться в каком-то конкретном случае, придумать сложно. 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.

1.7. Основные этапы проектирования базы данных

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

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

Процесс проектирования (моделирования) базы данных представляет собой многоэтапный процесс.

Рассмотрим основные этапы этого процесса (рис. 14).

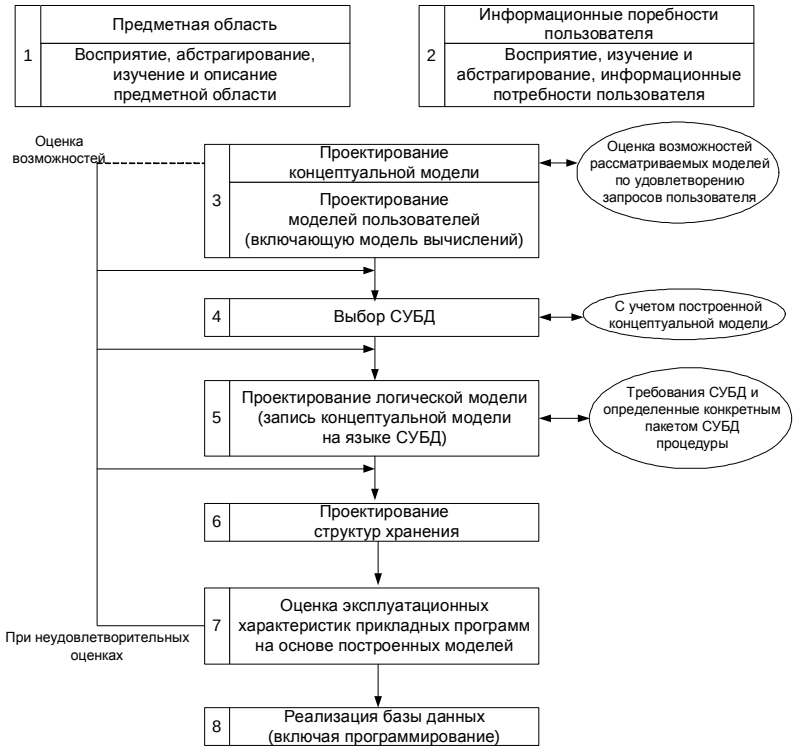


Рис.14. Этапы проектирования базы данных

Заметим, что в представленном процессе проектирования достаточно часто возникает необходимость возврата на один или несколько шагов назад. Так, например, при проектировании логической модели (блок 5) не удается достичь адекватного представления концептуальной модели средствами модели данных СУБД. В этом же 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).

1.8. Проблема целостности базы данных. Транзакции и блокировки

В функции современной СУБД кроме ведения собственно базы данных входит также ведение журнала транзакций. Транзакция – это единица действий, производимых с базой данных. В состав транзакции может входить несколько операторов изменения базы данных, но эти операторы либо выполняются все, либо не выполняется ни один.

Необходимость использования транзакций в базах данных проиллюстрируем на упрощенном примере.

Предположим, что база данных используется в некотором банке и один из клиентов банка 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 совершить либо все действия (увеличить счет одного клиента и уменьшить счет другого клиента) 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

случай стало с базой данных? Команда на уменьшение денег на счете первого клиента была выполнена, а вторая команда – на увеличение денег на другом счете – нет. Без использования транзакций описанная выше ситуация привела 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. Поскольку после выполнения первого действия транзакция не была завершена, изменения не будут внесены в базу данных. Изменения вносятся (фиксируются) только после завершения транзакции. Оператор завершения транзакций обычно называется COMMIT. 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.

С понятием транзакции тесно связано понятие блокировки. Блокировки необходимы для того, чтобы дать различным

пользователям возможность одновременно работать с базой данных. При одновременной работе в некоторый момент времени разные пользователи, могут обратиться к одной и той же записи. Или один пользователь может использовать данные, которые в данный момент меняет другой пользователь. СУБД должна запрещать подобные действия, поскольку они могут 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, снимаются блокировки.

ГЛАВА 2. КОНЦЕПТУАЛЬНОЕ МОДЕЛИРОВАНИЕ БАЗЫ ДАННЫХ

2.1. Сложный пример предметной области

Чтобы исследовать различные аспекты использования СУБД, мы рассмотрим более сложный пример, приближенный к действительности.

В качестве такого примера создания базы данных рассмотрим задачу зачисления абитуриентов в вузы. Каждый вуз решает данную задачу по-своему, чаще всего без использования баз данных. Наш пример не является реальным примером зачисления абитуриентов в ННГУ или другой вуз, однако очень близок к тем задачам, которые решает в действительности приемная комиссия ННГУ и других вузов.

До начала приема документов у абитуриентов формируются основные документы вуза, регламентирующие прием. Обычно, это Правила приема (они составляются и утверждаются ежегодно), расписание экзаменов, списки признаков, наличие которых у абитуриента, учитывается приемной комиссией при зачислении (дает абитуриенту дополнительное преимущество) и многие другие документы. Причиной ежегодного изменения Правил приема и других регламентирующих документов является добавление новых специальностей и специализаций, изменение законов Российской Федерации в части предоставления льгот некоторым категориям граждан при поступлении в вузы, желание вуза улучшить процедуру зачисления, сделать ее более справедливой.

Информационная система, которая описана ниже, позволяет решить важные для приемной комиссии вопросы – упорядочить доступ к информации, сделать информацию доступной в короткие сроки, разгрузить работников приемной комиссии, и, самое главное, избежать ошибок при зачислении.

Проведение зачисления в вуз осуществляется в несколько этапов.

Первый этап – прием документов у абитуриента. При приеме документов, абитуриент заявляет о своем желании участвовать в конкурсе на конкретную специальность, а вернее, на учебную программу. Он также может указать в заявлении, что в случае не прохождения на данную специальность, он желает участвовать в конкурсе на другие специальности этого же или другого факультета.

Для каждой выбранной специальности абитуриент указывает ее приоритет для себя.

Обычно вузы проводят сквозной конкурс. Это означает, что абитуриент, не набравший достаточное количество баллов на выбранную им специальность, может участвовать в конкурсе на другие специальности, указанные им при поступлении.

Вуз обычно предоставляет выбор предметов, которые может сдавать абитуриент. Например, на экономический факультет ННГУ можно сдавать либо математику, либо информатику. Абитуриент при поступлении указывает, какой именно экзамен он будет сдавать.

Кроме информации о приоритете специальностей и информации о том, какой экзамен он будет сдавать, абитуриент сообщает о себе некоторые дополнительные сведения (фамилию, имя, отчество, паспортные данные, предшествующее образование, гражданство и так далее), а также документы, подтверждающие его право на льготы и преимущества.

Второй этап – проведение вступительных испытаний. Испытания проводятся в форме экзаменов и последующего собеседования. На собеседовании работник приемной комиссии вместе с абитуриентов просматривает все документы абитуриента и формирует балл к зачислению по каждой специальности, на которую претендует абитуриент.

Третий этап – зачисление абитуриентов, показавших лучшие результаты на вступительных испытаниях и имеющих другие документы, предоставляющие им преимущества и льготы при зачислении. В связи с тем, что законодательство России выделяет множество категорий граждан, имеющих льготы и преимущества при поступлении в вуз, приемная комиссия формирует приказы, состоящие из большого количества пунктов, в каждом из которых указывается, в соответствии с каким нормативным документом зачисляется данный абитуриент.

Необходимо отметить, что основное количество абитуриентов не имеют льгот и зачисляются исходя их того, сколько баллов они набрали на вступительных испытаниях и какие дополнительные документы они имеют. Говорят, что абитуриенты, не имеющие льгот, участвуют в общем конкурсе. Именно общий конкурс является основным для вуза. Именно информация об абитуриентах, участвующих в общем конкурсе является объемной и требует автоматизированной обработки.

Можно также отметить, что в связи с тем, что задача зачисления актуальна для всех вузов, мы должны учесть возможное расширение нашей задачи, касающееся возможности подавать документы не только в один вуз, а сразу в несколько вузов. В самом деле, было бы вполне правильно, что абитуриент выбирает, прежде всего, специальность, а не вуз. Предположим, что абитуриент хочет изучать информационные системы. Однако специальности, связанные с информационными системами, есть во многих вузах города. Логично было бы дать возможность абитуриенту сформулировать свой выбор следующим образом. «Для меня лучшим вариантом было бы поступить на ВМК ННГУ, но, если я не пройду туда по конкурсу, то я бы хотел участвовать в конкурсе на радиофак НГТУ». Такой возможности пока нет у абитуриента, но она может появиться в том случае, если в качестве вступительных экзаменов будут засчитываться результаты централизованного тестирования.

2.2. Способы описания предметной области

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 об абитуриентах, сущностью может являться абитуриент, если обрабатывается информация об экзамене, то сущность – экзамен и т.п. 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 АБИТУРИЕНТ можно указать фамилию, дату рождения, место рождения, в качестве свойств сущности ЭКЗАМЕН можно указать предмет, дату проведения экзамена, экзаменаторов.

Совокупность сущностей, характеризующихся в информационной системе одним и тем же перечнем свойств, называется классом сущностей (набором объектов). Так, например совокупность всех сущностей АБИТУРИЕНТ составляет класс сущностей АБИТУРИЕНТ, совокупность всех сущностей – ЭКЗАМЕН составляет класс сущностей ЭКЗАМЕН.

Класс сущностей описывается перечнем свойств сущностей, составляющих этот класс.

Экземпляром сущности будем называть конкретную сущность (сущность с конкретными значениями соответствующих свойств). Пример класса сущностей АБИТУРИЕНТ и конкретного экземпляра сущности показан на рис. 15.

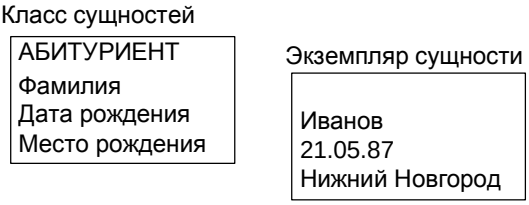


Рис.15. Класс сущностей и экземпляр сущности

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 АБИТУРИЕНТ связан с классом сущностей ЭКЗАМЕН связью «сдает». Степень этой связи равна двум. В качестве примера связи степени три можно указать связь «родители» между тремя классами сущностей МАТЬ, ОТЕЦ, РЕБЕНОК. При n=2 связь называется бинарной.

Рассмотрим классификацию бинарных связей. В зависимости от того, сколько экземпляров сущности одного типа связаны со сколькими экземплярами сущности другого типа, различают следующие типы связей:

- Связь 1:1. Одиночный экземпляр сущности одного типа связан с одиночным экземпляром сущности другого типа. Примером является связь «соответствует» между классами сущностей ФАКУЛЬТЕТ и РАСПИСАНИЕ ЭКЗАМЕНОВ НА ФАКУЛЬТЕТ (каждому факультету соответствует свое расписание).
- Связь 1:М. Единый экземпляр сущности одного типа связан со многими экземплярами сущности другого класса. Примером является связь «зачисление» между классами сущностей

ФАКУЛЬТЕТ и АБИТУРИЕНТ (на один факультет зачисляется много абитуриентов).

§ Связь M:N. Несколько экземпляров сущности одного класса связаны с несколькими экземплярами сущности другого класса. Примером является связь «поступают» между классами сущностей АБИТУРИЕНТ и ЭКЗАМЕН (каждый абитуриент сдает несколько экзаменов, и каждый экзамен сдают много абитуриентов).

Числа, описывающие типы бинарных связей (1:1, 1:M, M:N) обозначают максимальное количество сущностей на каждой стороне связи. Эти числа называются максимальными кардинальными числами, а соответствующая пара чисел называется максимальной кардинальностью.

2.3. Описание информационного представления предметной области

В качестве основного понятия для описания предметной области, как уже отмечалось, используется понятие сущности (объекта), характеризуемого набором определенных свойств. Для информационного описания сущности водится понятие атрибута.

Атрибут – поименованное свойство (характеристика) сущности. Атрибут представляет собой информационное отображение свойства сущности. Атрибут объекта принимает конкретное значение из множества допустимых значений. Так, например, для сущности АБИТУРИЕНТ атрибут «фамилия» у конкретного экземпляра сущности принимает конкретное значение «Иванов».

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.

Другим основным понятием для описания предметной области является понятие связи. Разные способы информационного описания связей будут рассмотрены позднее. В данном разделе отметим, что, в частности, для представления связей между экземплярами сущностей могут использоваться атрибуты. В этом случае связь устанавливается путем включения в совокупность атрибутов сущности атрибута, однозначно идентифицирующего экземпляр сущности, находящийся в отношении с исходным экземпляром сущности.

Так, рассмотрим класс сущностей ФАКУЛЬТЕТ представленный совокупностью атрибутов (название, номер) и класс сущностей РАСПИСАНИЕ ЭКЗАМЕНОВ НА ФАКУЛЬТЕТ, представленный совокупностью атрибутов (название экзамена 1, дата экзамена 1, название экзамена 2, дата экзамена 2, название экзамена 3, дата экзамена 3). Для представления связи «экзамены» (тип связи 1:1) в совокупность атрибутов РАСПИСАНИЕ ЭКЗАМЕНОВ НА ФАКУЛЬТЕТ можно включить атрибут «название факультета».

2.4. Описание информационных потребностей пользователя

Информационные потребности пользователя представляются в виде запросов к информации об экземплярах сущностей. Запросы бывают простые и сложные. Сложные запросы составляются из простых запросов с помощью логических операторов *.and.*, *.or.* или *.not.*, поэтому без ограничения общности будем говорить только о простых запросах. Для реализации подавляющего числа запросов пользователю, прежде всего, необходимо найти интересующий его экземпляр сущности (с целью обработки, корректировки, удаления). Для идентификации экземпляра используется один или несколько атрибутов.

Атрибут или множество атрибутов, значения которых однозначно идентифицируют экземпляр записи, называется потенциальным ключом.

Потенциальный ключ, который выбран в качестве основного идентификатора, называется первичным ключом.

Потенциальный ключ, который состоит из двух или больше атрибутов, называется составным ключом.

Часто для поиска используется атрибут или множество атрибутов, которые не идентифицируют экземпляр сущности единственным образом и характеризуют определенную группу записей. Такой атрибут (множество атрибутов) называется вторичным ключом.

Приведем шесть абстрагированных типов простых запросов, относящихся к экземпляру сущности E, атрибутам A и значениям атрибутов V [23].

Тип	Форма	Смысл	Пример
1	*A(E)=?*	Каково значение атрибута *A* экземпляра сущности *E*	Место рождения абитуриента Иванов

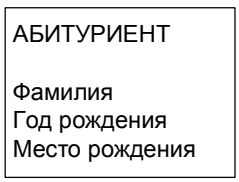
2	= ı *A(?)* < *V* >	Какой экземпляр *E* имеет значение атрибута *A* равное (неравное, меньшее, большее) *V*	Кто из абитуриентов имеет год рождения <1969 (старше 35 лет)
3	= ı *?(E)* < *V* >	Какой атрибут или атрибуты экземпляра *E* имеют значение равное (неравное, меньшее, большее) *V*	За какие экзамены абитуриент Иванов получил оценку «отлично»

4	*?(E)=?*	Запрос на получение значений всех атрибутов экземпляра *E*	Сообщить всю информацию об абитуриенте Иванове
5	*A(?)=?*	Перечислить значения данного атрибута для каждого экземпляра	Перечислить названия всех специальностей (сущность СПЕЦИАЛЬНОСТЬ)
6	= ı *?(?)* < *V* >	Перечислить все атрибуты экземпляров, имеющие значение равное (неравное, меньшее, большее) *V*	Перечислить все атрибуты абитуриентов, получивших оценку «2»

2.5. Построение ER-диаграмм

Чаще всего концептуальная модель представляется в виде диаграммы сущностей-связей (entity-relationship) или ER-диаграммы. Процесс построения ER-диаграммы называется ER-моделированием. При этом используются следующие классические обозначения.

Класс сущностей представляется в виде четырехугольника. В четырехугольнике записано уникальное имя класса сущности (заглавными буквами) и имена атрибутов строчными буквами.



Связи между сущностями обозначаются стрелками, рядом со стрелками указывается имя связи, а также максимальная

кардинальность связи (максимальное число сущностей, которые могут участвовать в связи). Чтобы показать, что сущность обязана участвовать в связи (каждый экземпляр должен быть связан с экземпляром другого класса) на линию связи помещают перпендикулярную черту, а чтобы показать, что сущность может (но не обязана) участвовать в связи, на линию связи помещают овал.

При описании сущностей выделяют особые совокупности атрибутов – ключи. Роль ключа состоит в уникальной идентификации экземпляра сущности и реализации связей.

На диаграммах атрибуты, входящие в первичный ключ, подчеркиваются (обозначение PK).

Важность понятий ключа и внешнего ключа будет проиллюстрирована далее на примерах.

Приведенные выше обозначения не являются общеупотребительными. Часто производитель программ, позволяющих рисовать ER-диаграммы, использует свою собственную систему обозначений. Например, при использовании программы MS Visio связи обозначаются стрелками между сущностями и названия связей не пишутся. Название сущности выделяется цветом, поля, входящие в первичный ключ отделяется чертой от остальных атрибутов. Кроме того, слева от атрибута указывается, входит ли данный атрибут в первичный ключ, а также является ли атрибут внешним ключом.

На практике использование различных способов записи ER-диаграмм не представляет собой особой сложности – беглое ознакомление с соответствующим разделом документации позволяет быстро освоить используемую систему обозначений.

2.6. Выявление и моделирование сущностей и связей

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

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

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 информационных потребностей пользователей).

Далее необходимо присвоить связям имена, и определить тип связей.

В качестве примера моделирования сущностей и связей рассмотрим небольшой фрагмент предметной области зачисления абитуриентов в вуз, рассмотренный в разделе 2.1.

Наш фрагмент будет относиться к представлению в базе данных информации о специальностях, факультетах и университетах.

Первоначально можно было рассмотреть сущность СПЕЦИАЛЬНОСТЬ, которая включает в себя следующие атрибуты: название специальности, номер специальности по общероссийскому классификатору; факультет, на котором преподается данная специальность; вуз, в состав которого входит факультет, отделение. Рассмотрев данную сущность, мы видим, что значительная часть информации дублируется.

Для иллюстрации дублирования информации возьмем в качестве примера специальность «прикладная математика». По этой специальности ведут обучение и, соответственно, осуществляют прием на дневное и вечернее отделение два факультета ННГУ. Каждая из этих специальностей различна с точки зрения нашей предметной области – зачисление на каждую производится отдельно. Соответственно, каждая специальность будет представлена отдельным экземпляром сущности. Однако информация по названию специальности и номеру специальности по общероссийскому классификатору будет дублироваться. Кроме того, для очной и вечерней форм обучения внутри одного факультета, будет дублироваться вся информация, кроме названия отделения. Это признак того, что внутри данной сущности находится несколько сущностей.

С учетом вышесказанного, можно выделить следующие сущности и связи.

Сущность УЧЕБНАЯ ПРОГРАММА – это сущность, которая описывает конкретную учебную программу. Для нее указаны специальность (название и номер), факультет и отделение, на которых она преподается.

Информация о специальностях – название и номер по общероссийскому классификатору вынесены в отдельную сущность СПЕЦИАЛЬНОСТЬ. Информация об отделениях будет находиться в сущности ОТДЕЛЕНИЕ.

Сущность ФАКУЛЬТЕТ определяет название и аббревиатуру факультета, а также вуз, в состав которого входит факультет.

Между сущностью УЧЕБНАЯ ПРОГРАММА и остальными сущностями выделяется связь «характеризуется». Конкретная учебная программа характеризуется факультетом, специальностью, отделением.

Этот вариант является хорошим с точки зрения нашей предметной области и возникает вопрос – необходимо ли было выделять отдельную сущность ВУЗ (см. итоговую диаграмму)? Может быть, достаточно было в сущности ФАКУЛЬТЕТ хранить атрибут «вуз»? Ответ на данный вопрос заключается в следующем.

Одно из основных свойств базы данных заключается в ее постоянной модификации в процессе эксплуатации. Вполне логично предположить, что помимо названия вуза, нам будет нужно хранить информацию о других характеристиках вуза – например, аббревиатуре и адресе вуза. Если это произойдет, (а это частично произошло – мы уже храним информацию об аббревиатуре и названии), то информация будет дублироваться. Таким образом, нужно выделить сущность ВУЗ, которая связана с сущностью ФАКУЛЬТЕТ (а не с сущностью УЧЕБНАЯ ПРОГРАММА!). Связь «входит в состав». Факультет вычислительной математики и кибернетики (ВМК) входит в состав ННГУ.

Определим типы связей.

Сущность УЧЕБНАЯ ПРОГРАММА характеризуется сущностью СПЕЦИАЛЬНОСТЬ. Каждому экземпляру сущности УЧЕБНАЯ ПРОГРАММА ставится в соответствии один экземпляр сущности СПЕЦИАЛЬНОСТЬ. Каждому экземпляру сущности СПЕЦИАЛЬНОСТЬ ставиться ноль, один или много экземпляров сущности УЧЕБНАЯ ПРОГРАММА. Связь один ко многим, причем для сущности УЧЕБНАЯ ПРОГРАММА связь полная – каждый экземпляр участвует в связи, а для сущности СПЕЦИАЛЬНОСТЬ связь не полная – может

существовать специальность в общероссийском справочнике специальностей, по которой не проводится подготовка в вузах, занесенных в базу данных.

Отметим, что для представления связей между сущностями (ссылки на другую сущность) используется идентификатор соответствующей сущности (обозначение id), в качестве которого используется специально вводимый код сущности, который однозначно идентифицирует объект.

Итоговая диаграмма «Сущность-Связь» приведена на рисунке 16.

Обозначения в диаграмме:

University – сущность ВУЗ – справочник вузов. Название вуза хранится в атрибуте «name», аббревиатура вуза – атрибут «brief».

Branch – сущность ОТДЕЛЕНИЕ, в которой хранится информация об отделении. Обычно выделяют три отделения – дневное, очно-заочное (вечернее), заочное. name – название отделения.

Faculty – сущность ФАКУЛЬТЕТ – справочник факультетов вуза. name – название факультета, brief – аббревиатура, univer_id – ссылка на вуз.

Program – сущность УЧЕБНАЯ ПРОГРАММА – учебные программы, по которым проводит обучение вуз. Данная сущность связана с другими сущностями – spec (специальности), branch (отделения), faculty (факультеты).

Spec – сущность СПЕЦИАЛЬНОСТЬ – справочник специальностей (общероссийский классификатор). Содержатся данные о названии специальности (name) и номере специальности (spec_number).

PK – первый ключ, FKN – вторичный ключ с номером N.

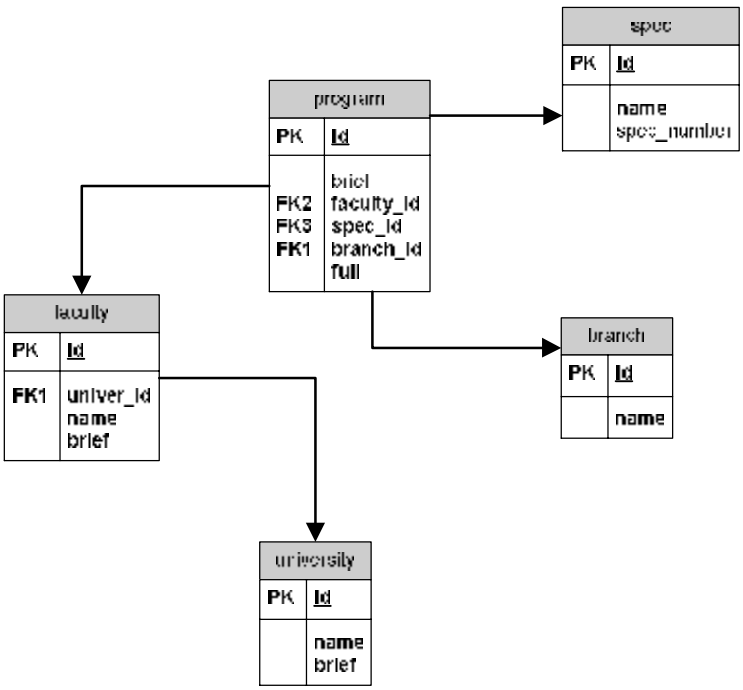


Рис. 16. Фрагмент ER-диаграммы

2.7. Построение концептуальной модели

Как уже отмечалось, концептуальная модель представляет собой обобщение представлений разных пользователей о данных. В связи с этим построение концептуальной модели, как правило, проводится 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 представления отдельных пользователей приводит к снижению уровня интеграции данных, а моделирование совместных представлений группы пользователей приводит к повышению

сложности проектирования. В связи с этим, при выборе области данных для локального моделирования приходится выбирать компромиссное решение между вышеуказанными вариантами.

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

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

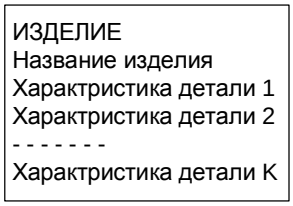
Вариативность моделирования обуславливается неоднозначностью выбора сущностей, атрибутов и связей. В одном варианте можно что-то взять за сущность, в другом варианте это что-то можно взять за атрибут (несколько атрибутов), в третьем варианте это что-то можно определить как связь. Рассмотрим вышесказанное на простом примере.

Пример. Предметная область описывается следующим образом. Есть ряд изделий. Каждое изделие состоит из совокупности деталей. Возможны следующие запросы пользователей:

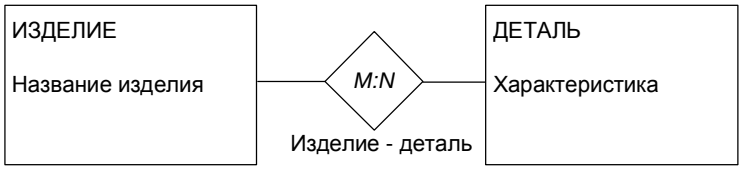
- Из каких деталей состоит конкретное изделие *B*?
- В какие изделия входит деталь *D*?

По-разному выбирая сущности, атрибуты и связи можно построить три варианта модели.

Вариант 1. Сущность – ИЗДЕЛИЕ. Атрибутами изделия являются название изделия и характеристики деталей, его составляющих.



Вариант 2. Сущность – ИЗДЕЛИЕ. Сущность – ДЕТАЛЬ. Связь – изделие состоит из определенных деталей и деталь входит в определенные изделия.



Таким образом, даже для такой простейшей предметной области можно построить две модели.

Оценим число просматриваемых сущностей для каждого варианта при ответе на два вышеуказанных запроса пользователей. Для определенности предположим, что конкретный экземпляр сущности в наборе отыскивается посредством перебора (возможны и другие предположения).

Обозначим M – число изделий; N – число деталей; k_B – число деталей, входящих в изделие B, l_D – число изделий, в которые входит деталь D. Результаты представлены в следующей таблице.

Номер запроса	Вариант 1	Вариант 1
1	$\frac{1+M}{2}$	$\frac{1+M}{2}+k_B$
2	M	$\frac{1+N}{2}+l_D$

Поясним оценки, получаемые при запросе 1. Для ответа на запрос по модели первого варианта просматриваем экземпляры набора сущностей ИЗДЕЛИЕ и ищем экземпляр с названием D. В лучшем случае, это может быть первый же экземпляр, в худшем – последний.

Среднее число просматриваемых экземпляров будет равно (1+*M*)/2. Найдя нужный экземпляр, получаем его атрибуты. Ответ на запрос получен. Для ответа на запрос по модели второго варианта нужно найти экземпляр изделия с названием *D* (аналогично варианту 1), далее по связи перейти в набор сущностей ДЕТАЛЬ и там выбрать (по связи) детали, входящие в изделие *B* (выбрать k_B экземпляров). Общая оценка для этого случая будет (1+*M*)/2+k_B. Подсчет остальных оценок аналогичен. Сравним варианты.

Обозначим частоту *i*-го запроса q_i (*i*= 1, 2).

Тогда сравнение вариантов по вышеуказанным оценкам сводится к сравнению величин

$$q_1 * (1 + M)/2 + q_2 * M$$

и

$$q_1[(1 + M)/2 + k_B] + q_2[(1 + N)/2 + l_D]$$

Меньшая из этих величин и определяет наилучший из представленных вариантов по оценке числа действий при ответах на запросы пользователей.

Нетрудно видеть, что в варианте 1 больше дублирования информации. Характеристика одной и той же детали может встречаться в нескольких изделиях. Рациональный выбор варианта основывается на компромиссе между оценками числа действий и оценками дублирования информации и зависит от конкретных соотношений этих оценок.

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).

Процесс выполнения вышеуказанных действий, вообще говоря, носит итерационный характер, т.к. после их выполнения вновь могут возникать и расплывчатые наименования, и синонимы, и омонимы.

Завершающим шагом локального моделирования является указание ключей для каждого набора сущностей. Здесь необходимо указать как первичные, так и, по возможности, вторичные ключи.

2.7.2. 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.

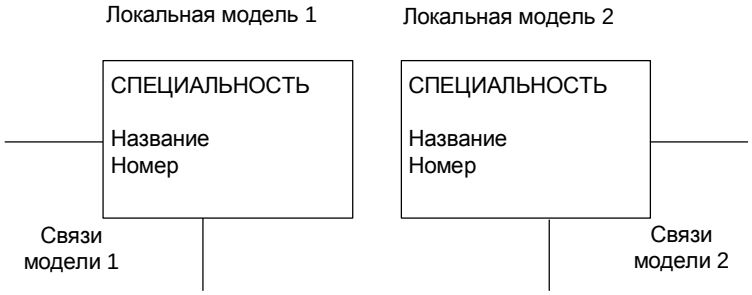


Рис.17 . Модели с идентичным элементом

Объединение моделей с идентичными элементами осуществляется путем «слияния» этих элементов в один элемент.

Два набора сущностей СПЕЦИАЛЬНОСТЬ в модели 1 и 2 имеют одинаковое смысловое значение, и могут быть заменены одним набором сущностей.

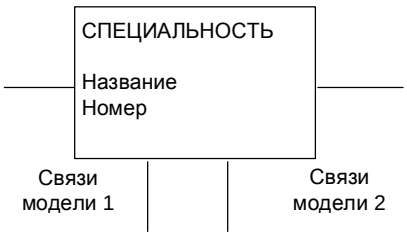


Рис.18. Объединенная модель

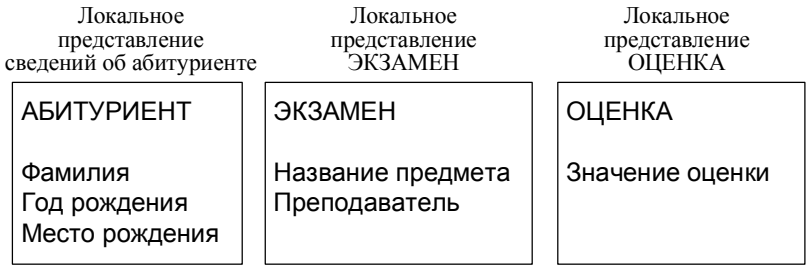
Установление связей между наборами сущностей разных моделей

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 абитуриентом вступительных экзаменов. Можно выделить ряд локальных представлений.



Оценка принимает значение «отлично», «хорошо», «удовлетворительно», «неудовлетворительно».

Объединяя локальные представления, устанавливаем новые связи:

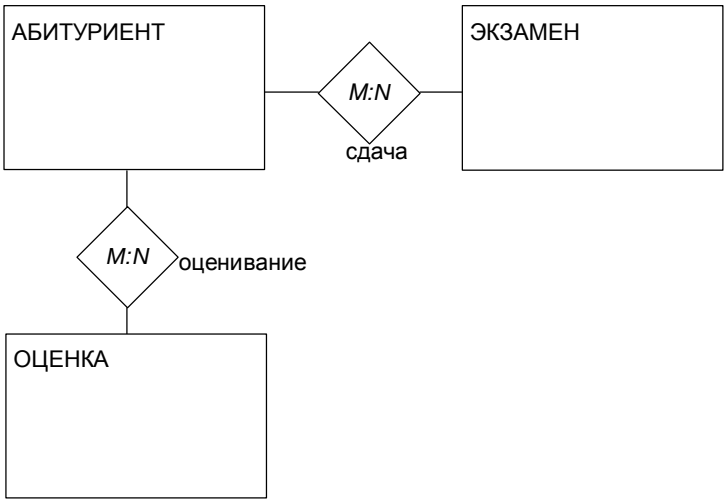
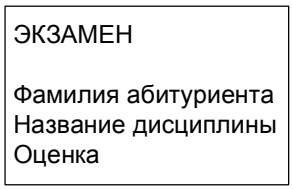


Рис.19. Объединение локальных представлений

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 не 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 следующим образом.



Далее процесс объединения локальных моделей продолжается обычным образом.

Обобщение подобных типов сущностей

Рассмотрим ситуацию, когда каждый факультет формирует свое отделение приемной комиссии. Работники факультетской приемной комиссии учитывают данные об абитуриентах, поступающих на данный факультет, используя информацию о специальностях данного факультета.

В этом случае можно выделить несколько локальных моделей – модель факультета ВМК, модель экономического факультета и так далее. В локальную модель факультета ВМК входят сущности «Специальности факультета ВМК» и «Абитуриенты факультета ВМК», в локальную модель экономического факультета входят, соответственно, сущности «Специальности экономического факультета» и «Абитуриенты экономического факультета».

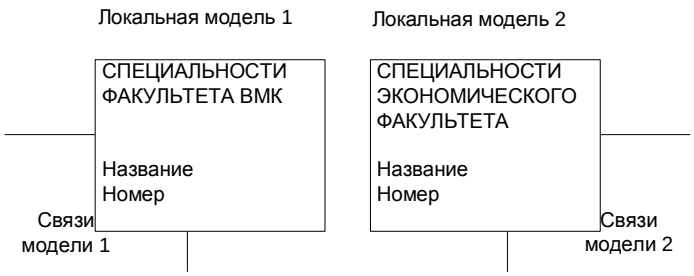


Рис. 20. Модели с идентичным элементом

Два набора сущностей СПЕЦИАЛЬНОСТИ ФАКУЛЬТЕТА ВМК и СПЕЦИАЛЬНОСТИ ЭКОНОМИЧЕСКОГО ФАКУЛЬТЕТА в модели 1 и 2 имеют одинаковое смысловое значение, и могут быть заменены одним набором сущностей с добавлением нового атрибута – факультет (рис. 21).

Отметим, что в данном случае подобным образом можно слить и все остальные сущности локальных моделей факультетов, так как сущности «Абитуриенты экономического факультета» и «Абитуриенты ВМК» также имеют одинаковое смысловое значение. Однако в общем случае каждая локальная модель может содержать сущности и связи, которых нет в других локальных моделях.

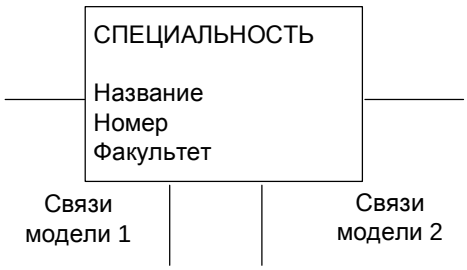
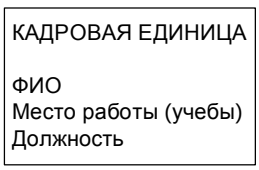


Рис. 21. Обобщенная модель

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 «ФИО» 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.



У студента атрибут «Место работы (учебы)» будет принимать значение «Курс. Группа», у преподавателя – название кафедры. Обобщенная модель представлена на рис. 22.

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.

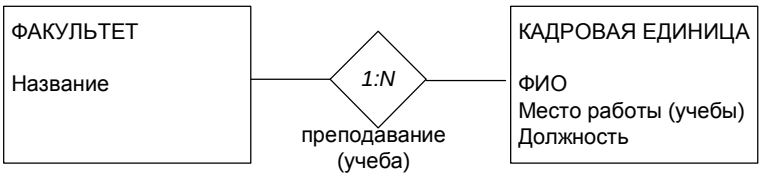


Рис.22. Обобщенная модель

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.

2.8. Пример построения диаграммы «Сущность-Связь» для предметной области зачисления абитуриентов

В предметной области можно выделить следующие группы пользователей: сотрудники приемной комиссии и администраторы данных.

Рассмотрим первую группу пользователей – сотрудников приемной комиссии, которые обрабатывают информацию об абитуриентах.

Данная группа пользователей принимает документы абитуриентов (на какую специальность поступает абитуриент, на какие специальности, в случае не прохождения основную, он хочет поступить, какие экзамены будет сдавать). Эти пользователи также заносят оценки, выставленные членами предметной комиссии в ведомости, проводят итоговое собеседование с абитуриентами,

добавляя дополнительные баллы за наличие соответствующих документов, готовят приказы к зачислению, исходя из полученных оценок и дополнительных баллов.

Диаграмма «Сущность-Связь» для данной группы пользователей приведена на рисунке 23. Разберем подробно, как она получилась.

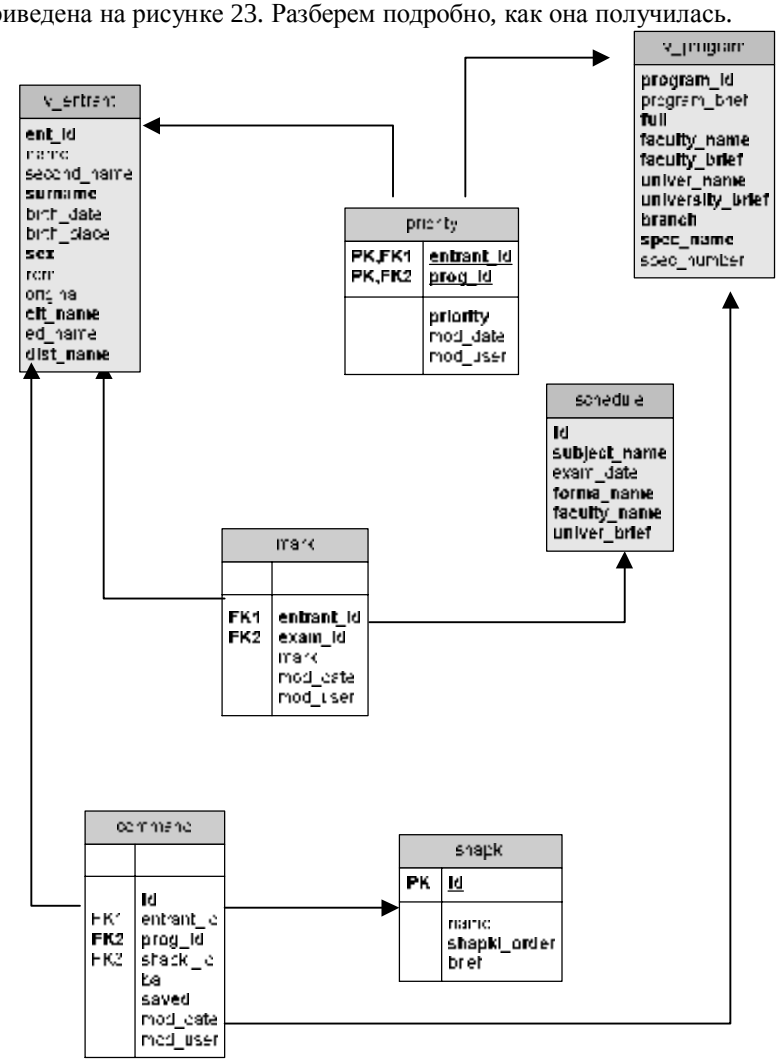


Рис.23. Представление приемной комиссии

При рассмотрении разобьем модель на две части – одна часть касается принятия документов абитуриента и проведения экзаменов, вторая – зачисление.

Информацию о том, какие экзамены будет сдавать абитуриент, составляет сущность ОЦЕНКИ (mark). До того момента, пока абитуриент не получил оценку, значение оценки в сущности «оценка» остается неопределенным.

В сущности ОЦЕНКИ (mark) выделяются следующие атрибуты.

- Оценка (mark) – значение оценки или неопределенное значение (null), если абитуриенту назначен экзамен, но он его еще не сдавал.
- Дата модификации экземпляра сущности (mod_date).
- Пользователь, который последний модифицировал экземпляр сущности (mod_user).
- Атрибуты entrant_id и exam_id служат для реализации связи данной сущности с сущностями АБИТУРИЕНТ и РАСПИСАНИЕ. Данные атрибуты являются внешними ключами, ссылающимися на соответствующие сущности.
- Приоритеты абитуриентов составляют сущность ПРИОРИТЕТЫ (priority).

В сущности ПРИОРИТЕТЫ выделяются следующие атрибуты.

- Значение приоритета (priority).
- Дата модификации экземпляра сущности (mod_date).
- Пользователь, который последний модифицировал экземпляр сущности (mod_user).
- Атрибуты entrant_id и prog_id служат для реализации связи данной сущности с сущностями АБИТУРИЕНТ и СПЕЦИАЛЬНОСТЬ. Данные атрибуты являются внешними ключами, ссылающимися на соответствующие сущности.

Сущность АБИТУРИЕНТ (v_entrant) состоит из всех атрибутов, относящихся к абитуриенту. Надо заметить, что после некоторого анализа будет понятно, что в сущности АБИТУРИЕНТ приведенной на рисунке скрыто еще несколько сущностей. То же относится и к

сущностям РАСПИСАНИЕ (schedule) и УЧЕБНАЯ ПРОГРАММА (v_program).

В сущности АБИТУРИЕНТ выделяются следующие атрибуты.

- Код абитуриента (ent_id).
- Имя абитуриента (name).
- Отчество абитуриента (second_name).
- Фамилия абитуриента (surname).
- Дата рождения (birth_date).
- Место рождения (birth_place).
- Пол (sex).
- Примечание (rem).
- Представлен подлинник документа (original). В данном атрибуте содержится информация о том, сдал ли абитуриент в приемную комиссию подлинник документа о предшествующем образовании. По современному законодательству абитуриент имеет право подавать документы в несколько вузов одновременно, соответственно, сдавая не оригинал документа, а копию. Однако он допускается к зачислению только в одном вузе – в том, в который он принес подлинник. Если абитуриент пытается сдавать экзамены в несколько вузов, то после сдачи экзаменов, но до зачисления, он выбирает один и приносит туда подлинник. Вуз устанавливает некоторую пороговую дату, до которой абитуриент может принести подлинник документа. После этой даты вуз не принимает подлинник и не допускает абитуриента к участию в конкурсе.
- Гражданство (cit_name).
- Предшествующее образование (ed_name).
- Документ о медали или дипломе с отличием (dist_name).

Сущность РАСПИСАНИЕ (schedule) содержит информацию о том, какие экзамены на какие факультеты какого вуза назначены приемной комиссией.

Атрибуты сущности РАСПИСАНИЕ.

- Предмет (subject_name).
- Дата экзамена (exam_date).
- Форма проведения экзамена (forma_name).
- Факультет (faculty).
- Аббревиатура вуза (univer_brief).

Сущность СПЕЦИАЛЬНОСТЬ (v_program) содержит информацию по учебным программам, на которые могут поступать абитуриенты. Мы уже говорили о том, что учебная программа и специальность – это, на самом деле, разные сущности. Специальность определяется названием специальности и ее номером согласно общероссийскому классификатору. Учебная программа отличается уточнением факультета, вуза, и отделения. С этой точки зрения логичнее было бы назвать сущность УЧЕБНАЯ ПРОГРАММА. Однако термин «специальность» является устоявшимся и понятен работникам приемной комиссии. В то же время термин «учебная программа» был введен искусственно – и понятен администратору данных, но не пользователю. Поскольку мы в настоящий момент строим диаграмму «сущность-связь» для пользователя, то нужно, по возможности, сохранять терминологию, привычную для предметной области. Выделение из рассмотренной сейчас сущности СПЕЦИАЛЬНОСТЬ нескольких сущностей относится к задаче, решаемой при построении обобщенной диаграммы «сущность-связь».

Атрибуты сущности СПЕЦИАЛЬНОСТЬ.

- Краткое название учебной программы или, в терминах специалиста предметной области, краткое название специальности (program_brief).
- Полное название специальности, а если более правильно, то полное название учебной программы (full).
- Название факультета (faculty_name).
- Аббревиатура факультета (faculty_brief).
- Название вуза (univer_name).
- Аббревиатура вуза (university_brief).
- Отделение (branch).
- Название специальности (spec_name).
- Номер специальности (spec_number).

Другие части диаграммы будут пояснены ниже.

Сущности АБИТУРИЕНТ, РАСПИСАНИЕ, ОЦЕНКА, ПРИОРИТЕТ, СПЕЦИАЛЬНОСТЬ связаны между собой. Абитуриент получает оценку по конкретному экзамену. Соответственно, выделяются связи «абитуриент получает оценку», «оценка ставится за экзамен». Абитуриент формирует свои приоритеты на специальности (на

учебные программы). Связи – «абитуриент определяет приоритет», «специальности соответствует приоритет».

Связи сформулированы неочевидно, поскольку являются надуманными. На самом деле надо было выделить две связи, каждая из которых связывает три сущности. Абитуриент получает оценку за экзамен (сущности АБИТУРИЕНТ, ЭКЗАМЕН, ОЦЕНКА). Абитуриент определяет приоритеты специальностей (сущности АБИТУРИЕНТ, СПЕЦИАЛЬНОСТЬ, ПРИОРИТЕТ). Причина выделения бинарных связей проста – только бинарные связи поддерживаются в реляционной модели, которая используется в большинстве современных СУБД.

Рассмотрим часть модели, касающейся зачисления.

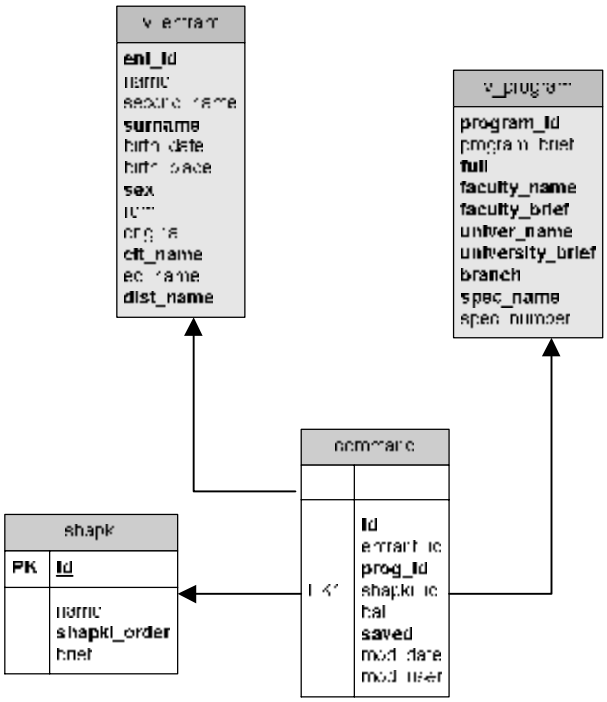


Рис.24. Представление о зачислении (фрагмент рис. 23)

Сущности АБИТУРИЕНТ и СПЕЦИАЛЬНОСТЬ были описаны ранее.

Сущность ПУНКТЫ ПРИКАЗА (shapki) содержит информацию об основаниях, согласно которым зачисляются абитуриенты.

Атрибуты сущности ПУНКТЫ ПРИКАЗА.

– Содержание пункта (name).
– Краткое название пункта (brief).
– Номер пункта по порядку (shapki_order).

Сущность COMMAND содержит информацию о том, какие абитуриенты зачислены на какие специальности и согласно какому пункту приказа.

Помимо связующих атрибутов (entrant_id, prog_id, shapki_id), хранится информация о дате последнего изменения экземпляра сущности (mod_date) и пользователе, осуществившем последнее изменение (mod_user), а также информация о набранном балле (ball) и признак того, что информация запомнена и не нуждается в пересчете (saved).

Связь между сущностями формулируется «абитуриент зачислен на специальность в соответствии с пунктом приказа». Формулировать разбиение данной связи на бинарные мы не будем.

Вторая группа пользователей – администраторы данных.

Администраторы данных имеют доступ ко всей информации, представленной в базе данных. Они могут изменять структуру данных и заполнять справочники. Для всех остальных пользователей справочники являются таблицами на чтение, то есть можно просмотреть данную информацию, но нельзя изменить. Например, пользователь не может изменить расписание. Это действие может сделать только администратор данных. Локальное представление администраторов данных является одновременно и глобальным представлением, то есть полной концептуальной моделью.

Для нашего случая отличиями полной концептуальной модели от локальной модели сотрудников предметной комиссии будут представление сущностей РАСПИСАНИЕ, СПЕЦИАЛЬНОСТИ, АБИТУРИЕНТ в виде совокупностей нескольких сущностей, а также добавление сущности, необходимой для автоматического зачисления абитуриентов.

Разбиение сущности СПЕЦИАЛЬНОСТЬ на несколько сущностей было описано в пункте «выявление и моделирование сущностей и связей».

Сущность АБИТУРИЕНТ (рис. 25) разбивается на несколько сущностей в связи с тем, что при изменении базы данных может начать дублироваться информация о гражданстве, предшествующем образовании и о полученной медали (дипломе с отличием). Обоснование данного разбиения предоставляется читателю.

Сущность РАСПИСАНИЕ (рис. 26).

В приведенной на рисунке диаграмме показано не только разбиение сущности РАСПИСАНИЕ, но и связанные сущности АБИТУРИЕНТ и ОЦЕНКА. Обоснование данного разбиения также предоставляется читателю, а формальное объяснение такого разбиения для случая реляционной модели приводится в соответствующем параграфе.

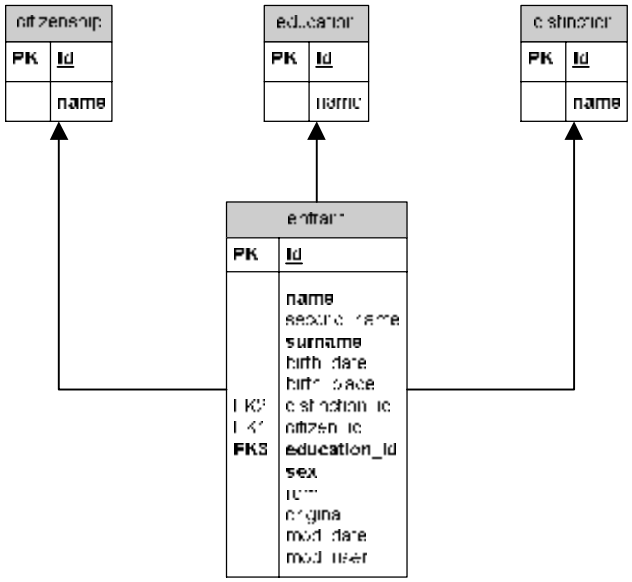


Рис. 25. Разбиение сущности АБИТУРИЕНТ

Помимо уже выделенных сущностей, нам потребуется еще одна. Она будет использоваться при создании алгоритма автоматического зачисления. Это сущность ПРЕДВАРИТЕЛЬНЫЕ ПРИКАЗЫ (pre_command).

Необходимость создания этой сущности неочевидна, исходя из тех данных, которые нам известны сейчас. Для понимания того, зачем нужна эта сущность, рассмотрим более подробно процедуру зачисления.

Итак, экзамены закончились, и абитуриенты получили некоторые оценки. Общий проходной балл вычисляется так: количество баллов, набранное на экзаменах, умножается на два, и к полученному числу добавляются десятые и сотые доли балла, исходя из документов, которые имеет абитуриент. Например, наличие медали или диплома с отличием приводит к добавлению 0,5 балла. Существуют и другие признаки, которые учитываются более сложным образом. Например, похвальная грамота по предмету засчитывается как 0,1 балла только в том случае, если этот предмет (или аналогичный) входит в перечень экзаменов. Если на ВМК ННГУ нужно сдавать математику, то похвальная грамота по алгебре или геометрии даст абитуриенту 0,1 балла.

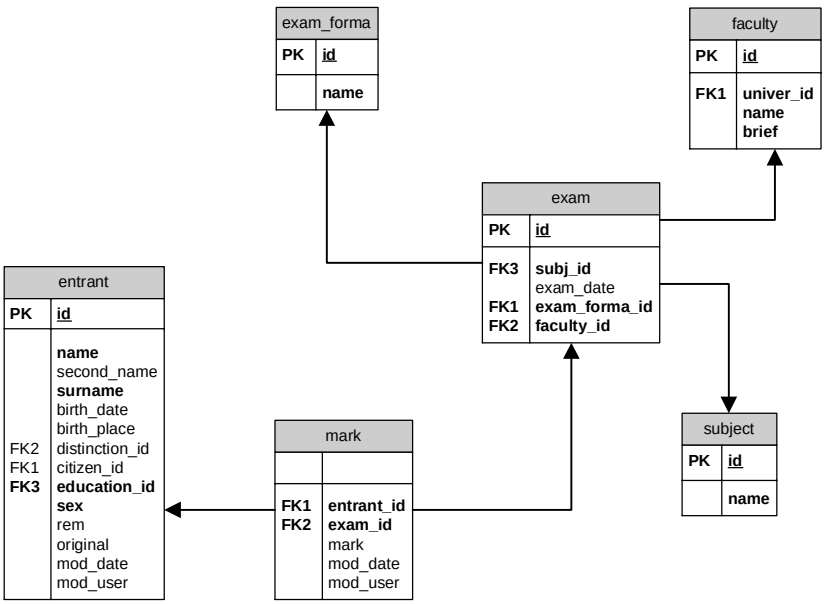


Рис. 26. Разбиение сущности РАСПИСАНИЕ

Существуют и более сложные признаки, сформулировать которые в общем виде очень трудно. Все это приводит к тому, что решение о добавлении баллов принимает работник приемной комиссии. Абитуриенту предоставляется возможность оспорить данное решение. Таким образом, в сущности ПРЕДВАРИТЕЛЬНЫЕ ПРИКАЗЫ хранится информация об общем набранном абитуриентом балле на каждую специальность, на которую он претендует. Этот балл складывается из результатов экзаменов и добавленных вручную десятых и сотых долей балла.

Рассмотрим ситуацию, когда дополнительных баллов нет. Обосновано ли в таком случае создание сущности ПРЕДВАРИТЕЛЬНЫЕ ПРИКАЗЫ? Конечно, нет. Если пользователь не меняет балл, набранный на экзаменах, то вся информация, хранящаяся в данной сущности, является производной, то есть может быть получена путем сложного запроса к другим сущностям.

Атрибуты сущности ПРЕДВАРИТЕЛЬНЫЕ ПРИКАЗЫ.

- Связующие атрибуты (entrant_id, prog_id).
- Набранный абитуриентом балл (ball).
- Информация о том, кто последним модифицировал данный экземпляр сущности, и когда это было сделано (mod_user, mod_date).
- Признак того, что экземпляр сущности проверен и не нуждается в автоматической обработке (saved). Данный атрибут требует пояснений. До проведения собеседования атрибут «набранный балл» является вычислимым, так как он формируется, исходя из полученных абитуриентом оценок. Было бы слишком долго считать балл вручную, поэтому в базе данных должна существовать хранимая процедура, которая заполняет таблицу «предварительные приказы» начальными значениями. В качестве начального значения для атрибута «набранный балл» выбирается суммарная оценка на экзаменах, умноженная на два.

Предположим, что мы запустили данную процедуру, которая внесла информации в сущность ПРЕДВАРИТЕЛЬНЫЕ ПРИКАЗЫ. Один из факультетов провел собеседование и изменил информацию по некоторым абитуриентам. В этот момент выяснилось, что некоторые оценки на другом факультете были внесены неверно. Мы должны или запустить процедуру заново, но тогда мы теряем результаты уже проведенного собеседования, или изменять результаты вручную, но тогда резко увеличивается вероятность ошибки.

Выходом в этой ситуации и является добавление атрибута «saved». Если данный атрибут установлен в «истинно», то данный экземпляр сущности не должен быть подвергнут изменению.

Итоговая диаграмма «сущность-связь» получается большая и приведена на рисунке в сокращенном виде. Для воссоздания диаграммы полностью, необходимо использовать фрагменты диаграмм, рассмотренные выше.

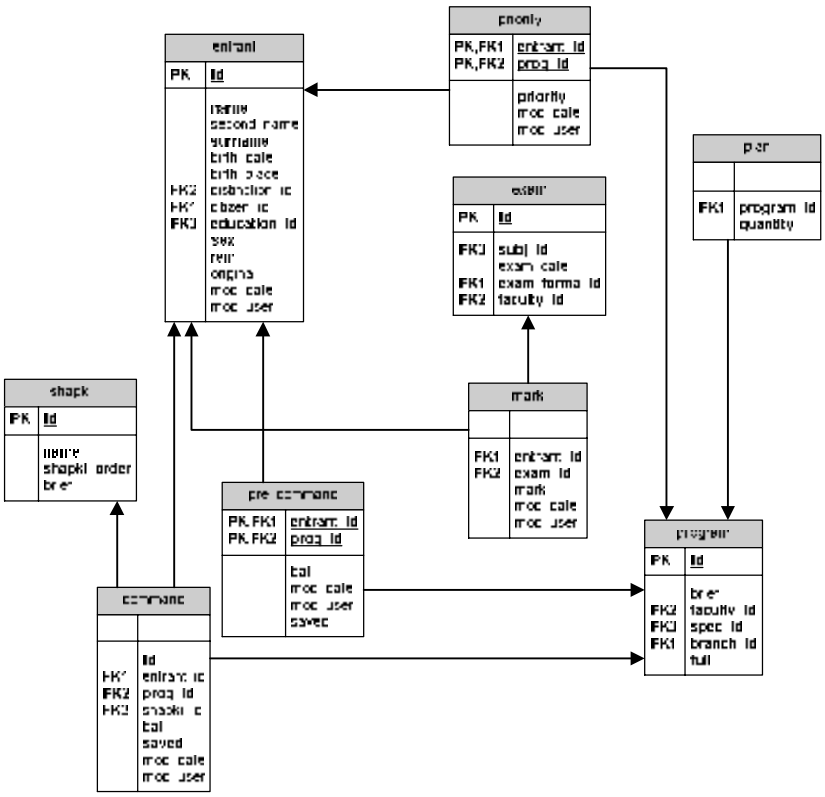


Рис. 27. Общий вид ER-диаграммы

2.9. Ограничения целостности

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

Огромный объем вводимых данных в базу данных, причем разные данные могут вводиться разными пользователями, обуславливает 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 2 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 можно разделить на две группы.

1. Внешние ограничения.

   Эти ограничения связаны с адекватностью отражения предметной областью. 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.

2. Ограничения, описанные с помощью специальных конструкций.

   Например, база демографических показателей содержит, в частности, атрибуты ЧИСЛО МУЖЧИН, ЧИСЛО ЖЕНЩИН, ЧИСЛО ЛИЦ ОБЕИХ ПОЛОВ (по разным возрастным группам, по разным регионам и т.п.) Для контроля достоверности вводимых данных можно использовать следующую формулу.

   ЧИСЛО ЛИЦ ОБЕИХ ПОЛОВ = ЧИСЛО МУЖЧИН + ЧИСЛО ЖЕНЩИН

   Если равенство не достигается, какое-то из введенных чисел недостоверно.

   Такие конструкции строятся, исходя из специфики данных рассматриваемой предметной области. Можно 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.

2.10. 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. Использование спиральной модели 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.

Напомним, что альтернативным способом является каскадная схема разработки программного обеспечения. Каскадный подход хорошо подходит для тех задач, для которых в самом начале разработки можно достаточно полно и точно сформулировать все требования заказчика. В случае построения баз данных каскадный подход является неприемлемым.

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 внесению изменений может оказаться трудоемкой и, что самое неприятное, может 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 создается 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, которые будут в конечном продукте 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 диаграмм «Сущность-Связь» 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 собственное программный продукт автоматизированного проектирования. Это Oracle Designer (Oracle), PowerDesinger (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). Программа AllFusion Process Modeler предназначена для моделирования бизнес-процессов. Результатами ее работы могут быть не только диаграммы, но и сгенерированный для различных сред код для доступа к базам данных. Для этого, однако, необходимо еще создать диаграммы «сущность-связь» с помощью AllFusion ERwin Data Modeler.

Поскольку данный учебный курс не предполагает знакомство со средствами описания бизнес-процессов, мы рассмотрим только ERwin Data Modeler – программный продукт, непосредственно использующийся при создании баз данных.

По данным Interface Ltd. AllFusion ERwin Data Modeler (ранее: ERwin) позволяет проектировать, документировать и сопровождать базы данных, хранилища данных и витрины данных (data marts).

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. (www.interface.ru) доступна для загрузки демонстрационная версия AllFusion ERwin Data Modeler, которая представляет собой полнофункциональную версию, ограниченную по времени.

Основные характеристики AllFusion ERwin Data Modeler:

§ Поддержка различных способов записи диаграмм «Сущность-Связь» (нотаций).

Нотация диаграмм «Сущность-Связь» в AllFusion ERwin Data Modeler несколько отличается от нотации классических моделей «сущность-связь».

Эти отличия проявляются, прежде всего, в отсутствии атрибутов у связи. Наличие атрибутов у связи при классическом способе записи свидетельствует, что внутри связи скрыта некоторая сущность. В нотациях, используемых в AllFusion ERwin Data Modeler, связь не может иметь атрибутов, и изображается в виде линии, а ее название пишется выше и/или ниже линии.

Атрибуты внутри сущности в AllFusion ERwin Data Modeler разделяются на две группы линией. Ключевые атрибуты находятся сверху черты, остальные атрибуты снизу.

Сильная сущность обозначается прямоугольником с прямыми углами, слабая сущность – прямоугольником с закругленными углами. Идентифицирующая связь – сплошная линия, не идентифицирующая – прерывистая.

- Поддержка проектирования информационных хранилищ.
- Поддержка совместного проектирования.
- Поддержка триггеров, хранимых процедур и шаблонов.
- Развитые средства проверки корректности моделей данных Reverse Engineering (генерация модели данных на основе анализа существующей базы данных), включая восстановление связей по индексам.
- Автоматическая генерация SQL DDL для создания баз данных.
- Полная совместимость и поддержка 20-ти типов СУБД.

Приведем в качестве примера использования данного CASE-средства диаграмму «Сущность-Связь» фрагмента базы данных абитуриентов. В сущности entrant сохраняются анкетные данные об абитуриентах (entrant). Помимо этой сущности на диаграмме приведены сущности-справочники: виды предшествующего образования (education), данные о гражданстве (citizenship), а также данные о наличии у абитуриента золотой или серебряной медали или диплома с отличием (distinction). Сущности связаны между собой связью «имеет». Абитуриент имеет предшествующее образование, гражданство и данные об отличии диплома или аттестата.

На рисунках 28 и 29 приведены диаграммы «Сущность-Связь» на логическом и физическом уровне. Разделение на уровни используется в ERwin для удобства – на самом деле диаграмма одна, однако в

зависимости от потребностей разработчика, она может быть представлена в двух видах. Необходимо отметить, однако, что можно создавать элементы, которые присутствуют только на логическом или только на физическом уровне.

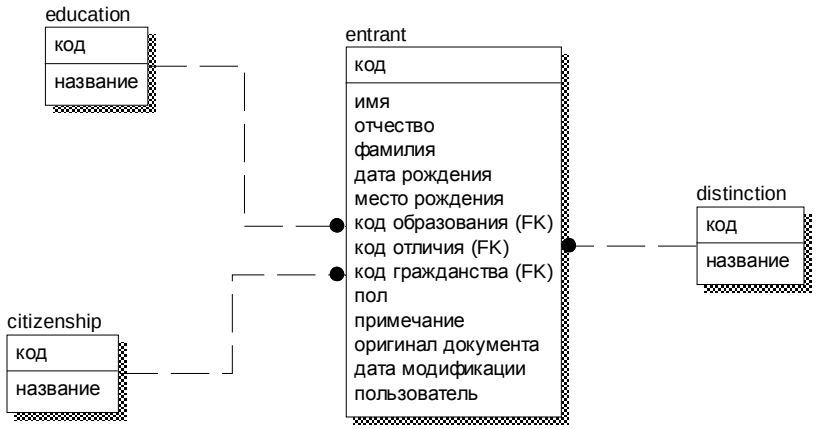


Рис. 28. Логический уровень ER-диаграммы

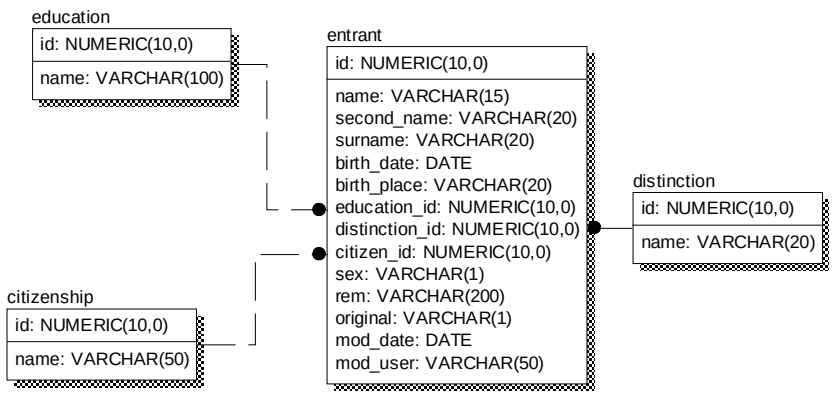


Рис. 29. Физический уровень ER-диаграммы

Логический уровень позволяет сконцентрироваться на описании бизнес-правил, а физический уровень – на представлении модели в конкретной СУБД. В частности, названия полей могут отличаться на логическом и физическом уровне. В нашем примере в СУБД используются английские названия полей, а на логическом уровне мы определяем русские названия.

На физическом уровне диаграмма будет выглядеть по-разному в зависимости от того, для какого сервера баз данных будет использоваться данная модель, и, соответственно, какие типы данных есть в данной СУБД. Приведенная на рисунке 29 диаграмма построена для СУБД Oracle 8.0.

После создания модели мы можем сгенерировать с помощью CASE-средства соответствующую структуру баз данных. Генерация осуществляется с помощью выполнения автоматически создаваемого скрипта. Настройки, задаваемые при генерации, влияют на полученный скрипт.

Фрагмент скрипта для создания структур баз данных приведен ниже.

```
…
CREATE TABLE education (
       name                 VARCHAR(100) NULL,
       id                   NUMERIC(10,0) NOT NULL,
       PRIMARY KEY (id)
);
CREATE TABLE distinction (
       name                 VARCHAR(20) NOT NULL,
       id                   NUMERIC(10,0) NOT NULL,
       PRIMARY KEY (id)
);
CREATE TABLE citizenship (
       name                 VARCHAR(50) NOT NULL,
       id                   NUMERIC(10,0) NOT NULL,
       PRIMARY KEY (id)
);
CREATE TABLE entrant (
       name                 VARCHAR(15) NOT NULL,
       id                   NUMERIC(10,0) NOT NULL,
       second_name          VARCHAR(20) NULL,
       surname              VARCHAR(20) NOT NULL,
```

```
        birth_date           DATE NULL,
        birth_place          VARCHAR(20) NULL,
        education_id         NUMERIC(10,0) NOT NULL,
        distinction_id       NUMERIC(10,0) NOT NULL,
        citizen_id           NUMERIC(10,0) NOT NULL,
        sex                  VARCHAR(1) NOT NULL,
        rem                  VARCHAR(200) NULL,
        original             VARCHAR(1) NOT NULL,
        mod_date             DATE NULL,
        mod_user             VARCHAR(50) NULL,
        PRIMARY KEY (id),
        FOREIGN KEY (education_id)
                              REFERENCES education,
        FOREIGN KEY (distinction_id)
                              REFERENCES distinction,
        FOREIGN KEY (citizen_id)
                              REFERENCES citizenship
);
…
```

Окончание скрипта не приведено.

CASE средство ERwin Data Modeler в совокупности с AllFusion Process Modeler и другими продуктами фирмы AllFusion представляет стандартные возможности, характерные и для других CASE средств, таких как Oracle Designer и PowerDesigner (Sybase).

ГЛАВА 3. МОДЕЛИ ДАННЫХ СУБД КАК ИНСТРУМЕНТ ПРЕДСТАВЛЕНИЯ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ

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, составляющих модель данных СУБД. Следующий этап проектирования базы данных состоит в представлении концептуальной модели средствами модели 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.

3.1. Общие представления о модели данных

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

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

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

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

Агрегат данных – поименованная совокупность элементов данных внутри записи, которую можно рассматривать как единое целое.

дата		
число	месяц	год

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 «агрегат», «запись».

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

Важнейшим понятием концептуальной модели является понятие связи между сущностями (наборами сущностей). В моделях данных, 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 описывается следующим образом:

- определяются типы и характеристики логических структур данных (полей, записей, файлов);

- описывается правила составления структур более общего типа из структур более простых типов;
- описываются возможные действия над структурами и правила их выполнения, включающие:
  - основные элементарные операции над данными;
  - обобщенные операции (процедуры);
  - средства контроля относительно простых условий корректности ввода данных (ограничения);
  - средства контроля сколь угодно сложных условий корректности выполнение определенных действий (правила).

В качестве основных элементарных операций обычно рассматриваются следующие: поиск записи с заданным значением ключа, чтение нужной записи, добавление записи, корректировка, удаление.

В моделях данных 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 как для реализации внешних ограничений концептуальной модели, так и для реализации 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.

3.2. Сетевая модель данных

Это одна из наиболее ранних моделей данных. Типовая сетевая модель данных была предложена рабочей группой по базам данных (Data Base Task Group –DTBG) 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. Результаты соответствующей работы комитета CODASYL изложены в [13]. Структура данных сетевой модели определяется в терминах понятий раздела 3.1. (элемент, агрегат, запись, группа, групповое отношение, файл, база данных).

Реализация групповых отношений в сетевой модели осуществляется с использованием 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, 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. 30).

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 образуют некую сетевую структуру (ориентированный граф общего вида). Вершинами графа являются группы; дугами графа, направленными от владельца к члену группового отношения, являются связи между группами.

Сетевая модель данных поддерживает все необходимые операции над данными, реализованные как действия со списковыми структурами.

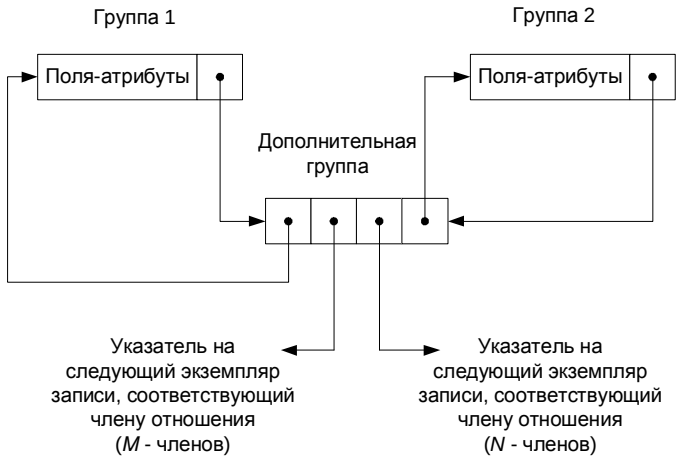


Рис. 30. Представление связей типа *M*:*N*

Сетевая модель данных является, вероятно, наиболее общей моделью данных 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.

Наиболее существенным недостатком, на наш взгляд, 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 данными. 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 системы IDMS, UNIBAD (БАНК), 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.

3.3. Иерархическая модель данных

Это также одна из наиболее ранних моделей данных. Реализация групповых отношений в иерархической модели, как и 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 предков);
   - 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 указать вышеуказанные недостатки сетевой модели.

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

3.4. 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 должно приводить к реструктуризации всей модели данных и базы данных в целом.

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

Основными используемыми понятиями здесь также являются поле, запись, файл. Структура записи определяет структуру таблицы,

содержащей экземпляры соответствующей записи. Столбцы таблицы представляют собой имена полей записи, строки таблицы представляют собой экземпляры записи. Таким образом, понятие «таблица» здесь соответствует понятию «файл» модели данных.

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.

Для формального описания таблицы используется теоретико-множественное понятие отношения. Список названий столбцов таблицы (имен полей записи, соответствующих атрибутам), называют схемой отношения и обозначают $R\ (A_1, A_2, \ldots, A_n)$.

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

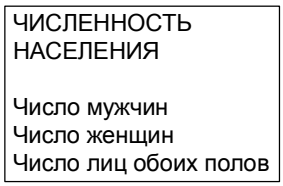
В качестве основного недостатка реляционной модели можно указать дублирование информации при представлении связей.

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 далее.

3.5. 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. Например: сущность ЧИСЛЕННОСТЬ НАСЕЛЕНИЯ



Значение атрибутов зависит от параметров «год», «административный район». Если использовать для описания соответствующей концептуальной схемы реляционную модель, то необходимо вводить множество таблиц ЧИСЛЕННОСТЬ НАСЕЛЕНИЯ по каждому году для каждого района. Так, при 60 административных районах и необходимости анализировать данные за 10 лет число таблиц будет равно 600. 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 связано с многомерностью визуализации.

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

Измерение – упорядоченный набор значений, принимаемых конкретным параметром, и соответствующий одной из граней гиперкуба. Для нашего примера можно указать в качестве измерений: год – 1998, 1999, 2000, 2001, 2002, 2003; область – Моск5овская, Ленинградская, Нижегородская и т.д.

Ячейка или показатель – это поле, соответствующее атрибуту сущности, значение которого однозначно определяется фиксированным набором значений параметров (значениями «измерений», например, 2003 г., Нижегородская область).

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 пользователю из гиперкуба, например, суммируя значения показателей по всем значениям одного из параметров, например, по всем областям.

Такая модель данных 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 и Cache.

ГЛАВА 4. ФОРМАЛИЗАЦИЯ РЕЛЯЦИОННОЙ МОДЕЛИ

4.1. Формализованное описание отношений и схемы отношений

Как уже отмечалось в п.3.4. реляционная модель описывает представление данных в виде двумерной таблицы, называемой отношением. Наименованиями столбцов этой таблицы служат имена атрибутов. Рассмотрим формализованное описание соответствующих понятий.

Схемой отношений $R(A_1, A_2,\ldots, A_n)$ называется конечное множество имен атрибутов $\{A_1, A_2, \ldots, A_n\}$, где n – арность схемы отношения. Понятию схема отношения соответствует описание структуры двумерной таблицы (имена столбцов). 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.

Пусть $D = D_1 È D_2 È \ldots È D_n$

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:$\{ D_1 È D_2 È \ldots È D_n \}$, таких, что $t_k(A_i) \in D_i$, $k = \overline{1, p}$; $i = \overline{1,n}$.

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

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

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

Ключом (первичным ключом) отношения r со схемой R называется 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 $K=\{A_{i1}, A_{i2}, \ldots, A_{im}\}$.

Таким образом, достаточно знать значение кортежа на множестве *K*, чтобы однозначно его идентифицировать.

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

Отметим следующие свойства отношения:

1. Порядок рассмотрения атрибутов в схеме отношения (отношении) не имеет значения, т.к. для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута.
2. 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.
3. Порядок рассмотрения кортежей в отношении не имеет значения, т.к. отношение представляет собой множество кортежей, а элементы множества, по определению теории множеств, неупорядочены.

4.2. 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.

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 эти 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.

Механизмы реляционной алгебры и реляционного исчисления эквивалентны, т.е. для любого допустимого выражения реляционной алгебры можно построить эквивалентную формулу реляционного исчисления и наоборот. Почему же в реляционной модели данных присутствуют оба эти механизма?

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

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

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

4.3. 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* È *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 к1 и к2 соответственно. Декартовым произведением *r*×*s* называется множество кортежей длины к1+к2, первые к1 компонентов которых образуют кортежи, принадлежащие *r*, а последние к2 – 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 $p_{Ai_1, Ai_2, \ldots, Ai_m}(r)$

Проекция $p_{Ai_1, Ai_2, \ldots, Ai_m}(r)$ есть множество кортежей, получаемых из кортежей отношения *r* выбором столбцов с именами $Ai_1, Ai_2, \ldots, Ai_m$

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

$p_{1,3}(r)$

a	c
d	f
c	d

Выбор (селекция) $s_F(r)$

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

$s_{(1)=(3)}(r)=$

A	B	C

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

Приведем ряд примеров представления запросов в задаче зачисления абитуриентов с помощью формальных операций.

Пример 1.

Сформировать список абитуриентов (фамилия, имя, отчество).

Рассмотрим сущность АБИТУРИЕНТ (*entrant*), представленную в ER-диаграмме рис. 23. Атрибут «Фамилия» обозначен здесь «*surname*». Для ответа на запрос необходимо взять проекцию отношения entrant на столбец *surname*.

$$p_{surname}(entrant)$$

Пример 2.

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

Рассмотрим то же отношение entrant. Сначала выбираем абитуриентов, которым больше 35 лет. $s_{birth\_date+35<tek\_date}(entrant)$.

Затем берем проекцию полученного отношения на столбцы *surname* и *birth_date*.

$$p_{surname,berth\_date}(s_{birth\_date+35<tek\_date}(entrant))$$

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 абитуриентов, получивших оценку «неудовлетворительно». Рассматриваем ту же ER-диаграмму на рис. 23.

Оценка абитуриента является атрибутом отношения *mark*. Если бы в этом отношении присутствовал атрибут «фамилия», то задача решалась бы аналогично примеру 2. В отношении mark присутствует атрибут «идентификатор абитуриента» – *entrant_id*, а фамилия присутствует в отношении *entrant*. Для ответа на этот запрос необходимо связывать по *entrant_id* отношение mark и отношение *entrant*.

Сначала выберем из отношения mark кортежи с оценкой «*неудовлетворительно*». Обозначим полученное отношение *R*1. (Дальнейшие промежуточные отношения будем обозначать последовательно *R*2, *R*3 и т.д.).

$$R1 = s_{mark=«неудовлетворительно»}(mark)$$

Далее нас будет интересовать только атрибуты *entrant_id* и *mark*. Поэтому возьмем проекцию на эти столбцы.

$$R2 = p_{entrant\_id,mark}(R1)$$

Далее необходимо связать отношения *entrant* и *R*2 (склеить таблицы). Для склейки таблиц используется операция декартово произведение.

$$R3 = entrant \times R2.$$

В отношении *R*3 присутствуют два одинаковых столбца: *entrant_id* из отношения *entrant* и отношения *R*2. Выбирая из отношения *R*3 строки, в которых значение *entrant_id* в соответствующих столбцах совпадают, получим сведения об абитуриентах, получивших оценку неудовлетворительно.

$$R4 = s_{entrant.\ entrant\_id\ =\ R2.\ entrant\_id}(R3)$$

entrant. entrant_id и *R*2. *entrant_id* – обозначают соответственно столбец *entrant_id* соответствующей первой и второй составной части декартова произведения. Теперь осталось только выбрать фамилии соответствующих студентов

$$R5 = p_{surname}(R4)$$

Получаем требуемый результат. Заметим, что для экономии действий и памяти, перед тем как склеивать таблицы, целесообразно было сделать операцию проекция отношения *entrant* на столбцы *entrant_id, surname* (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* Ç *s*

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

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

***q*-соединение** $r \underset{iq\ j}{><} s$

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

$$r \underset{iqj}{><} s = s_{\,iq\,(l+j)}(r \times s),$$

где *l* – арность отношения *r*.

Пример.

r

1	2	3
4	5	6
7	8	9

s

3	1
6	2

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

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 обуславливает 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 $r >< s$

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

Пусть

$r = (A1, ..., Ak, B1, ..., Bn)$, $s = (A1, ..., Ak, C1, ..., Cm)$,

имена $A1, ..., Ak$ совпадают.

Тогда $r >< s$ определяется следующим образом

$r >< s = \Pi_{B1,...,Bn,A1,...,Ak,C1,...,Cm}(s_{r.A1=s.A1\&r.A2=s.A2\&...r.Ak=s.Ak}(r \times s))$.

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».

4.4. Использование формального аппарата для оптимизации схем отношений

4.4.1. Проблема выбора рациональных схем отношений

При представлении концептуальной схемы в виде реляционной модели возможны различные варианты выбора схем отношений. Одни варианты выбора схем отношений рассматривались в предыдущих разделах (п.3.4.), другие варианты 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 об абитуриентах, специальностях, формах обучения включены в таблицу, содержащую данные об атрибутах. Схема отношения имеет следующий вид:

АБИТУРИЕНТ (код абитуриента, Ф.И.О., факультет, специальность, форма обучения).

Эта схема отношений обуславливает следующее недостатки соответствующей базы данных:

- Дублирование информации (избыточность).

  У абитуриентов, поступающих на один факультет будет повторяться название факультета. Для разных факультетов будут повторяться одинаковые формы обучения, специальности.
- Потенциальная противоречивость (аномалии обновления).

Если, например, изменится название специальности, то, изменяя её в одном кортеже (у одного абитуриента), необходимо изменять и во всех других кортежах, где она присутствует.

§ Потенциальная возможность потери сведений (аномалии удаления).

При удалении всех абитуриентов, поступающих на определенную специальность, мы теряем все сведения об это специальности.

§ Потенциальная возможность не включения информации в базу данных (аномалии включения).

В базе данных будут отсутствовать сведения о специальности, если на нее абитуриенты не подали заявления.

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

Построение рационального варианта схем отношений (обладающих лучшими свойствами при операциях включения, модификации и удаления данных, чем все остальные наборы схем) осуществляется путем так называемой нормализации схем отношений. Нормализация схем отношений 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 являющимися множеством или кортежем из более элементарных составляющих. 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 нормальную форму. Построение соответствующих форм описано в [24, 28 и др.], и в данной книге не рассматривается.

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

4.4.2. Функциональные зависимости (зависимость между атрибутами отношения)

Пусть R ($A_1, A_2,\ldots, A_n$) – схема отношения, а X и Y – подмножества $\{A_1, A_2,\ldots, A_n\}$.Тогда X функционально определяет Y или Y функционально зависит от X (обозначается X **à** 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. 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 функциональной зависимости:

Код Абитуриента **à** Фамилия, Имя, Отчество.

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 $A_1A_2\ldots A_n$, F – некоторое множество функциональных зависимостей и X – некоторое подмножество R. Тогда X называется ключом, если, во-первых, в F^+ существует зависимость X **à** $A_1A_2\ldots A_n$ и, во-вторых, ни для какого

подмножества Y, входящего в X, зависимость $Y \to A_1A_2\ldots A_n$ 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 задана некоторая схема отношения R $\{A_1, A_2,\ldots,A_n\}$ с множеством атрибутов $U=\{A_1, A_2,\ldots,A_n\}$ и множество функциональных зависимостей F, заданных на данном множестве U.

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

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

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

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. $X \to X$

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

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

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

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

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

4.4.3. Декомпозиция схемы отношения

Последовательный переход от одной нормальной формы к другой при нормализации схем отношений осуществляется с помощью декомпозиции. 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.

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. В этом случае говорят, что декомпозиция обладает свойством соединения без потерь.

Алгоритм декомпозиции основан на следующей теореме.

Теорема Фейджина. Пусть $R(A, B, C)$ отношение, A, B, C – атрибуты.

Если R удовлетворяет зависимости $A \rightarrow B$, то R равно соединению его проекций A, B и A, C

$$R(A, B, C) \qquad R(A, B)$$
$$R(A, C)$$

При нормализации необходимо выбирать такие декомпозиции, которые обладают свойством соединения без потерь. Для проверки, обладает ли декомпозиция данным свойством используется специальный алгоритм проверки [28, 40]. Алгоритм состоит в следующем.

Пусть есть некоторая схема отношения $R = A_1 \ldots A_n$, множество функциональных зависимостей F и некоторая декомпозиция $(R_1, \ldots R_k)$ исходной схемы, состоящая из k подсхем.

Необходимо построить таблицу с n столбцами и k строками. Столбец j соответствует атрибуту A_j, строка k схеме отношения R_k. На пересечении строки i и столбца j поместим символ a_j, если A_j принадлежит R_i. В противном случае поместим туда символ b_{ij}.

Рассматриваем каждую зависимость из множества F до тех пор, пока в таблице невозможно сделать какие-либо изменения. Всякий раз, рассматривая зависимость $X \rightarrow Y$, мы ищем строки, которые совпадают по всем столбцам, соответствующим атрибутам из X. При

обнаружении таких строк, отождествляем символы в столбцах, соответствующих атрибутам из Y. Если при этом один из отождествляемых символов равен a_j, то приравниваем и другой a_j. В том случае, когда они равны b_{ij} и b_{lj}, делаем их оба равными b_{ij} или b_{lj} по своему усмотрению.

После модификации строк таблицы указанным выше способом может обнаружиться, что некоторая строка стала равной $a_1 \ldots a_k$. Тогда декомпозиция обладает свойством соединения без потерь. Если такой строки не получается, то декомпозиция не обладает таким свойством.

Пример.

Декомпозиция схемы $ABCD$ на AB и ACD.

$A \rightarrow B$, $AC \rightarrow D$ (AC – сокращенная запись $A \cup C$).

A	B	C	D
a1	a2	b13	b14
a1	b22	a3	a4

Поскольку $A \rightarrow B$ и две строки совпадают по A, то

A	B	C	D
a1	a2	b13	b14
a1	a2	a3	a4

Так как одна строка состоит из всех a, то мы наша декомпозиция обладает свойством соединения без потерь.

Вторым важнейшим желательным свойством декомпозиции является свойство сохранения функциональных зависимостей.

Стремление к тому, чтобы декомпозиция сохраняла зависимости, является естественным. Функциональные зависимости являются некоторыми ограничениями на данные. Если декомпозиция не обладает данным свойством, то для того чтобы проверить, не нарушают ли введенные данные 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.

4.4.4. 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 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 во 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 $X \to A$, 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 ЗНФ. Иными словами любое отношение, находящееся в НФБК, находится в ЗНФ. Обратное неверно.

Пример. Расписание консультаций. Каждая группа может прийти на консультацию один раз в день. Для проведения консультации или консультаций преподавателю на определенный день предоставляется аудитория. В течение дня данная аудитория может использоваться разными преподавателями.

РАСПИСАНИЕ КОНСУЛЬТАЦИЙ (Группа, Дата, Время, Преподаватель, Аудитория)

В качестве первичного ключа выбрали «Группа, Дата».

Потенциальные ключи:

«Группа, Дата».

«Преподаватель, Дата, Время».

«Аудитория, Дата, Время».

Группа	Дата	Время	Преподаватель	Аудитория
721	13 мая	10.30	Визгунов	401
722	13 мая	12.00	Визгунов	401
723	13 мая	12.00	Трифонов	402
722	1 июня	10.30	Визгунов	402

Все атрибуты зависят от всего ключа. Нет неполных зависимостей. Нет транзитивных зависимостей.

Однако существует следующая зависимость – Преподаватель, Дата определяют (**à**) Аудитория.

Нарушается определение формы БК – данный составной атрибут является детерминантом, но не является потенциальным ключом.

Данная функциональная зависимость не нарушает третьей нормальной формы, однако порождает аномалии обновления.

Преподавателю Визгунову на 13 мая выделена аудитория 401. Если по каким-то причинам администрация выделяет преподавателю другую аудиторию, то информация об этом должна быть внесена два раза, что может быть не сделано и, соответственно, быть причиной приведения базы данных в противоречивое состояние.

Выход – разбиение исходного отношения на два.

РАСПИСАНИЕ КОНСУЛЬТАЦИЙ (Группа, Дата, Время, Преподаватель)

АУДИТОРИИ (Преподаватель, Дата, Аудитория)

Важно отметить, что хотя такое разбиение (декомпозиция) приводит к устранению избыточности данных, оно также приводит к потере функциональной зависимости: Аудитория, Дата, Время перестают быть потенциальным ключом, так как теперь эти атрибуты находятся в разных отношениях.

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.

4.4.5. Пример нормализации до 3НФ

В настоящее время можно выделить два способа применения нормализации.

Во-первых, можно рассмотреть исходные документы предметной области, выписать все атрибуты, которые в них используются, свести их в одну таблицу и применить процедуру нормализации, основываясь на функциональных зависимостях. Полученное множество отношений будет являться структурой создаваемой базы данных.

Данный подход широко использовался на начальном этапе развития баз данных. Его основным недостатком является большая трудоемкость при выписывании всех функциональных зависимостей. В настоящее время в базах данных хранятся сотни и тысячи таблиц. Применение данного подхода в этом случае становится нерациональным, а, иногда и невозможным.

Второй способ, который в настоящее время широко используется на практике, является проверка построенной диаграммы «сущность-связь». Исходя из рассмотренной ранее теории данные должны храниться в таблицах, находящихся в третьей нормальной форме или в более высоких формах. Каждая сущность будет представлена в базе данных в виде таблицы или представления. Представления, в свою очередь, также будут выбираться из таблиц.

Таким образом, задача сводится к проверке нормализации все 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.

Рассмотрим пример из предметной области зачисления абитуриентов.

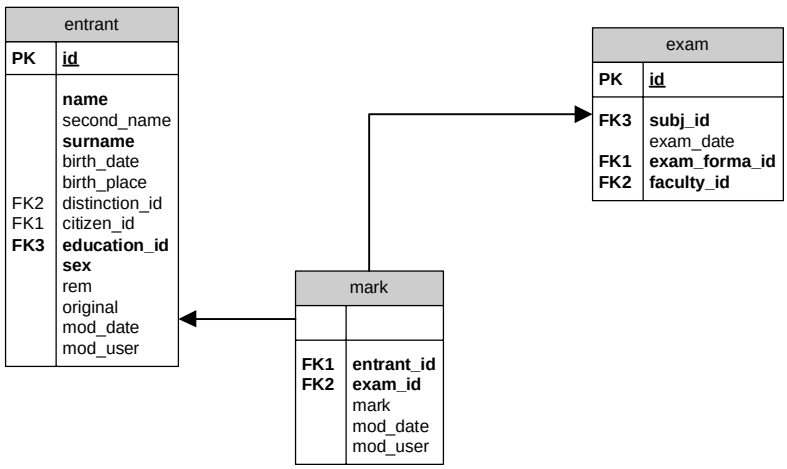


Рис. 31. Фрагмент ER-диаграммы

Таблица, соответствующая сущности «entrant», находится в третьей нормальной форме. Действительно, все атрибуты зависят от ключа (от поля id) и эта зависимость полная. Также в таблице отсутствуют транзитивные зависимости.

То же самое можно сказать про сущности «mark» и «exam». Проверка на соответствие третьей нормальной форме остальной части модели предоставляется в качестве упражнения читателю.

Рассмотрим пример применения первого подхода.

Пусть мы имеем набор ведомостей сданных экзаменов и хотим построить структуру таблиц для хранения данной информации.

Сведем все данные, которые у нас есть, в одну таблицу. Она будет выглядеть примерно так (для упрощения мы не рассматриваем часть атрибутов):

Код абиту-риента	Фамилия	Имя	Код экзамена	Предмет и дата	Оценка
1	Сергеев	Сергей	1	Математика	4
2	Иванов	Иван	1	1.08.03	5

1	Сергеев	Сергей	2	Физика	5
2	Иванов	Иван	2	5.08.03	5

Данная таблица ненормализованная, так как на пересечении строки и 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 абиту-риента	Фамилия	Имя	Код экзамена	Предмет	Дата	Оценка
1	Сергеев	Сергей	1	Математика	1.08.03	4
2	Иванов	Иван	1	Математика	1.08.03	5
1	Сергеев	Сергей	2	Физика	5.08.03	5
2	Иванов	Иван	2	Физика	5.08.03	5

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

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

Действительно, эта совокупность атрибутов функционально определяет имя абитуриента, фамилию абитуриента, предмет, дату экзамена и оценку. Ни код абитуриента, ни код экзамена по отдельности не определяют все атрибуты. Например, код абитуриента не определяет дату экзамена, а код экзамена не определяет фамилию абитуриента.

В данной таблице существуют неполные зависимости.

Код абитуриента функционально определяет Имя и Фамилию.

Код экзамена функционально определяет Предмет и Дату.

Выделим из нашей таблицы новые таблицы, соответствующие неполным зависимостям. Получаем следующие отношения:

- АБИТУРИЕНТЫ (Код абитуриента, Имя, Фамилия). Ключ – Код абитуриента.
- ЭКЗАМЕНЫ (Код экзамена, Предмет, Дата). Ключ – Код экзамена.

В исходном отношении остаются атрибуты Код абитуриента, Код экзамена, Оценка.

§ ОЦЕНКИ (Код абитуриента, Код экзамена, Оценка). Ключ, по-прежнему, совокупность атрибутов Код абитуриента и Код экзамена.

Полученные отношения находятся в третьей нормальной форме, так как в них нет неполных функциональных зависимостей.

Транзитивных зависимостей в данных отношениях тоже нет, следовательно, они находятся в третьей нормальной форме.

Таким образом, мы провели процесс нормализации для ведомостей экзаменов. В результате были получены три отношения в третьей нормальной форме.

4.4.6. 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, ФИО) 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 – отно-

шения 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 предназначен 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 сопровождение задачи.

ГЛАВА 5. ФИЗИЧЕСКИЕ МОДЕЛИ ДАННЫХ (СТРУКТУРЫ ХРАНЕНИЯ)

Как уже отмечалось, концептуальная схема, специфицированная к СУБД, отображается в структуру хранения автоматически 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 главе 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.

5.1. 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.

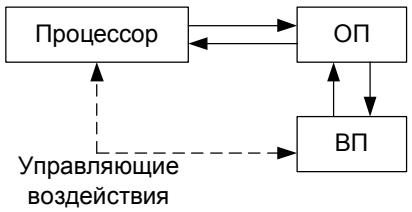


Рис.32 Схема работы ЭВМ

Как уже многократно отмечалось, базы данных создаются для работы с большими объемами данных, что обуславливает 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:

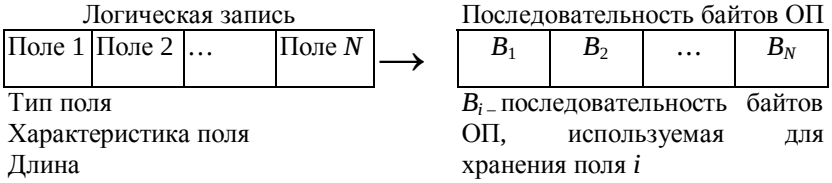
- единицей памяти является байт;
- память прямоадресуема (каждый байт имеет адрес);
- процессор выбирает для обработки нужные данные, непосредственно адресуясь к последовательности байтов, содержащих эти данные.

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.

5.2. Представление экземпляра логической записи

Логическая запись представляется в оперативной памяти следующим образом:



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 сетевой иерархической и реляционных моделях. В случае сетевой и иерархической моделях 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. Использование в СУБД других форматов 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 реальных задачах 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 не эффективно. 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 памяти является областью внешней памяти прямого доступа. В процессе ввода соответствующего значения в выделенной области памяти занимается столько памяти, какова длина этого значения.

На рис. 33 представлен пример вышеуказанного представления экземпляров записей из N полей, причем поле N – принимает значения соответственно разной длинны у разных экземпляров записей.

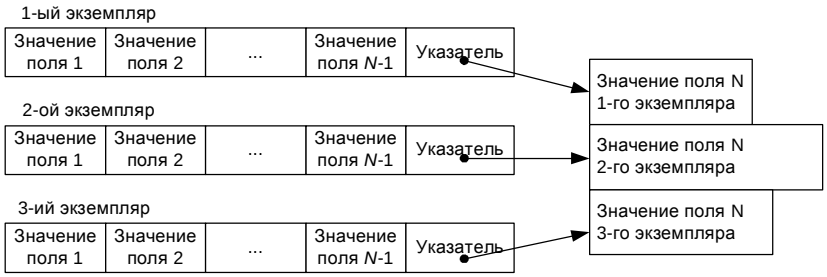


Рис. 33. Представление полей переменной длины

Конкретной реализацией такой схемы является поле типа MEMO в dbase-подобных СУБД (Dbase III+, FoxPro и т.д.).

5.3. 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 одной 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, называют 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*-ую физическую запись, изображен на рис. 34.

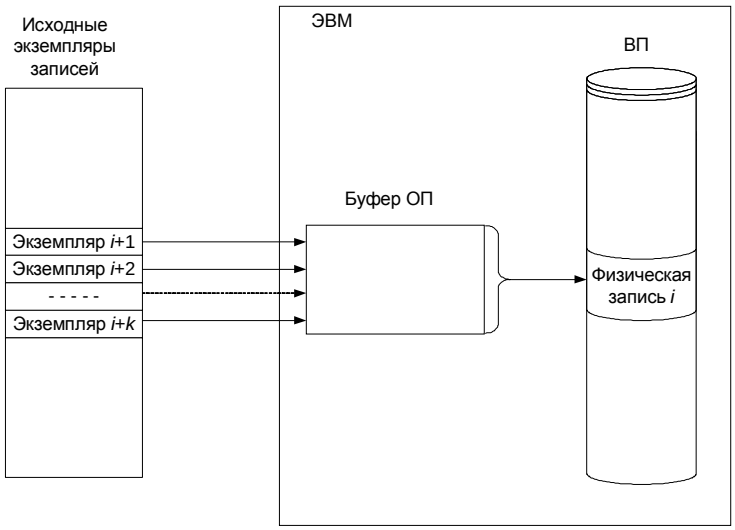


Рис.34. Схема занесения записей во внешнюю память

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.).

5.4. 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.

5.4.1. 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 соответствующего вычисления необходимо знать формат соответствующей физической памяти).

Физическая запись с номером I содержит логические записи с номерами

$$(I-1)*k+1$$
$$(I-1)*k+2$$
$$\dots$$
$$(I-1)*k+k$$
$$I=1, 2, \dots, \lceil N/k \rceil;$$

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 обращение 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 (блокировка), 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*-ую строку таблицы (*i*=1, 2, …, *n*). В этом случае читается физическая запись, с номером $\lceil i-1/k \rceil$, содержащая *i*-ую

логическую запись. Если соответствующая физическая запись содержит пустые логические записи, то добавляемая запись вставляется в этот блок, блок записывается на свое место во 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 во 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) составит $2*\lceil N/k \rceil/2$. Тогда суммарное число обращений к ВП при добавлении записи в этом случае будет равно $3+\lceil N/k \rceil$.

5.4.2. Последовательное размещение физических записей с упорядочением по ключу

В этом случае исходные логические записи упорядочиваются по ключу, затем объединяются в физические записи. Значением ключа физической записи является максимальное значение ключей логических записей соответствующего блока. Таким образом, физические записи становятся также упорядочены. Этот порядок определяет последовательность размещения записей в ВП. Заметим, что как и в предыдущем случае, можно вычислить физический адрес записи по номеру и непосредственно обратиться к записи по этому адресу. Рассмотрим, как реализуются основные элементарные операции.

Поиск записи с заданным значением ключа

Поиск осуществляется дихотомическим методом. Номер физической записи, соответствующей середине файла, определяется ближайшим целым числом к числу $\lceil N/k \rceil/2$. По этому номеру определяется номер физической записи и запись считывается в ОП. Проверяется, запись с заданным значением ключа лежит выше или ниже соответствующей записи. Соответствующая половина файла делится пополам, процесс повторяется до тех пор, пока не будет найдена искомая физическая запись.

$$\text{TP} = \text{C} \log_2 \lceil N/k \rceil,$$

где *C* некоторая константа.

После этого в ОП найденная запись разблокируется, в ней ищется нужная логическая запись.

Чтение записи

Аналогично операции поиска. Число обращений к ВП равно *TP*.

Корректировка и удаление записи

По аналогии с соответствующим описанием в п.5.4.1. Число обращений к ВП равно *TP*+1.

Добавление записи

Необходимость сохранения упорядочения по ключу обуславливает повышенную трудоемкость соответствующей операции. Прежде всего находится физическая запись, после которой должна быть размещена добавляемая запись. Читается следующая физическая запись и проверяется полнота соответствующего блока. Если блок не полный, логическая запись вставляется в этот блок, блок записывается на свое место во ВП. Если блок полон, то формируется новая физическая запись, которая будет состоять из одной добавляемой логической записи. Число обращений к ВП при этом равно *TP*+2. Для высвобождения места этой физической записи все последующие блоки переписываются на одну позицию ниже (сдвигаются). Сформированная физическая запись записывается на освободившееся место.

Среднее число обращений к ВП для перезаписи блоков (чтение+запись) оценивается, также как в п. 5.4.1. и составляет $\lceil N/k \rceil$.

Тогда суммарное число обращений к ВП при добавлении записи равно $TP+2+\lceil N/k \rceil$.

Заметим, что при таком способе организации структуры хранения время поиска, чтения, корректировки, удаления будет существенно меньше, чем в предыдущем случае: $C \log_2 \lceil N/k \rceil \ll (1+\lceil N/k \rceil)/2$.

Время добавления записей будет существенно больше. Кроме того, необходимо заметить, что, как правило, вышеуказанные операции производятся с записями, задаваемыми значениями целого ряда вторичных ключей, и упорядочение по первичному ключу не сокращает время соответствующих операций. В связи с этим, вышеуказанный способ организации структуры хранения в современных СУБД практически не используется.

5.4.3. Размещение физических записей в виде списковой структуры

Основная проблема в использовании в п. 5.4.1. и п. 5.4.2. способов 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. 35).

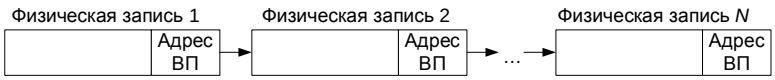


Рис.35. Список физических записей

Кроме этого списка во ВП формируется список свободных элементов («пустых» физических записей), элементы которого используются при вводе новой записи с данными (рис. 36).

Напомним, что каждая физическая запись состоит, как и ранее, из k логических записей.

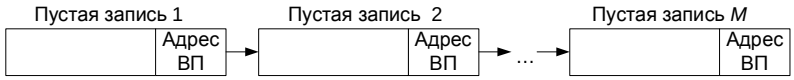


Рис. 36. Список свободных элементов

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 (в отличие от п. 5.4.2.) . 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 её 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 сравнивается 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 во 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 *РК*. 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.

5.4.4. 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 записи 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 на рис. 37.

Рассматриваемую структуру хранения называют еще инвертированным списком. Смысл этого термина состоит в следующем. Можно было бы упорядочить записи основной структуры хранения, объединив их в соответствующий упорядоченный список.

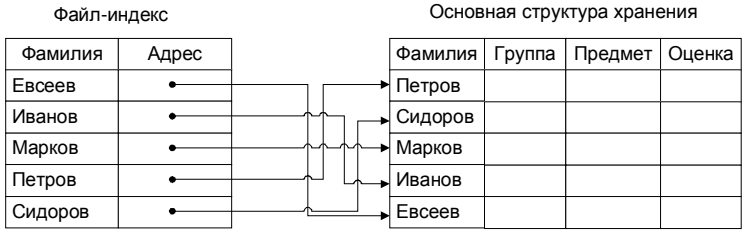


Рис. 37. Индексирование

В нашем случае адреса связи как бы удаляются из списка и включаются в состав файла-индекса (инвертируются). Поэтому полученная структура интерпретируется как инвертированный список.

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 производитс по аналогии с расчетами пп. 5.4.1.–5.4.3.

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 нахождения 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 во 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 во 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.

5.4.5. Бинарное дерево (B-дерево)

Структура бинарного дерева является следствием дальнейшего расширения концепции использования индексов (строится индекс над индексом) и представляет собой многоуровневые индексы.

Бинарное 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. Запись 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., до тех пор, пока количество записей индекса на определенном уровне будет не более *k*. Полученная структура изображена на рис. 38 (для упрощения рисунка на уровне 4 представлены только ключи логических записей и не представлены значения других полей этих записей).

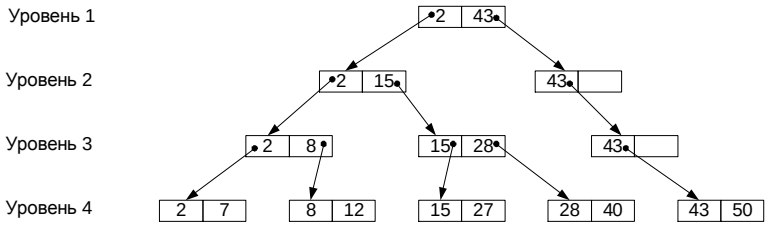


Рис. 38. B-дерево

В блоках указано значение ключа соответствующего блока. Значение *k* принято равным 2.

По построению бинарного 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 или равно чем значение ключа очередной записи индекса (если такая запись имеется), то по адресу связи, указанному в текущей

записи читается блок записей индекса следующего уровня. Далее процесс повторяется.

Считаем, что все блоки расположены во ВП. Тогда, число обращений к ВП при поиске информации будет равно числу уровней дерева. Число уровней дерева равно минимальному значению l, при котором выполняется $k^l \geq N$ (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 запись.

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 заносится 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 обуславливает 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. 38, 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 |
5. 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.
6. 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 |
7. 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.
8. 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 |.

9. 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.
10. 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 | |
11. Необходимо сформировать еще один уровень дерева | 2 | 43 |.

Полученная структура будет иметь вид, представленный на рисунке 39.

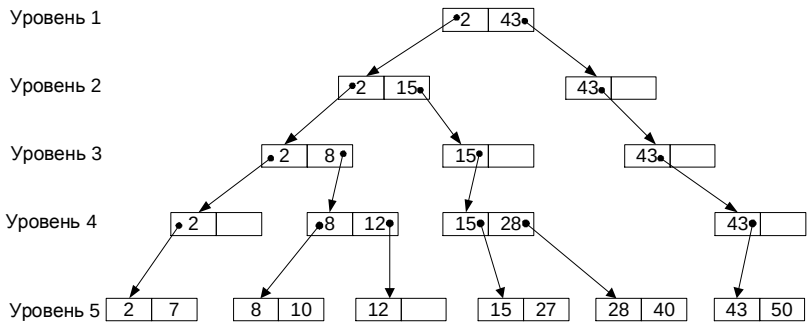


Рис. 39. B-дерево после добавления элемента

Необходимо заметить, что используемый прием деления полностью заполненного блока пополам при добавлении в него записи приведет к тому, что блоки будут заполнены, в среднем, на половину. 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 ($l \approx log_k N$), 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.

5.4.6. 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 считывается из ВП 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 при предыдущих операциях.

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 при предыдущих операциях.

Таким образом, рассмотренная структура хранения с использованием хэширования является наиболее эффективной структурой хранения (из рассмотренных выше) по критерию минимизации числа обращений к ВП при реализации основных операций.

5.4.7. Комбинированные структуры хранения

Необходимо заметить, что в СУБД могут использоваться как каждая из выше рассмотренных структур в отдельности, так и комбинация указанных структур. Так, например, в ряде промышленных систем UNIBAD, БАНК для ЭВМ типа IBM 360/370 (ЕС ЭВМ), PARADOX для персональных ЭВМ используется следующие комбинации методов:

- размещение записей по первичному ключу организовано с использованием хэширования;
- последовательность следования записей по вторичному ключу задается с помощью списковой структуры.

ГЛАВА 6. АНАЛИЗ СОВРЕМЕННОЙ ТЕХНОЛОГИИ РЕАЛИЗАЦИИ БАЗ ДАННЫХ. ЯЗЫКИ И СТАНДАРТЫ

6.1. Структура современной СУБД на примере Microsoft SQL Server

Для лучшего понимания принципов работы современных СУБД рассмотрим структуру одной из современных СУБД, а именно Microsoft SQL Server. 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. Итак, приступим к краткому обзору Microsoft SQL Server. За основу для настоящего рассмотрения возьмем Microsoft SQL Server 2000.

Краткий обзор возможностей Microsoft SQL Server был приведен в разделе, посвященном обзору современных СУБД. В данном разделе рассмотрим основные моменты, связанные с архитектурой соответствующей базы данных.

6.1.1. Архитектура базы данных

Что понимается нами под архитектурой базы данных? В предыдущих главах достаточно подробно были рассмотрены вопросы проектирования и функционирования реляционных баз данных. Microsoft SQL Server – реляционная СУБД. В данном случае под архитектурой базы данных мы будем понимать ее основные составляющие и принципы их взаимодействия (имея в виду, что речь идет именно о реляционной базе данных). При этом архитектуру можно рассматривать на двух уровнях абстракции:

- логический уровень (логическая архитектура базы данных) – данный уровень рассмотрения подразумевает изучение базы данных на уровне ее содержательных объектов. Здесь каждая СУБД имеет некоторые отличия, но они являются не очень значительными. Однако полезно знать, что у разных СУБД существенно отличаются механизмы перехода от логического к физическому уровню представления.
- физический уровень (физическая архитектура базы данных). Понятно, что любая, сколь угодно сложная база данных на самом деле представляет собой набор файлов, хранящихся на жестком

диске одного или нескольких компьютеров. 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.

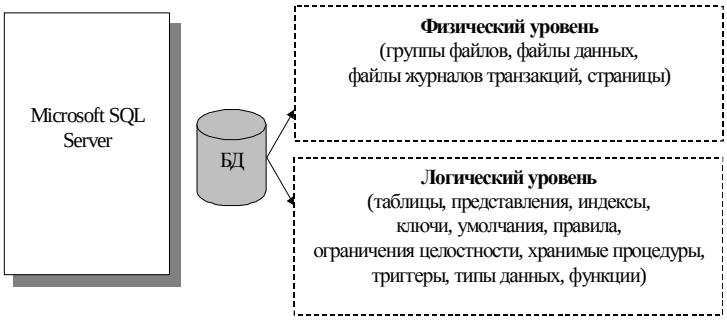


Рис. 40. Архитектура базы данных в Microsoft SQL Server 2000

Логический уровень

Рассмотрим, что собой представляет логический уровень представления базы данных.

Запустим специальную утилиту Microsoft SQL Server Enterprise Manager, выберем одну из демонстрационных баз данных и посмотрим, что с точки зрения данной утилиты (а следовательно, с точки зрения СУБД) является ее основными компонентами (рис. 41).

Приведем краткие характеристики этих компонент основных типов объектов базы данных с целью осознания ее типовой структуры.

Tables (Таблицы). Таблицы базы данных, предназначены собственно для хранения данных. Подразделяются на 2 категории: User (пользовательские) и System (системные). Пользовательские таблицы хранят полезные данные из предметной области. Системные таблицы – различную служебную информацию.

Views (Представления). По сути своей являются «виртуальными таблицами». С точки зрения пользователя, представление есть почти то же самое, что и таблица. На самом деле представление формируется на основе SQL-запроса SELECT, формируемого по обычным правилам. Т.о. представление есть поименованный запрос SELECT.

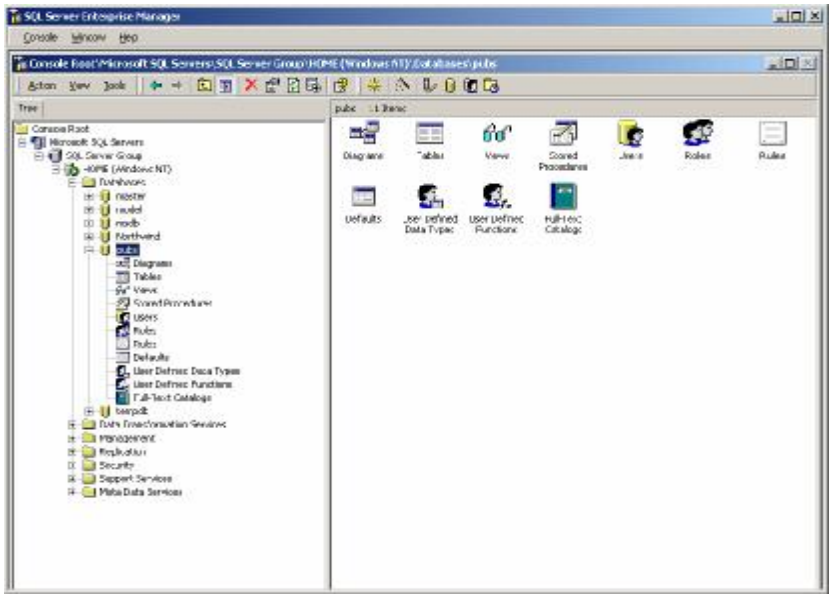


Рис. 41. База данных в Microsoft SQL Server 2000.
SQL Server Enterprise Manager

Indexes (Индексы). Индексы существуют для поддержания вместе с данными информации об их упорядоченности по различным критериям. Наличие информации об упорядоченности позволяет существенно повысить производительность некоторых операций, в частности, поиска данных. Индексы существуют непосредственно вместе с таблицами и не имеют смысла сами по себе. Индексирование может быть выполнено по одному или нескольким столбцам. Индексирование может быть произведено в любой момент.

Diagrams (Диаграммы). Диаграммы представляют собой специальные визуальные средства изучения и описания структуры базы данных. Так, при помощи диаграмм Вы можете изучать структуру таблиц и связи между ними, а также вносить в схему БД изменения.

Keys (Ключи). Так же, как и индексы, ключи не существуют сами по себе. Ключ – фундаментальное понятие в области баз данных. Являются одним из типов ограничений целостности.

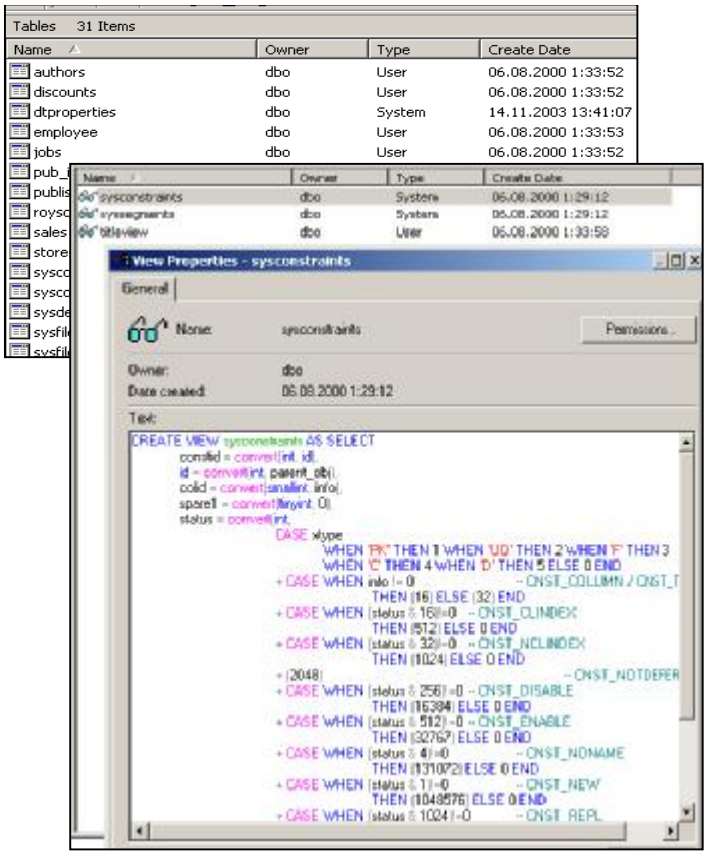


Рис. 42. Некоторые характеристики Tables и Views

Defaults (Умолчания). Умолчания не существуют отдельно от таблиц. Умолчания определяют, какие значения будут подставлены в поле данных при добавлении строки в таблицу в том случае, если значение не задано явно.

Rules (правила). Правила – специальный механизм, предназначенный для установления ограничений на диапазон возможных значений поля таблицы или нового типа данных, определяемого пользователем.

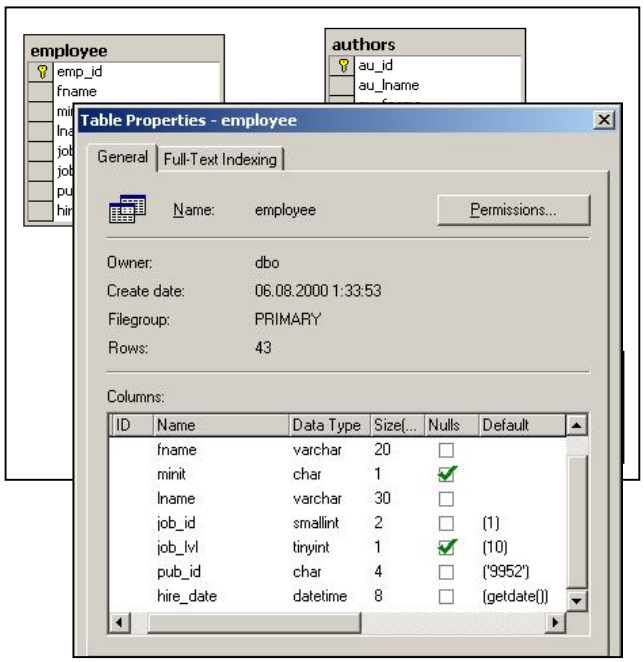


Рис. 43. Характеристика полей таблицы

Constraints (Ограничения целостности). Ограничения целостности, так же, как и большинство объектов базы данных не имеют смысла отдельно от таблиц. Ограничения целостности определяют диапазон возможных значений полей таблицы.

Stored Procedures (Хранимые процедуры). Хранимая процедура – набор команд SQL, сохраненных специальным способом в базе данных. Хранимые процедуры располагаются на сервере вместе с базой данных и могут запускаться различными пользователями, имеющими соответствующие права. При этом выполняется не один SQL-запрос, а все, что реализованы в теле процедуры.

Triggers (триггеры). Триггеры не существуют отдельно от таблицы. Триггер – специальный вид хранимой процедуры, предназначенный для производства некоторых специальных действий при выполнении операций вставки, удаления, редактирования данных.

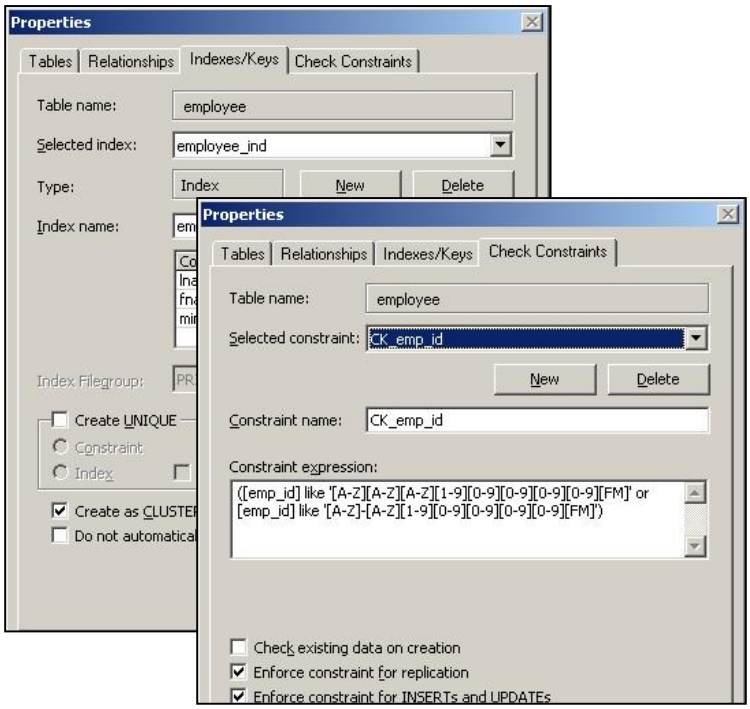


Рис. 44. Ограничения целостности

User-defined data types, UDDT (определяемые пользователем типы данных). Предоставляет специальный аппарат для создания пользовательских типов данных.

User-defined functions, UDF (определяемые пользователем функции). Функция, определяемая пользователем, представляет собой набор команд SQL, сохраненных в специальном виде. Наряду с пользовательскими функциями, существует некоторый набор стандартных функций Microsoft SQL Server, которыми можно пользоваться в работе с базой данных.

Users (пользователи). Определяет список пользователей базы данных. Служит для работы механизма защиты данных.

Roles (Роли). Роли пользователей – важный механизм для организации разграничения доступа. Для базы данных могут быть определены различные роли (например, администратор). Далее эти роли могут быть назначены пользователям.

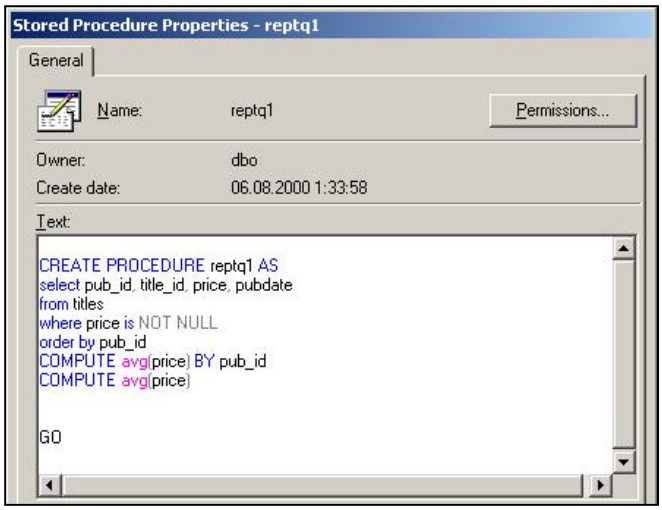


Рис. 45. Хранимые процедуры

Физический уровень

Файлы и группы файлов

Рассмотрим, что собой представляет физический уровень представления базы данных [22].

На физическом уровне база данных в Microsoft SQL Server 2000 представляет собой набор файлов с данными и набор файлов, содержащий журнал транзакций. Каждая база данных состоит из собственного уникального набора файлов, что существенно повышает надежность системы (в случае случайной порчи файлов одной из баз, например, вследствие частичной поломки жесткого диска, с другой ничего не произойдет).

Рассмотрим физическую структуру на уровне ее основных абстракций. Основными элементами структуры являются файлы и группы файлов. Все файлы делятся на 2 типа: файлы данных и файлы журнала транзакций.

- **Файлы данных** содержат собственно данные и различные объекты, входящие в логическую структуру базы данных, такие как триггеры, хранимые процедуры и т.д.
- **Файлы журнала транзакций** содержат сведения о ходе выполнения транзакций. В них содержится информация о том, когда началась и закончилась транзакция, какие ресурсы, кем и когда были заблокированы или разблокированы и т.д.

Каждая база данных содержит как минимум 2 файла: файл с данными и файл журнала транзакций. При необходимости имеется возможность создания новых файлов с данными и новых фалов для журнала транзакций. При этом можно указать, какие данные в каком файле хранить. Совершенно очевидно, что за всем этим многофайловым механизмом кроется определенный смысл. Он заключается в возможности оптимального распределения данных по файлам с целью повышения эффективности их обработки. Поиск подобного оптимального распределения – сложная практическая задача. Для ее однозначного решения трудно предложить какие-то общие рецепты, в каждом конкретном случае необходимо действовать по-разному. В принципе, для решения проблемы можно применить 2 подхода. Один из возможных подходов состоит в попытке построения теоретических оценок того, какое размещение является наилучшим в некотором смысле. На практике применение этого подхода сопряжено с большими трудностями. Второй подход заключается в проведении испытаний с различным распределением данных по файлам с целью выявления наилучшего распределения. Этот подход также трудно претворить в жизнь, из-за того, что на этапе разработки невозможно успешно смоделировать реальную ситуацию, в которой будет функционировать база данных. В итоге, на практике применяются некоторые элементы обоих подходов. Так, например, очевидно, что при наличии нескольких жестких дисков разумно создать по крайней мере по одному файлу на каждом из них, разместив в них данные, которые обычно могут обрабатываться параллельно. Благоприятно скажется на производительности и размещение на разных носителях файлов с данными и журнала транзакций.

Файлы с данными бывают 2-х типов [22]:

- **Основной или главный файл** (**Primary File**) – присутствует в любой базе данных. Если в базе данных всего один файл с данными, то он является главным. Главный файл содержит так называемые метаданные – данные о том, как устроена база данных (системные таблицы…). Кроме системной информации, главный файл может хранить и пользовательские данные. Главный файл имеет расширение **mdf** (Master Data File).
- **Вторичный или дополнительный файл** (**Secondary File**) – дополнительный файл для хранения пользовательских данных. Не содержит системной информации. Такие файлы могут создаваться администратором по мере необходимости. Имеет расширение **ndf** (secoNdary Data File).

Файлы журнала транзакций бывают только одного типа – **файлы журнала транзакций** (Transaction Log File). Эти файлы содержат информацию о транзакциях. Имеют расширение **ldf** (Log Data File).

При необходимости администратор может поменять расширения файлов.

Для удобства администрирования и с целью повышения эффективности обработки файлы с данными могут объединяться в группы. По умолчанию существует одна группа Primary File Group – основная группа файлов и все файлы добавляются в нее. При необходимости возможно создание пользовательских групп файлов (User File Group). Существует также группа файлов по умолчанию (Default File Group), содержащая объекты базы данных, не приписанные явно ни к одной группе. Такая группа бывает только одна.

Microsoft SQL Server 2000 применяет специальные оптимизационные механизмы для улучшения ситуации с обработкой данных с применением групп файлов. Так, СУБД стремится к равномерному распределению данных по файлам в рамках одной группы. Умелое использование механизмы групп позволяет существенно повысить быстродействие.

Страницы и группы страниц

Для работы с файлами в Microsoft SQL Server активно применяется принцип постраничной организации файлов. В рамках этого принципа файл рассматривается как набор последовательно расположенных страниц с данными. Таким образом, страница является минимальным блоком (квантом) данных, с которым может идти работа (физической

записью). Заметим, что размер страницы в Microsoft SQL Server составляет 8Кб.

Нетрудно понять, в чем заключается смысл постраничной организации. Этот способ представления позволяет использовать разные алгоритмы буферизации и существенно повысить эффективность работы с файлами. Однако постраничная организация в применяется в SQL Server лишь для файлов с данными, т.к. именно они имеют очень сложную структуру и на работу с ними ложится основная нагрузка. Файлы журнала транзакций, напротив, устроены достаточно просто, т.к. содержат последовательно расположенные записи о происходящих транзакциях; здесь допустима работа с отдельными записями.

Страницы файлов данных могут использоваться не только для хранения собственно данных, но и для представления различной служебной информации. Так, страницы могут быть одного из следующих типов [22]:

- **Data**. Предназначены для хранения данных таблиц, кроме т.н. «тяжелых» данных (Image, Text, nText).
- **Index.** Предназначены для хранения индексов таблиц и представлений.
- **Text/Image**. Используются для хранения «тяжелых» данных Image, Text, nText, занимающих обычно более одной страницы. Для таких данных место сразу же выделяется постранично.
- **Global Allocation Map (GAM)**. Служебный тип страницы. Содержит информацию об использовании групп страниц.
- **Page Free Space (PFS)**. Содержит сведения о расположении свободного пространства в страницах файла. Своеобразная замена «списка свободных элементов».
- **Index Allocation Map (IAM)**. Содержат информацию о группах страниц, которые используются таблицами.

Каждая страница имеет некоторую внутреннюю архитектуру. Так, у любой страницы присутствует заголовок, содержащий служебную информацию. Структура заголовка одинакова для страниц разных типов.

Подобная страничная организация была бы не слишком удобной для хранения огромных объемов данных, а ведь современные серверные СУБД и, в частности, Microsoft SQL Server часто используются именно в таких целях. Для дополнительной оптимизации в SQL Server было введено понятие «группа страниц»

(экстент). Экстент содержит 8 страниц и, следовательно, имеет размер 64Кб. При необходимости выделить новую страницу создается сразу целый экстент. Наряду с различиями в типах страниц, существуют аналогичные различия и в типах экстентов. Детальное описание структуры и типов экстентов можно найти, например в [22]. Мы же в рамках данного пособия ограничимся изложением общих принципов физической организации и перейдем к описанию организации БД на логическом уровне.

6.2. Программное окружение БД. Проблемы доступа и обработки данных

6.2.1. Проблемы доступа и обработки данных

В этом разделе мы перейдем к рассмотрению реляционных баз данных, функционирующих в рамках архитектуры «Клиент-сервер». Краткая характеристика подобных баз данных была изложена в п. 1.5.. В данном разделе описаны основные подходы, применяемые для решения проблемы организации данных и последующего доступа к ним.

Проблемы организации данных и доступа к ним имеют огромное значение. Существует ряд вопросов, которые неизбежно возникают у всех разработчиков баз данных на протяжении всего цикла создания продукта:

- Как создать таблицы (описать схему базы данных)?
- Как описать ограничения целостности?
- Как определить значения по умолчанию для полей?
- Что делать с неопределенными значениями (NULL)?
- Как добавить записи в таблицу?
- Как обновить записи?
- Как удалить записи?
- Как осуществить выборку нужных записей из базы данных?
- Как удалить часть записей?
- Как проконтролировать непротиворечивость данных после операций редактирования?
- Как осуществлять сортировки?
- Как осуществлять разграничение доступа?

Понятно, что перечень этих вопросов не полон, но уже его достаточно для того, чтобы понять тот факт, что создание и обработка

базы данных – сложная задача, требующая применения специальных технологий. Проанализировав список вопросов, можно прийти к выводу, что все они делятся на три категории:

- вопросы создания базы данных (создание таблиц, индексов, ограничений целостности);
- вопросы обеспечения безопасности и разграничения доступа;
- вопросы обработки данных (выборка, редактирование, удаление, добавление).

6.2.2. Навигационный подход

Первая категория вопросов может быть решена посредством создания в каждой конкретной СУБД некоторой утилиты, позволяющей пользователю осуществлять все необходимые действия. Однако что делать, если необходимо создать таблицу динамически во время работы программы? Как объяснить удаленному серверу, что ему нужно добавить в таблицу столбец?

Вторая категория вопросов может быть решена тем же способом, но остаются те же проблемы.

А что делать с третьей категорией вопросов? Очевидно, эти вопросы возникают постоянно при эксплуатации информационной системы. Пользователи постоянно что-то делают с данными. Рассмотрим простой пример. Пусть у нас есть таблица «Абитуриент», хранящая информацию следующего рода:
АБИТУРИЕНТ (Код абитуриента, Фамилия, Имя, Отчество, Номер аттестата, Дата выдачи аттестата).

Теперь мы хотим выполнить некоторый запрос к базе данных, результатом которого должны стать те строки таблицы «АБИТУРИЕНТ», для которых «Дата выдачи аттестата» окажется больше 01.06.2003. Как алгоритмически организовать выполнение подобного запроса? Видимо, необходимо осуществить следующее:

1. Получаем доступ к таблице «АБИТУРИЕНТ» и устанавливаем указатель текущей строки на первую строку таблицы.
2. Анализируем поле «Дата выдачи аттестата» в текущей строке.
3. Если значение «Дата выдачи аттестата» > «01.06.2003», 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 выдачи аттестата»).Value >
       «01.06.2003»
    then List.Add(FieldByName(«Фамилия»).AsString);
    Table.Next;
end;
```

Такой подход к обработке данных, ориентированный на последовательную работу с отдельными записями, называется **навигационным**.

Теперь представьте себе, что произойдет с текстом программы, если, например, нужно в базе данных найти всех абитуриентов из Нижегородской области, у которых средний бал по математическим дисциплинам за 3 семестра превысил 4 и упорядочить их по убыванию среднего балла. Очевидно, текст программы возрастет в объеме на порядок.

Теперь представьте, что необходимо предоставить пользователю базы данных интерфейс для осуществления подобных запросов. Можно ли заранее предугадать все запросы, потребность в которых может возникнуть и заранее запрограммировать их? Конечно, нет. Учитывая, что подавляющее большинство пользователей не владеет навыками программирования, это означает, что объем того, что они могут делать, будет ограничен рамками написанной программы, а именно, теми запросами, реализация которых предусмотрена при написании этой программы.

Но и это еще не все. Представьте себе, как будет функционировать весь этот механизм в рамках архитектуры «Клиент-Сервер». Каким образом клиентское приложение будет объяснять серверной части, что именно необходимо сделать с данными?

6.2.3. Подход, основанный на использовании интерпретируемых языков запросов

Для решения всех этих и некоторых других проблем в настоящее время используется альтернатива навигационному подходу, использующая специальный интерпретируемый язык запросов **SQL**.

Язык SQL (Structured Query Language – структурированный язык запросов) применяется для общения пользователя с реляционной базой данных и состоит из трех частей [6]:

- DDL (Data Definition Language) – язык определения данных. Предназначен для создания базы данных (создания 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.

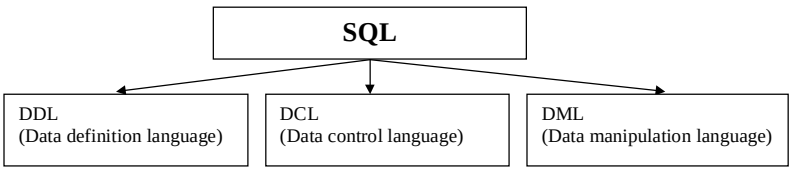


Рис. 46. Составные части SQL

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 абитуриентов по дате выдачи аттестата.

```
SELECT Фамилия
FROM  Абитуриенты
WHERE Дата выдачи аттестата > «01.06.2003»
```

У того, кто читает эти строки, может сложиться ложное впечатление о том, что мы хотим принизить значение языков программирования высокого уровня, выставляя им некоторую альтернативу – язык SQL. Это не соответствует действительности. Понятно, что выполнение запроса все равно сводится к работе с отдельными записями и от этого никуда не уйти. Однако, важно понимать, что появление языка SQL привело к появлению некоторого нового уровня абстракции между пользователем и СУБД. Этот уровень находится ближе к пользователю, чем уровень программирования на языке программирования высокого уровня, что снижает требования на квалификацию пользователей. Второй существенный момент заключается в том, что многие вопросы, которые раньше все программистское сообщество многократно решало в силу своего понимания, зачастую дублируя действия друг друга, теперь решены в серверных СУБД, которая, к примеру, умеет самостоятельно выбирать лучший с точки зрения быстродействия план выполнения запроса. Т.о. отпала необходимость самостоятельного решения многих проблем, они уже решены в СУБД, а стандарт SQL предоставляет средства для доступа к возможностям СУБД.

6.3. Понятие языка SQL и его основные части

6.3.1. История возникновения и стандарты языка SQL

История возникновения языка SQL восходит к 1970 году[8], когда доктор 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 и произношение названия остались неизменными (аббревиатура SQL иногда читается «Эс-Кю-Эль», а иногда «Сиквел»).

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-го века. Именно наличие разработанных и официально утвержденных стандартов позволило утвердиться многим современным технологиям (не только в индустрии разработки ПО, но и во 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 2 организации:

- **ANSI** (American National Standards Institute) – Американский национальный институт стандартов;
- **ISO** (International Standards Organization) – Международная организация 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) были созданы, представлены на рынке и активно использовалось несколько различных СУБД, в которых в том или ином виде был реализован некоторый диалект языка SQL. Принимая во внимание тот факт, 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 заполнил многие «белые пятна», впервые добавив в стандарт возможности, еще не реализованные в существующих коммерческих СУБД.

Работа над стандартизацией продолжается и сейчас. Конечно, SQL-92 не решил всех проблем, связанных с наличием нескольких диалектов языка. Одно только описание стандарта разрослось от ста страниц (SQL-89) до 600 страниц (SQL-92). Кроме того, все 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. Однако, не все так плохо, как может показаться. Несмотря на имеющиеся отличия, все коммерческие СУБД поддерживают некоторое ядро языка, описанное в стандарте SQL-92, одинаково. Отличий не очень много, они не носят уж слишком принципиального характера. Хотя, каждая СУБД по-прежнему поддерживает свой диалект языка. Из этих диалектов есть некоторые, более уважаемые многими разработчиками, чем непосредственно стандарт, например диалект компании IBM, родоначальника SQL. Компания IBM, выпуская СУБД DB2 и Informix, по-прежнему удерживает существенные позиции на рынке серверных СУБД.

6.3.2. Достоинства языка SQL

Для ознакомления с достоинствами языка обратимся к соответствующей литературе [8]. 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 Интернет;
- интеграция с языком Java (протокол JDBC);
- промышленная инфраструктура [8].

6.3.3. Разновидности SQL

Стандарты языка SQL регламентируют синтаксис операторов. Если посмотреть на операторы языка, становится понятно, что в отличие от «обычных» языков программирования в SQL отсутствует возможность объявления переменных, нет инструкции IF, нет цикла FOR и т.д. Одним словом, в таком виде язык годился исключительно для интерактивного режима работы с базой данных: пользователь вводит запрос – получает результат, вводит другой запрос – получает другой результат и т.д. Программирование на подобном языке представлялось крайне затруднительным. Естественно, дело этим не кончилось. Технологии продолжают двигаться вперед, и на настоящий момент известны следующие разновидности языка SQL:

- интерактивный SQL;
- программный (встроенный) SQL:
  - статический SQL;
  - динамический SQL;
  - API – интерфейсы вызова подпрограмм.

Разберем эти разновидности подробнее.

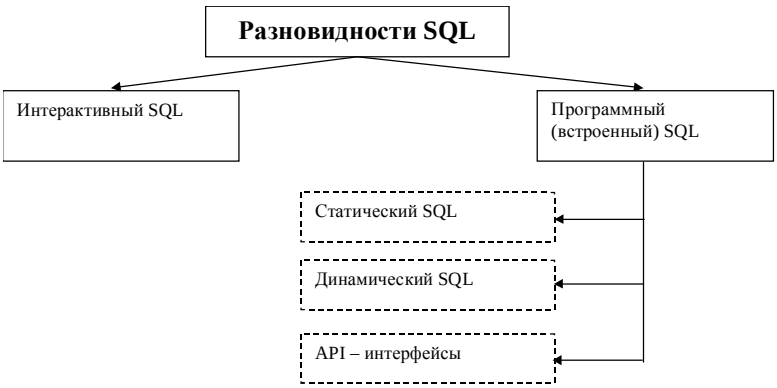


Рис. 47. Разновидности SQL

Интерактивный SQL будет рассмотрен в данном учебнике подробнее, чем программный. Для всех разновидностей SQL будут приведены основные идеи и рассмотрены ключевые концепции. Детальное рассмотрение статического, динамического SQL и различных API-интерфейсов (ODBC, JDBC, DB Library и др.) выходит за рамки нашего курса. Подробное изучение этих тем выносится на специальные курсы, посвященные данной проблематике.

6.4. Понятие интерактивного SQL. Элементы интерактивного SQL. Использование SQL для манипулирования данными

Итак, **интерактивный SQL** 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 текст SQL-запроса;
- запускает запрос на выполнение.

Текст запроса поступает в СУБД, которая:

- осуществляет синтаксический анализ запроса (проверяет, является ли запрос корректным);

- проверяет, имеет ли пользователь право выполнять подобный запрос (может быть пользователь пытается что-то удалить, а права есть только на чтение);
- выбирает, каким образом осуществлять выполнение запроса – план выполнения запроса;
- выполняет запрос;
- результат выполнения отсылает пользователю.

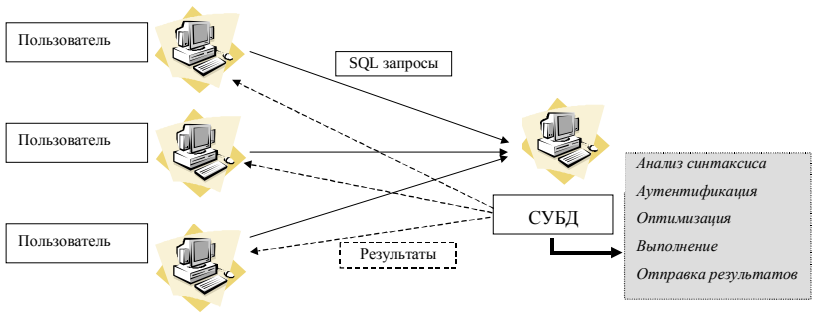


Рис. 48. Схема работы интерактивного SQL

В данном разделе кратко рассмотрим ту часть языка SQL, которая предназначена для манипулирования данными, находящимися в таблицах реляционной базы данных. Соответствующую подсистему SQL обычно называют DML (Data Manipulation Language) – язык манипулирования данными. Как было указано в предыдущих разделах, DML содержит операторы для внесения изменений в содержимое таблиц базы данных.

В ходе рассмотрения будут рассмотрены примеры SQL-запросов. Эти примеры предназначены для лучшего понимания изложения идеологии языка и синтаксиса операторов. Примеры будут основаны на базе данных, приведенной выше по тексту в качестве примера предметной области.

6.4.1. Использование SQL для выбора информации из таблицы

Простейший запрос

Вообще говоря, под запросом можно понимать команду, посылаемую пользователем системе управления базами данных для выполнения того или иного действия. С точки зрения русского языка это определение несколько отличается от того смысла, который обычные люди, как правило, вкладывают в слово «запрос». Так, позже мы увидим, что далеко не все запросы связаны именно с извлечением какой-то информации из таблиц, однако, в данном разделе рассматриваются именно такие запросы.

Простейший запрос – команда СУБД, которая указывает ей, что она должна выбрать определенные данные из одной конкретной таблицы. Для подобных запросов (и не только для них, но и для значительно более сложных) предназначена команда SELECT.

Команда Select

Простейший запрос на выборку может выглядеть следующим образом:

```
SELECT  *
FROM    entrant
```

В результате этого запроса будут выбраны все данные об абитуриентах, содержащиеся в таблице entrant.

Аналогичного результата можно достигнуть, явным образом перечислив через запятую названия всех столбцов таблицы после ключевого слова SELECT. Вот как это могло бы выглядеть:

```
SELECT  id, name, second_name,
        surname, birth_date, birth_place,
        distinction_id, citizen_id, education_id,
        sex, rem, original, mod_date, mod_user
FROM    entrant
```

Естественно, перечисление всех названий столбов не имеет особого смысла и приносит только дополнительные неудобства по сравнению с использованием символа *. Кроме того, первый способ сработает правильно и после добавления новых столбцов в таблицу, чего не скажешь о втором.

Легко догадаться, что нужно сделать, если не все столбцы таблицы нужны в качестве результатов запроса:

```
SELECT  name, second_name, surname
FROM    entrant
```

Данный запрос приведет к включению в результат только 3-х столбцов таблицы entrant.

А что делать, если Вам нужно переупорядочить столбцы в результате запроса? Решение следующее – необходимо поменять порядок их следования в первой части команды SELECT.

```
SELECT  second_name, name, surname
FROM    entrant
```

Рассмотрим еще один момент, связанный с простейшей формой оператора SELECT. Иногда в качестве результатов запроса вы получаете набор строк, некоторые из которых дублируются. Так, например, вы хотели бы получить список стран, в которых родились Ваши абитуриенты. Как было показано выше, для этого достаточно применить следующий запрос:

```
SELECT  birth_place
FROM    entrant
```

Однако, есть одна проблема: в качестве результата Вы скорее всего получите список, содержащий одинаковые значения, например:

```
Нижний Новгород
Москва
Нижний Новгород
Нижний Новгород
Дзержинск
Нижний Новгород
Нижний Новгород
Козино
Нижний Новгород
Нижний Новгород
Нижний Новгород
Лукино
```

Обычно это не то, что требуется. Для получения нужного результата следует удалить из этого списка повторяющиеся города. Этот результат достигается при помощи ключевого слова DISTINCT:

```
SELECT DISTINCT birth_place
```

```
FROM    entrant
```

В результате выполнения запроса Вы получите:

```
Нижний Новгород
Москва
Дзержинск
Козино
Лукино
```

Противоположность DISTINCT – ключевое слово ALL. Во всех предыдущих запросах мы ничего не указывали после SELECT и ALL подразумевалось по умолчанию.

Квалифицированный выбор – предложение WHERE. Начало использования

Используя перечисление имен столбцов непосредственно после ключевого слова SELECT, Вы могли ограничить результат запроса лишь несколькими необходимыми столбцами. Аналогичные действия необходимо производить и для строк. Так, очень часто Вам требуется выбрать из таблицы данные, удовлетворяющие определенным условиям. Для реализации этой возможности в языке SQL существует ключевое слово WHERE. Ключевое слово WHERE пишется в команде SELECT после окончания секции FROM и содержит после себя логическое выражение, которое проверяется последовательно для каждой строки рассматриваемой таблицы.

Рассмотрим несколько несложных примеров запросов:

```
SELECT  name, second_name, surname
FROM    entrant
WHERE   birth_place = 'Нижний Новгород'
```

Этот запрос распечатает имена, фамилии все абитуриентов, родившихся в Нижнем Новгороде.

```
SELECT  name, second_name, surname
FROM    entrant
WHERE   birth_place <> 'Нижний Новгород'
```

Этот запрос распечатает имена, фамилии все абитуриентов, родившихся не в Нижнем Новгороде.

```
SELECT  name, second_name, surname
```

```
FROM    entrant
WHERE   birth_date > ‘01.01.1984’
```

Этот запрос распечатает имена, фамилии все абитуриентов, родившихся после 1 января 1984 года.

```
SELECT  name, second_name, surname
FROM    entrant
WHERE   birth_date <= ‘01.01.1986’
```

Этот запрос распечатает имена, фамилии все абитуриентов, родившихся до 2 января 1986 года.

Квалифицированный выбор – предложение WHERE. Использование реляционных и булевых операторов

Нетрудно видеть, что современные языки программирования высокого уровня предоставляют гораздо более мощные средства для составления условий, чем указано выше. Не остался в стороне и SQL. Так, SQL допускает формирование сложных составных условий с использованием реляционных и булевых операторов.

Под реляционными операторами в данном случае понимаются:

> – больше;
< – меньше;
>= – больше или равно;
<= – меньше или равно;
= – равно;
<> – не равно.

Примеры использования реляционных операторов мы рассмотрели выше.

Под булевыми операторами в данном случае понимают следующие:

AND – и;
OR – или;
NOT – не.

При помощи булевых операторов Вы можете создавать составные условия, комбинируя различные ограничения. Рассмотрим примеры соответствующих запросов:

```
SELECT  name, second_name, surname
FROM    entrant
WHERE   (birth_place = ‘Нижний Новгород’) AND
```

```
        (birth_date <= '01.01.1986')
```

Этот запрос выведет в качестве результата данные всех абитуриентов, которые родились в Нижнем Новгороде ранее 2 января 1986 года.

```
SELECT DISTINCT entrant_id
FROM     mark
WHERE   (mark >= 2)
    AND (mark <= 3)
```

Этот запрос выведет в качестве результата идентификаторы всех абитуриентов, которые получили на экзаменах хотя бы одну двойку или тройку.

```
SELECT  name, second_name, surname
FROM    entrant
WHERE   NOT (birth_place = 'Нижний Новгород')
```

Этот запрос выведет в качестве результата данные всех абитуриентов, которые родились не в Нижнем Новгороде.

Квалифицированный выбор – предложение WHERE. Использование специальных операторов

Наряду с рассмотренными операторами для составления условий, мало чем отличающихся от таких же операторов, существующих в языках программирования высокого уровня, 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 значений.

Оператор IN используется для проверки вхождения в некоторое множество значений. Так, запрос

```
SELECT  DISTINCT entrant_id
FROM    mark
WHERE   mark IN (2,3)
```

Выведет всех абитуриентов, получивших хотя бы одну двойку или тройку на экзаменах.

Того же результата можно добиться, используя оператор BETWEEN:

```
SELECT  DISTINCT entrant_id
FROM    mark
WHERE   mark BETWEEN 2 AND 3
```

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.

Так, например, следующий запрос покажет данные всех абитуриентов, фамилии которых начинаются с буквы «А».

```
SELECT  name, second_name, surname
FROM    entrant
WHERE   name LIKE 'A%'
```

Оператор IS NULL позволяет обнаруживать в таблицы неопределенные значения (напомним, что подобные значения называются NULL).

Так, например, следующий запрос покажет всех абитуриентов, которые не указали место своего рождения:

```
SELECT  name, second_name, surname
FROM    entrant
WHERE   birth_place IS NULL
```

Использование агрегатных функций. Простые запросы

Очень часто возникает необходимость произвести вычисление минимальных, максимальных или средних значений в столбцах. Так, например, Вам может понадобиться вычислить средний балл или что-то другое. Для осуществления подобных вычислений в языке программирования высокого уровня Вы бы предприняли некоторые действия, в частности организовали бы цикл, сосчитали сумму, потом разделили бы ее на количество слагаемых в сумме, получив значение

среднего балла. SQL избавляет Вас от необходимости программирования всей последовательности подобных действий. Для этого SQL предоставляет Вам 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.

Так, следующие запросы считают, соответственно, минимум, максимум, сумму, среднее среди всех баллов, полученными студентами на экзаменах. Количество оценок считает последний запрос.

```
SELECT  MIN(mark)
FROM    mark
SELECT  MAX(mark)
FROM    mark
SELECT  SUM(mark)
FROM    mark
SELECT  AVG(mark)
FROM    mark
SELECT  COUNT(mark)
FROM    mark
```

Заметим, что, изменив следующим образом текст последнего запроса, мы получим количество разных оценок, полученных на экзаменах:

```
SELECT  COUNT(DISTINCT mark)
FROM    mark
```

Использование агрегатных функций. Комбинирование с предложением WHERE

Дополнительно к описанным выше простым случаям, Вы можете использовать агрегатные функции совместно с предложением WHERE:

```
SELECT  AVG(mark)
FROM    mark
WHERE   entrant_id = 100
```

Данный запрос вычислит средний балл студента с кодом 100 по результатам всех сданных им экзаменов.

```
SELECT  AVG(mark)
FROM    mark
WHERE   exam_id = 10
```

Данный запрос вычислит средний балл студентов по результатам сдачи экзамена с кодом 10.

Нетрудно видеть, что агрегатные функции, в сочетании с ограничением области данных при помощи предложения WHERE, представляют собой мощный аппарат, позволяющий ответить на многие реально возникающие вопросы путем применения SQL-запросов.

Использование агрегатных функций и группировка. Предложение GROUP BY

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     entrant_id, AVG(mark)
FROM       mark
GROUP BY   entrant_id
```

Все это, как обычно, может быть совмещено с предложением WHERE. При этом, не вдаваясь в тонкости выполнения запроса внутри СУБД, Вы можете 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. Для этого достаточно выполнить следующий запрос:

```
SELECT     entrant_id, AVG(mark)
FROM       mark
WHERE      exam_id = 100
GROUP BY   entrant_id
```

Заметим, что группировка может производиться более, чем по одному полю. Так, вы можете вычислить средний балл, введя дополнительное разграничение по дням, в которые вносилась информация в базу (считая, например, что эта дата совпадает с датой проведения экзамена):

```
SELECT     entrant_id, mod_date, AVG(mark)
FROM       mark
WHERE      exam_id = 100
GROUP BY   entrant_id, mod_date
```

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.

Использование агрегатных функций и группировка. Ограничение результатов запроса. Предложение HAVING

В языке SQL присутствуют средства, которые позволяют Вам сделать следующее: допустим, Вы хотите получить в качестве результатов запроса данные только тех абитуриентов, средний балл которых превышает 4. Как это сделать? Проблема в том, что стандарт языка не допускает использование агрегатных функций в предложении WHERE. Получается, что сначала необходимо выполнить запрос и сосчитать среднее, а потом ограничить результат по условию «Среднее > 4». Для осуществления подобных действий стандартом языка предусмотрена специальная секция HAVING. Так, запрос может выглядеть следующим образом:

```
SELECT     entrant_id, mod_date, AVG(mark)
FROM       mark
WHERE      exam_id = 100
GROUP BY   entrant_id, mod_date
HAVING     AVG(mark) > 4
```

Форматирование вывода. Выражения в запросе. Упорядочение

Для того, чтобы форматировать вывод, Вы можете использовать различные возможности SQL. Так, например, допустимым является включение текста в запрос. Рассмотрим пример того, как это делается:

```
SELECT  'Средний балл=', AVG(mark)
FROM    mark
WHERE   exam_id = 10
```

В результате данного запроса Вы увидите не просто некоторое число, а число, сопровожденное поясняющим текстом.

Наряду с обычной выборкой значений из столбцов таблицы, имеется возможность вносить изменения в эти значения путем вычисления выражений.

Так, например, Вы можете перейти от 5-балльной к 10-балльной системе, преобразовав значение среднего балла следующим образом:

```
SELECT  'Средний балл=', AVG(mark)*2
FROM    mark
WHERE   exam_id = 10
```

Следующий важный момент связан с осуществлением сортировки (упорядочения) данных. Для организации сортировок в стандарте SQL предусмотрена специальная секция ORDER BY. В этой секции Вам необходимо указать, по какому столбцу (по каким столбцам) упорядочивать результаты запроса. При этом, при использовании нескольких столбцов упорядочение будет производится последовательно (по первому критерию, если значения совпали – по второму критерию и т.д.).

Следующий запрос распечатывает список абитуриентов, в котором фамилии следуют в алфавитном порядке.

```
SELECT   name, second_name, surname
FROM     entrant
ORDER BY name
```

По умолчанию подразумевается, что сортировка производится по возрастанию. Если Вам необходимо упорядочивать по убыванию, необходимо использовать указание «DESC».

```
SELECT   name, second_name, surname
FROM     entrant
```

```
ORDER BY name DESC
```

Для упорядочения по возрастанию присутствует указание «ASC», но его обычно не пишут явным образом.

При упорядочении по нескольким столбцам Вы можете упорядочивать как по возрастанию, так и по убыванию, указывая соответствующую конструкцию после наименования каждого столбца.

```
SELECT   name, second_name, surname
FROM     entrant
ORDER BY name DESC, second_name ASC
```

Наряду с упорядочиванием в обычных запросах, стандарт SQL позволяет сортировать и запросы, содержащие агрегирование. В этом случае порядок следования секций следующий:

```
SELECT   entrant_id, mod_date, AVG(mark)
FROM     mark
WHERE    exam_id = 100
GROUP BY entrant_id, mod_date
HAVING   AVG(mark) > 4
ORDER BY entrant_id
```

6.4.2. Использование SQL для выбора информации из нескольких таблиц

До сих пор мы рассматривали ту часть SQL, которая касалась выбора информации из единственной таблицы. Естественно, возможности языка этим не ограничиваются. Так, Вы можете запрашивать информацию из нескольких таблиц, реализуя описанные в соответствующем разделе учебника реляционные операции. Рассмотрим некоторые примеры того, как это делается. Стоит упомянуть, что полное рассмотрение этой темы выходит за рамки данного учебника. Подробно этот вопрос можно изучить при помощи, например, [8] и [6]. Мы рассмотрим некоторые несложные случаи, часто встречающиеся на практике, в которых присутствует необходимость выбора информации из нескольких таблиц сразу.

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 ко многим или один к одному по некоторому полю.

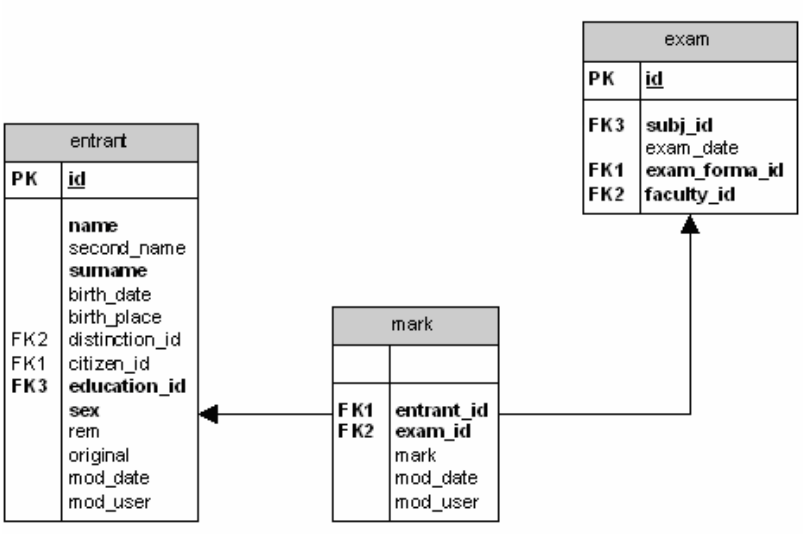


Рис. 49. Пример связанных таблиц

В примере (рис. 48) тоже присутствуют такие таблицы. Рассмотрим таблицы entrant, mark и exam.

Таблица mark (оценки) связана с таблицей exam (экзамены) по полям **id–entrant_id**.

Таблица mark (оценки) связана с таблицей entrant (абитуриенты) по полям **id– exam_id**.

Допустим нам требуется распечатать список абитуриентов с оценками, которые они получили на экзаменах. Для этого необходимо выполнить следующий запрос:

```
SELECT  entrant.name, mark.exam_id, mark.mark
FROM    entrant, mark
WHERE   entrant.id = mark.entrant_id
```

Заметим, следующие изменения по сравнению с теми запросами, которые мы писали ранее:

1. В секции FROM указаны 2 таблицы.
2. Так 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.

Предположим, нас не удовлетворяет тот факт, что мы видим код предмета, но не видим дату сдачи экзамена. Для того, чтобы решить эту проблему необходимо выполнить следующий запрос:

```
SELECT  entrant.name, exam.exam_date, mark.exam_id,
        mark.mark
FROM    entrant, mark, exam
WHERE   (entrant.id = mark.entrant_id) AND
        (exam.id = mark.exam_id)
```

Нетрудно заметить, что использование префиксов-имен таблиц сильно загромождает запрос. Для того, чтобы избежать подобного загромождения, используются псевдонимы. Так, мы можем переписать предыдущий запрос следующим образом:

```
SELECT  E.name, X.exam_date, M.exam_id, M.mark
FROM    entrant E, mark M, exam X
WHERE   (E.id = M.entrant_id) AND
        (E.id = M.exam_id)
```

6.4.3. Использование SQL для вставки, редактирования и удаления данных в таблицах

Для добавления данных в таблицу в стандарте SQL предусмотрена команда INSERT. Синтаксис команды проще всего понять из следующего запроса:

```
INSERT INTO mark
VALUES (1, 2, 5, '01.08.2003', 11)
```

Данный запрос вставляет в таблицу mark строку, содержащую значения, перечисленные в списке VALUES.

Если Вы не хотите указывать значение какого-то поля, Вы можете присвоить ему NULL:

```
INSERT INTO mark
VALUES (1, 2, 5, '01.08.2003', NULL)
```

В случае, если Вы считаете необходимым использование для некоторых полей значений по умолчанию, SQL позволяет явно указать, какие поля необходимо заполнить Вашими данными, а какие – значениями по умолчанию.

```
INSERT INTO mark (entrant_id, exam_id, mark)
VALUES (1, 2, 5)
```

Кроме явного указания значений, Вы также можете вставить в таблицу данные, представляющие собой результат выполнения другого запроса. Так, пусть у Вас есть вспомогательная таблица entrant_temp с полями id, name.

```
INSERT INTO entrant_temp (name)
        SELECT      E.name
        FROM        entrant E
        ORDER BY    E.name
```

Для удаления данных из таблицы существует команда DELETE.

```
DELETE
FROM    entrant_temp
```

Этот запрос удаляет все данные из таблицы entrant_temp.

Вы можете ограничить диапазон удаляемой информации следующим образом:

```
DELETE
FROM    entrant_temp
WHERE   name > 'И'
```

Для обновления данных в таблице существует команда UPDATE. Так, запрос

```
UPDATE  mark
SET     mark = '5'
```

производит обновление таблицы mark, меняя в ней значения всех оценок на «5».

При необходимости, Вы можете ограничить состав обновляемых записей. Например, это может быть сделано следующим образом:

```
UPDATE  mark
SET     mark = '5'
```

```
WHERE   mark = ‘4’
```

При помощи этого запроса Вы поднимете балл на 1 у тех абитуриентов, кто получил на экзамене «4».

Итак, для изменения данных, содержащихся в таблицах, предназначены команды SQL INSERT, DELETE и UPDATE.

6.4.4. 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 объединение

Средствами языка SQL операция объединения представляется следующим образом:

```
SELECT *
FROM A
UNION
SELECT *
FROM B
```

Операция разность

Средствами языка SQL операция объединения представляется следующим образом:

```
SELECT *
FROM A
EXCEPT
SELECT *
FROM B
```

Операция проекция

```
SELECT Field_i1, … , Field_in
FROM A
```

Операция выборка (селекция)

```
SELECT *
FROM A
WHERE (<condition>)
```

Операция пересечение

```
SELECT *
FROM A
INTERSECT
SELECT *
FROM B
```

Операция соединение, эквисоединение

```
SELECT A.Field1, … , A.Fieldn, B.Field1, … , B.Fieldm
FROM A, B
WHERE (A.Fieldi Θ B.Field1)
```

Если Θ – операция «=», то это эквисоединение.

Операция естественное соединение

Пусть есть отношения A (X_1, … , X_n, A_1, …, A_m) и B (X_1, … , X_n, B_1, …, B_r).

```
SELECT A.X1, … , A.Xn, A.A1, … , A.Am,  B.B1, … , B.Br
FROM A, B
WHERE (A.X1 = B.X1) AND … AND (A.Xn = B.Xn)
```

6.5. Программный (встроенный) SQL

Программный (**встроенный**) **SQL** (Embedded SQL) предназначен для того, чтобы встраивать SQL запросы в прикладную программу. Представьте себе, что Вы – программист, разрабатывающий приложение для работы с базами данных. Совершенно, очевидно, что для Вас является неприемлемой интерактивная форма работы. Вместо этого, Вам необходимо каким-то образом встроить SQL-запросы в программу, написанную, скажем на языке C++. Но как это сделать?

- Как объяснить компилятору C++, что часть Вашей программы это не операторы C++, а операторы какого-то SQL?
- Как соединить возможности языка программирования высокого уровня (переменные, ветвления, циклы) и возможности SQL (запросы на языке, близком к естественному, где уже не на программиста, а на СУБД ложится труд по их оптимизации и выбору плана выполнения)?

Эти и некоторые другие проблемы решаются несколькими способами. Эти способы частично описаны и в стандарте SQL-92. На

настоящий момент используются 3 варианта программного SQL: статический, динамический и основанный на различных API. Посмотрим, что представляет собой каждый вариант.

6.5.1. Статический SQL

Статический SQL – разновидность программного SQL, предназначенная для встраивания SQL-операторов в текст программы на языке программирования высокого уровня.

Рассмотрим еще раз алгоритм выполнения SQL-запросов в интерактивном режиме работы. Легко видеть, что пользователь вынужден ожидать результатов выполнения запроса в течение всего времени работы алгоритма. 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. Итак, программный, а в частности и статический 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 в последствии согласованную работу программы и СУБД. Заранее (на этапе компиляции) выполнять действия по анализу и оптимизации запросов, экономя время, затрачиваемое на этапе выполнения программы.

Рассмотрим два основных этапа, связанных с работой статического SQL, – компиляция программы и работа (выполнение) программы.

Схема компиляции и сборки программы выглядит следующим образом:

- Программа, включающая операторы языка программирования высокого уровня 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. создатели 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 базе данных» (Database Request Module, DBRM) [8]. 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. 50. Схема компиляции программы со встроенными инструкциями статического SQL

Рис. 51. Схема выполнения программы со встроенными инструкциями статического SQL

Схема выполнения программы выглядит следующим образом:

- 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.

Основными **командами** статического SQL являются:

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, в последствии 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.
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, Вы не можете параметризовать имя таблицы, из которой производится выборка, а также названия столбцов. В результате мы приходим к следующему выводу.

При использовании статического варианта вложенного (программного) SQL необходимо на этапе написания программы точно знать состав запросов, которые необходимо будет выполнять в 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.

6.5.2. 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 динамический SQL. Лишь в 1992 году в стандарт SQL-2 были включены спецификации динамического SQL. Для того, чтобы использовать этот инструментарий в своих программах, необходимо представлять основные концепции, основные идеи, заложенные в динамический SQL и принципы его функционирования. Именно это и является предметом рассмотрения в данном разделе.

Теоретики считают основной концепцией динамического SQL 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 [8].

Вспомним, как работал статический SQL: схема использования подразумевала 2 этапа – компиляция программы и выполнение программы. При этом на этап компиляции ложилась основная нагрузка в том смысле, что он решал вопросы проверки, разбора и оптимизации запросов, поскольку было известно, что именно разбирать и

оптимизировать. Совершенно очевидно, что подобную двухэтапную схему нельзя реализовать для динамического SQL, ведь на этапе компиляции программы еще неизвестно, что именно нужно разбирать и оптимизировать.

Каковы последствия этого изменения? Любой механизм имеет свои достоинства и недостатки.

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.

Недостаток динамического SQL – низкая производительность по сравнению со статическим. Действительно, многие из операций, выполнявшихся на этапе компиляции один раз, теперь выполняются непосредственно во время работы программы. В связи с этим, там, где только возможно, представляется правильным рекомендовать использование статической разновидности SQL, применяя аппарат динамического SQL там, где это действительно необходимо.

Рассмотрим схему функционирования динамического SQL. Схема предусматривает одноэтапное и двухэтапное выполнение инструкций.

Одноэтапное выполнение инструкций осуществляется командой EXECUTE IMMEDIATE.

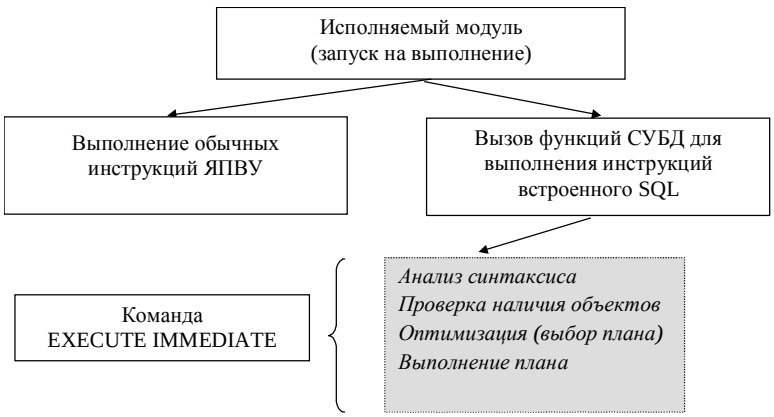


Рис. 52. Схема выполнения программы со встроенными инструкциями динамического SQL с применением одноэтапной схемы

Схема выполнения инструкции подразумевает:

- динамическое формирование команды SQL в строковом виде во время работы программы;
- передача 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. 53. 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.

Основными **командами** динамического SQL являются:

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.

6.6. Интерфейсы программирования приложений (API). DB-Library, ODBC, OCI, JDBC

В предыдущем разделе мы рассмотрели одну из разновидностей языка SQL – программный или встроенный SQL. Как замечено выше, программный SQL отличается от обычной, интерактивной формы наличием некоторых специальных инструкций, а также механизмом трансляции и выполнения 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 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 более прогрессивным, чем программный SQL, на самом же деле, такой вывод вряд ли является верным. Так, на настоящий момент очень активно используются оба подхода. В каждом из них существуют свои достоинства и недостатки и границы разумной применимости. Как обычно, выбор того, каким из подходов воспользоваться, лежит на административной группе проекта (менеджерах проекта), которая принимает решения в зависимости от особенностей конкретной задачи, имеющегося времени, специалистов и, иногда, финансовых ресурсов. В данном разделе мы рассмотрим подход, связанный с применением подхода, основанного на интерфейсе программирования приложений.

Посмотрим, как работают прикладные программы, использующие различные API. Как можно заметить, принцип работы от одной библиотеки к другой не претерпевает принципиальных изменений. Схема работы приложения совместно с SQL API выглядит следующим образом [8]:

- 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 [8].

Из имеющихся SQL 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 книге «Грофф Дж., Вайнберг П. Энциклопедия SQL, 3-е издание. Изд-во «Питер», 2003г.».

6.1.1. Библиотека DB-Library

Библиотека DB-Library реализует интерфейс программирования приложений для совместной работы с известной 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;
- dbresults() – получить результаты выполнения очередной инструкции SQL в текущем пакете;
- dbcancel() – прекращение выполнения пакета инструкций SQL;
- dbbind(), dbdata(), dbnextrow(), dbnumcols(), dbdatlen()… – для обработки 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.

На самом деле, несмотря на кажущуюся простоту, библиотека DB-Library представляет собой большой и сложный механизм. Так, в библиотеки предусмотрены специальные механизмы обработки ошибок, разные способы передачи результатов выполнения запросов в прикладную программу и т.д.

6.1.2. Протокол ODBC

ODBC (Open Database Connectivity – открытый доступ к базам данных) – разработанный компанией Microsoft универсальный интерфейс программирования приложений для доступа к базам данных [8].

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 [8]:

- **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 API, транслируя вызовы функций API в вызовы внутренних специализированных функций СУБД. Таким образом решается проблема стандартизации. Для многих современных СУБД существуют специализированные драйвера ODBC, отдельно устанавливаемые в операционную систему;
- **диспетчер драйверов ODBC:** данный программный механизм представляет средний уровень ODBC, управляя процессом загрузки необходимых драйверов.

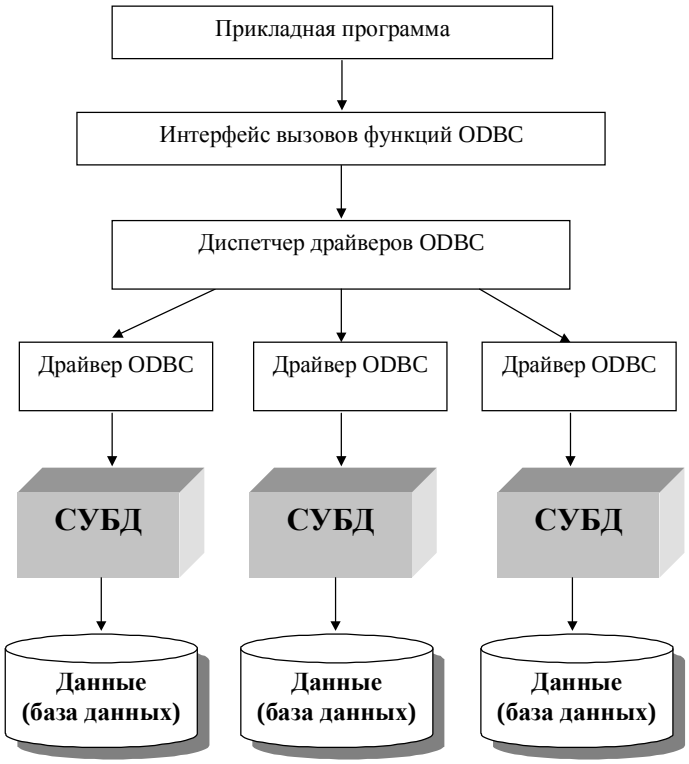


Рис. 54. Схема выполнения программы с использованием протокола ODBC для доступа к данным

6.6.3. Протокол OCI

Интерфейс вызовов Oracle [8] – Oracle Call Interface – OCI – специальный инструмент, представляющий собой программный интерфейс в Oracle.

Данный интерфейс появился еще в ранних версиях Oracle и продолжает свое существование по сей день, несмотря на успешное восхождение ODBC. Более того, при переходе от версии к версии OCI продолжает развиваться. Так, при переходе от версии 7 к версии 8 в OCI были внесены существенные изменения. По сути своей функции OCI очень похожи на функции динамического SQL, особенно в той

части, которая касается старой версии OCI. В новую версию были внесены изменения, многие из которых идеологически заимствованы из ODBC. Интерфейс OCI подробно описан в литературе и может быть изучен самостоятельно.

6.6.4. Протокол JDBC

JDBC (Java Database Connectivity) – представляет собой API для выполнения SQL-запросов к базам данных из программ, написанных на языке Java [8].

Рассмотрим кратко историю возникновения и основные принципы JDBC. Более полное описание можно обнаружить в соответствующей литературе, например [8].

Исторически, языки C/C++ долго являлись доминирующими на рынке разработки коммерческого программного обеспечения. Много копий сломано в том числе и в литературе на тему «Какой язык лучше?». Очевидно, что само понятие «лучше» существенно зависит от предметной области, решаемой задачи, имеющихся ресурсов и т.д. В рамках данного пособия мы не ставим своей задачей проведение сравнительного анализа различных языков программирования (интересующиеся могут обратиться, например, к работам Гради Буча и других ведущих теоретиков), однако то, что языки C/C++ доминируют на рынке разработки ПО, является тем фактом, который на настоящий момент трудно оспорить.

По этой причине долгое время все разрабатываемые дополнительные средства программирования (такие как, например, SQL API) ориентировались в первую очередь именно на C/C++. Однако 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. Имея стандартный набор операторов, будучи весьма похожим на язык C++, Java быстро завоевал большую популярность на рынке разработки приложений для Интернет.

Если обратиться к части данного пособия, содержащей описание основных архитектур, можно увидеть, что Интернет-приложения занимают существенное место на рынке, работая в рамках 2-х, 3-х 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 увеличение возможностей интерфейса было проведено его идеологическое разделение на 2 основных части: Core API (основные возможности) и Extensions API (так называемые расширения).

В литературе указаны следующие основные параметры JDBC [8]:

- **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 выполняют еще и роль обновления 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/отключения от базы данных [8].

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.

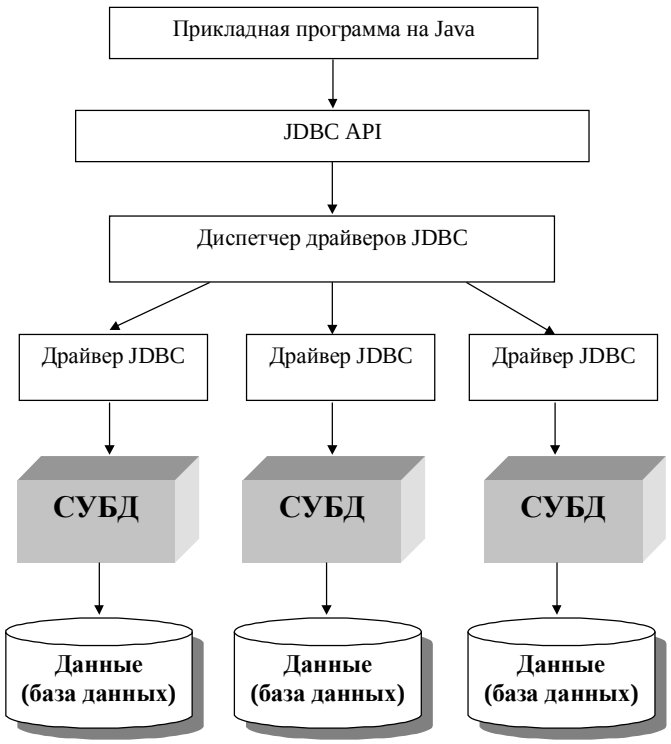


Рис. 55. Схема выполнения программы на Java с использованием протокола JDBC для доступа к данным

В отличие от ODBC, драйверы JDBC подразделяются на 4 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).

ГЛАВА 7. ТЕНДЕНЦИИ РАЗВИТИЯ БАЗ ДАННЫХ

7.1. Объектно-ориентированные базы данных

В начале 90-х годов XX-го века начались активные попытки по внедрению объектно-ориентированных технологий в отрасль проектирования и разработки баз данных. Так, 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 [7] 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 [7].
- 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 [7].
- 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 [7].

Здесь и далее по тексту условимся не отступать от традиций и понимать под объектно-ориентированным программированием (ООП) сразу три указанных выше аспекта.

Итак, 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 что 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 данный объект.

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 [7]: 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 допустимость 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, имеющим отношение к ООБД, в литературе [8] относят:

- объекты (в ООБД любая сущность – объект и обрабатывается как объект [8]).

Отметим, что здесь используется понятие «объект» объектно-ориентированного программирования, которое отличается от понятия «объект», рассматриваемого в главе 2.

- § классы (понятие тип данных из реляционной модели заменяется понятиями класс и подкласс [8]);
- § наследование (классы образуют иерархию наследования, заимствуя свойства друг друга);
- § атрибуты (характеристики объекта моделируются его атрибутами [8]);
- § сообщения и методы (каждый класс имеет определенную совокупность методов, классы взаимодействуют друг с другом посредством механизма сообщений [8]);
- § инкапсуляция (внутренняя структура объектов скрыта [8]);
- § идентификаторы объектов – дескрипторы (объекты идентифицируются с помощью специальных указателей – дескрипторов [8]).

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).

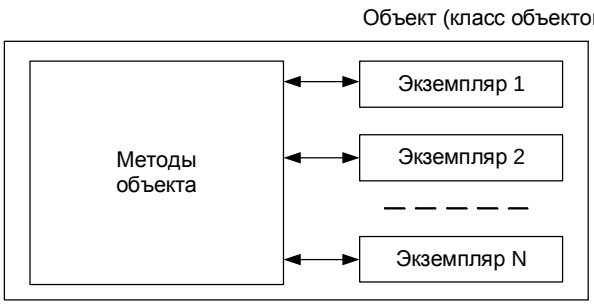


Рис. 56. Схема представления объекта

Используя «наследование» всем объектам ПРДРАЗДЕЛЕНИЕ на рис.57 можно приписать свойство объекта-родителя (ФАКУЛЬТЕТ) – название факультета, номер факультета.

Выделим основные положительные черты ООБД по сравнению с реляционной базой данных (РБД).

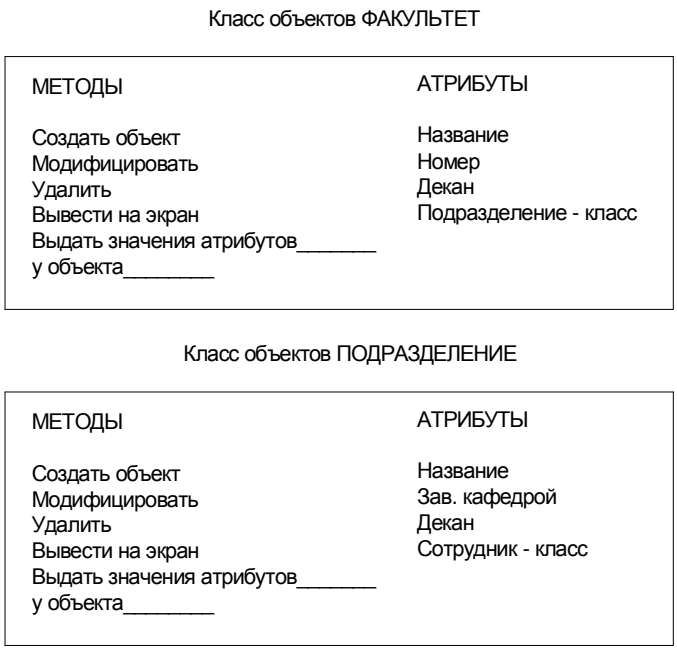


Рис.57. Фрагменты представления классов объектов

Сравнивая объектно-ориентированный и реляционный подходы к БД, можно отметить следующие особенности. В реляционных БД 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 отражает

внутренней структуры для многих предметных областей, является тем самым искусственной и становится совершенно непонятной при увеличении количества таблиц. Основной причиной несостоятельности реляционного подхода состоит 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 должно действительно упростить процесс проектирования и разработки. Опыт одного авторов по созданию соответствующей ООБД для моделирования физических процессов в прочностной механике показывает все удобство применения соответствующей технологии именно в простоте и естественности описания предметной области. Однако, нельзя не остановиться и на недостатках. К сожалению, в ООБД существуют свои проблемы. Например, там отсутствует мощная математическая база, лежащая в основе реляционной модели. Кроме того, в ООБД обычно приходится обходиться без интерпретируемых языков запросов, таких как SQL, что автоматически приводит к трудностям при работе с данными.

ООСУБД отличаются от реляционных СУБД тем, что программный интерфейс создания приложения либо очень слаб, либо вообще отсутствует. Это означает, что при написании приложения на основе ООБД не существует конструкторов (не считая, например, конструктор создания списка полей в объекте, который, например, поставляется вместе с ООСУБД ObjectStore). Поэтому разработчик создает приложения на одном из языков.

Один из ведущих аналитиков в этой области С. Кузнецов характеризует состояние дел в данной отрасли следующим образом: «Обсуждение затрудняло то обстоятельство, что в области ООБД отсутствовал общепринятый набор терминов и определений (эта ситуация сохраняется и сегодня). Сторонники ООБД выдвигали два основных аргумента. Во-первых, они утверждали, что ООБД представляют хорошую основу расширяемых СУБД. Во-вторых, идея наследования может быть упрощена до уровня, понимаемого разработчикам приложений. У противников тоже имелись два аргумента: 1) ООБД по своей природе не слишком пригодны для

непредвиденных запросов и поэтому не подходят для систем поддержки принятия решений; 2) типичный для ООБД навигационный интерфейс доступа возвращает нас во времена CODASYL. Однако реальной проблемой являлось то, что термин «объектная ориентированность» относился к слишком большому числу различных вещей. В общем-то за десять лет почти ничего не изменилось. Существует ряд коммерческих ООСУБД, они довольно активно используются, но по-прежнему общего согласия по поводу понятий и терминов нет» [41].

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 [9, 18, 27, 42 и др.]. Мы здесь лишь подведем краткие итоги:

- 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 на внесение 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 базы данных

Общая и вполне разумная идея заключается, с нашей точки зрения, в применении гибких подходов, позволяющих не ограничивать собственные возможности догматическими положениями, а, напротив, применяя различные приемы, адаптироваться к любой возникшей ситуации.

Именно эту идею и проповедуют разработчики объектно-реляционных баз данных. Так, соответствующие базы данных соединяют в себе лучшие качества реляционных и объектно-ориентированных баз данных. Рассмотрим кратко основные элементы, осуществляющие объектные расширения в реляционных базах данных, уже используемые на сегодняшний день:

- **Хранение больших объемов данных**. Наряду с теми данными, которые хранились в БД традиционно, современные объектно-

реляционные базы данных позволяют хранить в столбцах таблицы картинки, видео-ролики и другие большие документы.

- **Списки в столбцах.** Более того, объектно-реляционные базы данных позволяют хранить в столбцах целые списковые структуры.
- **Пользовательские расширения. В** объектно-реляционных базах данных пользователи имеют возможность вмешиваться в изначально предоставляемый СУБД инструментарий, создавая, в частности, новые пользовательские типы данных.
- **Хранимые процедуры.** В определенном смысле хранимые процедуры также являются объектным расширением, осуществляя необходимые пользователю воздействия на данные (стандартный для ООП процедурный подход).

В качестве примеров реализации такого подхода можно указать доработку СУБД DB2, произведенную фирмой IBM, а также добавление объектной надстройки над реляционным ядром систем ORACLE, произведенное фирмой ORACLE.

7.2. 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 появление баз данных проходило в период господства больших ЭВМ. База данных велась на одной ЭВМ, все пользователи работали именно на ЭВМ (возможные режимы работы описаны в разделе 1.5.). Других вариантов использования вычислительной техники в тот момент времени 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 с одной стороны, 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. 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 это совокупность логически взаимосвязанных разделяемых данных (и описаний их структур), физически распределенных в компьютерной сети.

Распределенная база данных (Ра БД) состоит из единой логической базы данных, разделенной на некоторое количество фрагментов.

Каждый фрагмент хранится на одном из компьютеров компьютерной сети.

Система управления распределенной базой данных – это программная система, обеспечивающая управление 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 (Ра СУБД) 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.

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).

В качестве одного 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.

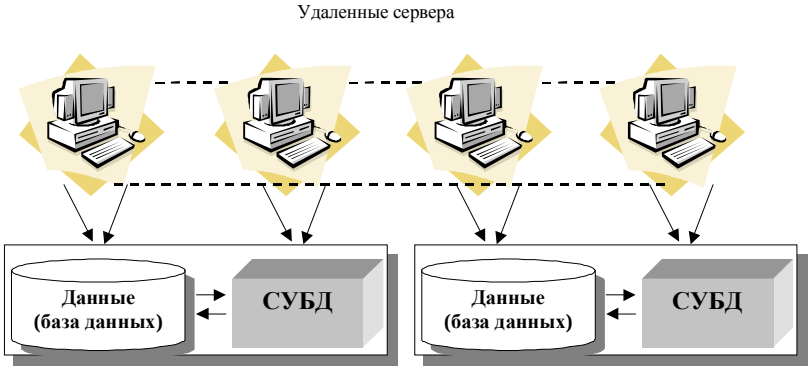


Рис. 58. Распределенная база данных

Дейт К.Дж. провозглашает следующий фундаментальный принцип распределенной базы данных [9]. Для пользователя распределенная система должна выглядеть точно так же, как нераспределенная система. Из этого принципа следует ряд следующих правил [9].

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. раздел 1.1.), 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.).

Существенным достоинством этой модели является приближение данных к месту их порождения, что позволяет существенно повысить их достоверность. Недостатком модели является 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, связанную с 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 [8, 9, 18 и др.]. Здесь мы хотим лишь поставить 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, связанные с ведением 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.

СПИСОК ЛИТЕРАТУРЫ

1. Баженова Н.Ю. Visual Fox Pro 6.0 – М.: Диалог-МИФИ. 1999. – 416 с.
2. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 1998. – 176с.
3. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. Учебник. – СПб: Питер. 2000 – 384 с.
4. Горев А., Ахаян Р., Макашаринов С. Эффективная работа с СУБД. СПб.: Питер, 1997. – 700 с.
5. Горев А., Макашарнпов С., Владимиров Ю. Microsoft SQL Server 6.5 для профессионалов – СПб: Питер, 1998 – 464 с.: ил.
6. Грабер Мартин SQL. Справочное руководство. – М: Изд-во Лори. 1997. – 291с.
7. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на С++. Второе издание.
8. Грофф Дж., Вайнберг П. Энциклопедия SQL. 3-е изд. СПб.: Питер, 2003.
9. Дейт К. Дж. Введение в системы баз данных.: Пер. с англ. – 6-е изд. – К.: Диалектика, 1998. – 784 с.
10. Джеймс Р. Грофф, Пол Н. Вайнберг, SQL: полное руководство: пер. с англ. – К.: Издательская группа BHV, 2000. – 608с.
11. Джейсон С. Каучмэн, Ульрике Швинн. Oracle8i Certified Professional DBA Подготовка администраторов баз данных. Пер. с англ. – М.: издательство «Лори», 2002.
12. Диго С.М. Проектирование баз данных. – М.: Финансы и статистика, 1988. – 216 с.
13. Информационные системы общего назначения (Аналитический обзор систем управления базами данных). Пер. с англ. – М, Статистика, 1975. – 472 с.
14. Калянов Г.Н. CASE структурный системный анализ (автоматизация и применение). – М.: «Лори», 1996.
15. Каратыгин С.А., Тихонов А.Ф., Тихонов Л.Н. Visual FoxPro 6. – М.: ЗАО Изд-во «Бином», 1999. – 784 с.
16. Карпова Т. Базы данных. Модели, разработка, реализация. СПб.: Питер, 2001. – 304 с.

17. Когаловский М.Р. Технология баз данных на персональных ЭВМ. – М.: Финансы и статистика, 1992. – 224 с.
18. Конноли Томас, Бегг Каролин, Страчан Анна, Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.
19. Корнеев В.В., Гариев А.Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации. – М.: «Нолидж», 2000.– 352 с.
20. Крёнке Д. Теория и практика построения баз данных. 8-е изд. – СПб: Питер, 2003. – 800 с.
21. Майерс Г. Архитектура современных ЭВМ. т.2. М.: Мир, 1985.
22. Мамаев Е. Microsoft SQL Server 2000 в подлиннике. СПб.: Изд-во BHV, 2001.
23. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Шерса. – М.: Мир, 1980. – 664 с.
24. Мейер Д. Теория реляционных баз данных. – Пер. с англ. – М.: Мир, 1987. – 608 с.
25. Попов А.А. Создание приложений для FoxPro 2.5/2.6 в DOS и WINDOWS. – М.: Издательство «ДЕСС», 1999. – 672 с.
26. Проектирование баз данных // Методические материалы для самостоятельной работы по курсу «Базы данных, модели и управление данными». – Н.Новгород. 1993.
27. 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.
28. Ульман Дж. Основы систем баз данных: Пер. с англ. / Под ред. М.Р. Когаловского. – М.: Финансы и статистика, 1983. – 334 с.
29. Ульман Дж. Д., Уидом Дж. Введение в системы баз данных. – Пер. с англ. – М.: Изд-во «Лори», 2000. – 374 с.
30. Хаббард Дж. Автоматизированное проектирование баз данных: / Пер. с англ. Под ред. А.Л. Щерса. – М.: Мир, 1984. – 296 с.
31. Хансен Гэри, Хансен Джэймс Базы данных: разработка и управление / Пер. с англ. – М.: ЗАО «Издательство БИНОМ», 1999. – 704 с.
32. Харрингтон Джен Л. Проектирование реляционных баз данных. – М.: «Лори», 2000. – 230 с.

33. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных. Учебник для вузов. – СПб.: КОРОНА принт, 2000. – 416 с.
34. Хотка Дэн, Oracle 9i: Пер. с англ. – СПб.: ООО «ДиаСофтЮП», 2002. – 560с.
35. Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н. Базы и банки данных, ВШ, 1986, 1992.
36. Шумаков П.В. Delphi 3 и создание приложений баз данных. М.: Изд-во «Нолидж», 1998.
37. Rentsch, T. September 1982. Object-Oriented Programming. SIGPLAN Notices vol.17(12), p.51.
38. Зеленков Ю.А. Введение в базы данных http://www.vsma.ac.ru/~pbarm/libraru/books/ db/toc.html.
39. Кириллов В.В. Основы проектирования баз данных. Учебное пособие. Сервер FORUM http://www.citforum.ru.
40. Кузнецов С.Д. Основы современных баз данных http://www.citforum.ru.
41. Кузнецов С. Будущие направления исследований в области баз данных: десять лет спустя. http://www.citforum.ru.
42. Кузнецов С. Объектно-ориентированные базы данных – основные концепции, организация и управление: краткий обзор. http://www.citforum.ru.
43. Трифонов Ю.В., Визгунов А.Н. Методические указания по выполнению лабораторных работ (курс «Базы данных и знаний»). Ч.1. Фонд компьютерных изданий Нижегородского государственного университета, 2001. (http://www.unn.ru/rus/books/table.html).
44. Сайт Oracle http://www.oracle.com.
45. Сайт Sybase http://www.sybase.com.
46. Сайт компании IBM в России http://www.ibm.com/ru.
47. Сайт компании Interface ltd http://www.interface.ru.
48. Шнитман В.З., Кузнецов С.Д. Серверы корпоративных баз данных. http://www.emanual.ru.
49. Сайт «Открытые системы» http://www.osp.ru.

УЧЕБНАЯ ПРОГРАММА КУРСА

Цели и задачи курса

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 обуславливает 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.

Отражением потребности в специалистах такого рода является включение курса по базам данных в учебный план целого ряда специальностей подготовки. Так, в Нижегородском государственном университете им. Н.И.Лобачевского такой курс читается на ряде факультетов:

- факультете вычислительной математики и кибернетики для трех специальностей и направлений подготовки («Прикладная математика и информатика», «Прикладная информатика», «Информационные технологии»);
- экономическом факультете для специальности «Прикладная информатика»;
- механико-математическом факультете для специальности «Прикладная математика и информатика».

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

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

Отметим, что в данном учебном курсе не ставится задача детального изучения конкретных программных систем управления базами данных (СУБД). Изучение конкретных СУБД должно рассматриваться в отдельных систематизированных курсах.

Настоящая учебная программа предназначена для подготовка по направлениям и специальностям подготовки «Прикладная математика и информатика», «Информационные технологии». Программа может быть также использована для подготовки по направлению «Прикладная информатика».

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

- тенденции развития основных понятий представления данных и интегрирование 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 специфицирования к конкретной модели данных СУБД;
- анализ моделей физического представления данных.

Рассмотрение указанных вопросов иллюстрируется на примерах конкретных систем управления базами данных – Access и MS SQL-сервер. В задачи курса входит также получение элементарных навыков работы с этими системами.

Дисциплины, изучение которых необходимо для данного курса

Курс «Базы данных» опирается на материалы следующих курсов:

- основы построения ЭВМ;
- ЭВМ и программирование
- дискретная математика.

Программа курса (36 ч. лекций, 18 ч. лабораторных работ)

1. Введение в базы данных. Общая характеристика основных понятий обработки данных (6 часов)

1.1. Развитие основных понятий представления данных. Основные понятия программирования, связанные с данными. Понятие переменной, понятие массива. Появление новых понятий программирования (поле, запись, файл) как следствие расширения круга решаемых задач и их отражения в системах программирования. Использование несколькими задачами общих данных. Интегрирование данных. База данных.

1.2. Системы управления базами данных как интерфейс между прикладными программами и базами данных. Краткий обзор наиболее распространенных СУБД для персональных ЭВМ. Основные функции систем управления базами данных с иллюстрацией сценариев их реализации в конкретных СУБД. Банк данных. Требования, предъявляемые к современным средствам хранения данных.

1.3. Краткий обзор литературы и других доступных источников.

1.4. Различные представления о данных в базах данных. Модели обрабатываемых данных (внешнее представление, концептуальная модель, структура хранения).

1.5. Различные модели организации работы пользователей с базой данных. Модель с централизованной архитектурой. Модель с автономным персональными ЭВМ. Модель вычислений с сетью и файловым сервером (Архитектура «файл-сервер»). Распределенная модель вычислений (Архитектура «клиент-сервер»). Распределенная модель вычислений (Клиент-сервер. Трехзвенная (многозвенная) архитектура).

1.6. Обзор СУБД. Настольные СУБД. Серверные СУБД.

1.7. Основные этапы проектирования базы данных. Жизненный цикл базы данных (создание, апробация, исправление ошибок, опытная эксплуатация, сопровождение). Структуры хранения данных как основа базы данных.

1.8. Проблема целостности базы данных. Транзакции и блокировки.

2. Концептуальное моделирование базы данных (6 часов)

2.1. Формальное описание предметной области Основные используемые понятия (сущность,связь. типы связей).

2.2. Описание информационного представления предметной области Атрибуты.
2.3. Описание информационных потребностей пользователя Ключи. Типы запросов.
2.4. Построение ER-диаграмм.
2.5. Выявление и моделирование сущностей и связей.
2.6. Построение концептуальной модели.
    2.6.1. Моделирование локальных представлений Варьирование понятиями «Атрибут», «Сущность», «Связь».
    2.6.2. Объединение локальных моделей Идентичность. Агрегация. Обобщение.
2.7. Пример построения диаграммы «Сущность-Связь»
2.8. Ограничения целостности Внешние ограничения. Ограничения, описанные с помощью специальных конструкций.
2.9. Средства автоматизированного проектирования концептуальной модели. Примеры использования CASE- средств.

3. Модели данных СУБД как инструмент представления концептуальной модели (4 часа)

3.1. Общие представления о модели данных. Основные используемые понятия (элемент, запись, файл,группа) Основные составляющие описания.
3.2. Сетевая модель данных Представление связей.
3.3. Иерархическая модель данных Представление связей.
3.4. Реляционная модель данных.
3.5. Многомерная модель данных. OLAP-технология.

4. Формализация реляционной модели (6 часов)

4.1. Формализованное описание отношений и схемы отношений Свойства отношений.
4.2. Манипулирование данными в реляционной модели Реляционная алгебра.Реляционное исчисление.
4.3. Операции реляционной алгебры Примеры представления запросов как последовательность формальных операций реляционной алгебры.
4.4. Использование формального аппарата для оптимизации схем отношений.
    4.4.1. Проблема выбора рациональных схем отношений Нормальные формы. Первая нормальная форма.

4.4.2. Функциональные зависимости (зависимости между атрибутами отношения) Ключи.
4.4.3. Правила вывода.
4.4.3. Декомпозиция схемы отношения.
4.4.4. Выбор рационального набора схем отношений путем нормализации Вторая номальная форма. Третья нормальная форма. Нормальная форма Бойса-Кодда.
4.4.5. Пример нормализации до ЗНФ.
4.4.6. Целостная часть реляционной модели. Реализация условия целостности данных в современных СУБД.

5. Физические модели данных (структуры хранения) (4 часа)

5.1. Структура памяти ЭВМ Внешняя и оперативная память.
5.2. Представление экземпляра логической записи в оперативной памяти.
5.3. Организация обмена между оперативной и внешней памятью.
5.4. Структуры хранения данных во внешней памяти ЭВМ.
5.4.1. Последовательное размещение физических записей Оценка числа действий при выполнении основных операций поиска данных, чтения, занесения данных, модификации(корректировки), удаления.
5.4.2. Последовательное размещение физических записей с упорядочением по ключу. Оценка числа действий при выполнении основных операций поиска данных, чтения, занесения данных, , корректировки, удаления.
5.4.3. Размещение физических записей в виде списковой структуры. Оценка числа действий при выполнении основных операций поиска данных, чтения, занесения данных, корректировки, удаления.
5.4.4. Использование индексов (индексирование) Оценка числа действий при выполнении основных операций поиска данных, чтения, занесения данных, корректировки, удаления.
5.4.5. Бинарное дерево (B-дерево) Оценка числа действий при выполнении основных операций поиска данных, чтения, занесения данных, корректировки, удаления.
5.4.6. Размещение записей с использованием хэширования. Оценка числа действий при выполнении основных

операций поиска данных, чтения, занесения данных, корректировки, удаления.

5.4.7. Комбинированные структуры хранения.

6. Анализ современной технологии реализации баз данных. Языки и стандарты (8 часов)

6.1. Структура современной СУБД на примере Microsoft SQL Server. Архитектура базы данных Физический и логические уровни данных.

6.2. Программное окружение БД. Проблемы доступа и обработки данных.

- 6.2.1. Проблемы доступа и обработки данных.
- 6.2.2. Навигационный подход.
- 6.2.3. Подход, основанный на использовании интерпретируемых языков запросов.

6.3. Понятие языка SQL и его основные части.

- 6.3.1. История возникновения и стандарты языка SQL.
- 6.3.2. Достоинства языка SQL.
- 6.3.3. Разновидности SQL.

6.4. Понятие интерактивного SQL. Элементы интерактивного SQL. Использование SQL для манипулирования данными.

- 6.4.1. Использование SQL для выбора информации из таблицы.
- 6.4.2. Использование SQL для выбора информации из нескольких таблиц.
- 6.4.3. Использование SQL для вставки, редактирования и удаления данных в таблицах.
- 6.4.4. Язык SQL и операции реляционной алгебры.

6.5. Программный (встроенный) SQL.

- 6.5.1. Статический SQL.
- 6.5.2. Динамический SQL.

6.6. Интерфейсы программирования приложений (API). DB-Library, ODBC, OCI, JDBC.

- 6.6.1. Библиотека DB-Library.
- 6.6.2. Протокол ODBC.
- 6.6.3. Протокол OCI.
- 6.6.4. Протокол JDBC.

7. Тенденции развития баз данных

7.1. Объектно-ориентированные базы данных.

7.2. Распределенные базы данных.

Список литературы

УЧЕБНЫЕ ПОСОБИЯ

1. Швецов В.И., Визгунов А.Н., Мееров И.Б. Базы данных. Н.Новгород: Изд-во ННГУ, 2004.
2. Карпова Т. Базы данных. Модели, разработка, реализация. С.-Петербург: Питер, 2001
3. Хомоненко А.Д., Цыганков В.М., Мальцев М.Г. Базы данных. Учебник для вузов.- СПб: КОРОНА принт. 2000 – 416 с.
4. Диго С.М. Проектирование баз данных.-М.: Финансы и статистика, 1988.
5. Зеленков Ю.А. Введение в базы данных http://www.vsma.ac.ru/~pbarm/libraru/books/ db/toc.html.
6. Кириллов В.В. Основы проектирования баз данных. Учебное пособие. Сервер FORUM http://www.citforum.ru.
7. Кузнецов С.Д. Основы современных баз данных http://www.citforum.ru.
8. Диго С.М. Проектирование баз данных. – М.: Финансы и статистика, 1988.
9. Четвериков В.Н., Ревунков Г.И., Самохвалов Э.Н. Базы и банки данных, ВШ 1986, 1992.

ОСНОВНЫЕ РЕКОМЕНДУЕМЫЕ МОНОГРАФИИ

1. Дейт К. Дж. Введение в системы баз данных.: Пер. с англ. — 6-е изд. — К.: Диалектика, 1998. — 784 с.: ил — Парал. тит. англ.
2. Когаловский М.Р. Технология баз данных на персональных ЭВМ. – М.: Финансы и статистика, 1992. – 224 с.
3. Ульман Дж. Основы систем баз данных: Пер. с англ./Под ред. М.Р. Когаловского. – М.: Финансы и статистика, 1983. – 334 с.
4. Ульман Дж. Д., Уидом Дж. Введение в системы баз данных. – Пер. с англ. – М.: «Лори», 2000. – 374 с.
5. Горев А., Ахаян Р., Макашаринов С. Эффективная работа с СУБД. С.-Пб.: «Питер», 1997. – 700 с.
6. Грофф Дж., Вайнберг П. Энциклопедия SQL, 3-е издание. С.-Пб.: «Питер», 2003.

ЛИТЕРАТУРА ПО ПРОГРАММНЫМ СРЕДСТВАМ

1. Горев А., Ахаян Р., Макашаринов С. Эффективная работа с СУБД. С.-Петербург, Питер, 1997. – 700 с.
2. Горев А., Макашарипов С., Владимиров Ю. Microsoft SQL Server 6.5 для профессионалов — СПб: Питер, 1998 — 464 с.: ил.
3. Грабер Мартин SQL. Справочное руководство. – М: Изд-во Лори. 1997. – 291с.
4. Джеймс Р. Грофф, Пол Н. Вайнберг. SQL: полное руководство: пер. с англ. – К.: Издательская группа BHV, 2000. – 608с.
5. Джейсон С. Каучмэн, Ульрике Швинн, Oracle8i Certified Professional DBA Подготовка администраторов баз данных, пер. с англ.- М.: издательство «Лори», 2002.
6. Каратыгин С.А., Тихонов А.Ф., Тихонов Л.Н. Visual FoxPro 6. – М.: ЗАО Изд-во «Бином», 1999, – 784 с.: ил.
7. Мамаев Е. Microsoft SQL Server 2000 в подлиннике. С.-Пб.: Изд-во BHV, 2001.
8. Попов А.А. Создание приложений для FoxPro 2.5/2.6 в DOS и WINDOWS. – М.: Издательство «ДЕСС», 1999г. – 672 с.
9. Хотка Дэн. Oracle 9i: Пер. с англ. – СПб.: ООО «ДиаСофтЮП», 2002. – 560с.
10. Шумаков П.В. Delphi 3 и создание приложений баз данных. М.: «Нолидж», 1998.

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА

1. Баженова Н.Ю. Visual Fox Pro 6.0 – М.: Диалог-МИФИ. 1999.-416 с.
2. Вендров А.М. CASE-технологии. Современные методы и средства проектирования информационных систем. – М.: Финансы и статистика, 1998.- 176с.
3. Гаврилова Т.А., Хорошевский В.Ф. Базы знаний интеллектуальных систем. Учебник. – СПб: Питер. 2000 – 384 с.
4. Гради Буч. Объектно-ориентированный анализ и проектирование с примерами приложений на C++. Второе издание.
5. Информационные системы общего назначения (Аналитический обзор систем управления базами данных). Пер. с англ.:- М, Статистика, 1975. – 472 с.
6. Калянов Г.Н. CASE структурный системный анализ (автоматизация и применение). – М.: «Лори», 1996.

7. Конноли Томас, Бегг Каролин, Страчан Анна, Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2000. – 1120 с.: ил.
8. Корнеев В.В., Гариев А.Ф., Васютин С.В., Райх В.В. Базы данных. Интеллектуальная обработка информации. – М.: «Нолидж», 2000. – 352 с.
9. Крёнке Д. Теория и практика построения баз данных. 8-е издание. – С.-Пб: Питер, 2003. – 800 с.
10. Майерс Г. Архитектура современных ЭВМ. М.: Мир, 1985, т.2.
11. Мартин Дж. Организация баз данных в вычислительных системах: Пер. с англ. /Под ред. А.А. Стогния и А.Л. Шерса. – М-: Мир, 1980. – 664 с.
12. Мейер Д. Теория реляционных баз данных. – Пер. с англ. – М.: Мир, 1987. – 608 с.: ил.
13. 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.: ил.
14. Хаббард Дж. Автоматизированное проектирование баз данных: / Пер. с англ. Под ред. А.Л. Щерса. – М.: Мир, 1984. – 296 с.
15. Хансен Гэри, Хансен Джэймс Базы данных: разработка и управление / Пер. с англ. — М.: ЗАО «Издательство БИНОМ», 1999. – 704.с.: ил.
16. Харрингтон Джен Л. Проектирование реляционных баз данных. – М.: «Лори», 2000. – 230 с.
17. Кузнецов С. Будущие направления исследований в области баз данных: десять лет спустя. http://www.citforum.ru.
18. Кузнецов С. Объектно-ориентированные базы данных – основные концепции, организация и управление: краткий обзор. http://www.citforum.ru.
19. Сайт Oracle http://www.oracle.com.
20. Сайт Sybase http://www.sybase.com.
21. Сайт компании IBM в России http://www.ibm.com/ru.
22. Сайт компании Interface ltd http://www.interface.ru.
23. Шнитман В.З., Кузнецов С.Д. Серверы корпоративных баз данных. http://www.emanual.ru.
24. Сайт «Открытые системы» http://www.osp.ru.

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Целями данного лабораторного практикума являются:

- приобретение практических навыков анализа и моделирования предметной области;
- ознакомление с работой специализированных CASE-средств;
- изучение подхода к обработке данных на основе применения структурированного языка запросов SQL;
- ознакомление с архитектурой «Клиент-Сервер».

Лабораторный практикум предполагает последовательное выполнение студентами 3-х циклов лабораторных работ, моделирующих определенную предметную область, предложенную каждому студенту в рамках конкретного задания.

Каждая лабораторная работа начинается с объяснения преподавателем задания на конкретном примере. Пример сохраняется неизменным на протяжении всего семестра. После объяснения преподавателем задания студенты выполняют свой вариант самостоятельно.

По каждой лабораторной работе должен быть подготовлен отчет.

В первом цикле лабораторных работ (работы 1–2) студенты приобретают навыки анализа и моделирования предметной области в рамках различных моделей данных, а также знакомятся с работой в настольной СУБД (для конкретизации предлагается СУБД MS Access; возможно использование других систем).

В первой работе студент должен построить инфологическую модель (выбирая наилучшую из нескольких вариантов), отобразить ее в сетевую, иерархическую и реляционную модели. Вторая работа посвящена дальнейшему анализу построенной реляционной модели и ее реализации с использованием MS Access. Для заполнения таблиц большим объемом данных используется специально разработанный генератор данных.

Во втором цикле лабораторных работ (работы 3–4) изучаются запросы языка SQL и строится простой интерфейс пользователя. Студенты самостоятельно формируют различные SQL–запросы, получая навыки решения конкретных практических задач.

В третьем цикле лабораторных работ (работы 5–6) студенты самостоятельно расширяют предметную область (или пользуются

предложенным им вариантом расширения), строят диаграммы «Сущность-Связь» и структуры баз данных для расширенной предметной области. В рамках новой модели производится модифицирование написанных ранее запросов к базе данных и написание новых. При проведении работ используются CASE-средства. По окончании производится анализ скрипта для генерирования структуры базы данных, а также изучение принципов создания хранимых процедур и триггеров, созданных преподавателем или сгенерированных автоматически с помощью CASE-средства.

ОПИСАНИЕ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа №1

Цель работы: приобретение навыков анализа предметной области.
Содержание работы:

- Анализ текстового описания предметной области.
- Выделение основных абстракций в предметной области и определение их параметров. Построение инфологической модели.
- Построение реляционной, иерархической и сетевой моделей.

Задания:

1. Проанализировать данные, описанные в предметной области (варианты предметных областей прилагаются).
2. Выделить основные абстракции.
3. Для каждой из абстракций определить параметры, ее характеризующие.
4. Выяснить, как абстракции связаны друг с другом.
5. Рассмотреть различные варианты построения инфологической модели. Выбрать наилучший. Выбор обосновать.
6. Провести моделирование в рамках реляционной, иерархической и сетевой модели.

Лабораторная работа №2

Цель работы: приобретение навыков моделирования предметной области, представленной в виде структурированных наборов данных, в рамках реляционной модели и ее реализации в MS Access.
Содержание работы:

- Анализ описания предметной области.
- Выбор структур таблиц и обоснование данного выбора.

- Наложение условий целостности.
- Определение ключей. Внешний ключ.
- Определение полей. Ограничения, налагаемые на поля.
- Наложение условий целостности.
- Работа с неопределенными значениями (Null).
- Ввод данных.

Задания:

1. Проанализировать данные, описанные в предметной области (варианты предметных областей прилагаются). При помощи среды MS Access создать таблицы для представления предметной области в рамках реляционной модели.
2. Для каждой создаваемой таблицы:
   2.1. Определить условия на значения и сообщения об ошибках некоторых полей.
   2.2. Определить начальное значение для некоторых полей.
   2.3. Определить ключ.
   2.4. Определить внешний ключ (если он есть).
   2.5. Определить (если это возможно) значения некоторых полей с помощью мастера подстановок.
   2.6. Определить обязательные поля.
   2.7. Ввести данные в таблицы. При вводе выяснить, что дает наложение условий на значения полей.
3. Определить схему базы данных, связи между таблицами и наложить условия целостности на таблицы, связанные отношением «один-ко-многим». Показать на примерах, что меняется при включении/выключении каждого из флажков «Обеспечение целостности данных» и «Каскадное обновление связанных записей» и «Каскадное удаление связанных записей».

Лабораторная работа №3

Цель работы: выборка данных из таблиц. Добавление, удаление, редактирование информации. Приобретение практических навыков использования языка SQL.

Содержание работы:

- Выборка данных из одной таблицы. Выбор отдельных полей таблицы. Квалифицированный выбор – предложение WHERE. Сложные условия (использование OR, AND, NOT).
- Выборка данных из связанных таблиц. Работа с главными и подчиненными таблицами (Master-Detail).

- Создание вычислимых полей.
- Сортировка результатов запроса.
- Проблема агрегирования данных. Изучение агрегатных функций (SUM, AVG, MAX, MIN, COUNT).
- Подсчет простых итогов.
- Организация группировки. Группировка по нескольким полям.
- Организация отбора при группировке.
- Создание перекрестных запросов.
- Создание новых таблиц.
- Создание запросов на добавление, редактирование, удаление.

Задания:

1. Простой выбор данных (select, *, from, where, order by, вычислимые поля, distinct).
   1.1. Создать простой запрос на выборку из одной таблицы. Включить несколько полей таблицы.
   1.2. Включить в запрос все поля с помощью знака «*».
   1.3. Создать запрос на выборку данных из двух связанных таблиц.
   1.4. Выбрать несколько полей, по которым сортируется вывод.
   1.5. Определить условия отбора («И» и «ИЛИ»). Создать не менее 2-х запросов.
   1.6. Определить условия отбора с помощью параметра запроса.
   1.7. Создать вычислимые поля.
   1.8. Создать отсортированный по вычислимому полю запрос из нескольких таблиц, в котором определены условия «И» и «ИЛИ».
   1.9. Использовать предложение Distinct.
2. Внешнее объединение таблиц.
   2.1. Создать запрос на внешнее объединение таблиц.
   2.2. Продемонстрировать использование предложений Is null, Is not null.
   2.3. Использовать предложение Like.
   2.4. Использовать оператор UNION.
3. Выбор данных с помощью группирующих запросов с условием (group by, having, min(), max(), sum(), count(), …).
   3.1. Создать итоговый запрос, содержащий несколько итоговых цифр.
   3.2. Создать простой группирующий запрос.
   3.3. Создать группирующий запрос с группировкой по нескольким полям.

3.4. Создать группирующий запрос, в котором определяются условия, причем сначала выполняются вычисления, а затем происходит отбор.
3.5. Создать группирующий запрос, в котором определяются условия, причем сначала происходит отбор, а затем выполняются вычисления.
3.6. Создать группирующий запрос, в котором есть вычислимое выражение, содержащее несколько итоговых полей.

4. Выбор данных с помощью подзапросов.
4.1. Создать запрос с выбором при помощи In.
4.2. Использовать предложения All, Any, Exists.

Лабораторная работа №4

Цель работы: разработка интерфейса пользователя. Создание форм.
Содержание работы:

- Создание форм для ввода, редактирования и удаления записей.
- Создание форм для навигации по базе данных и выполнения запросов.

Задания:

1. Создать формы для ввода каждой из таблиц-справочников.
2. Создать сложную форму для таблиц, связанных отношением 1 ко многим.
3. Создать кнопочную форму, которая бы предоставляла доступ ко всем созданным формам и запросам.
4. Поместить в созданные формы кнопки навигации по записям и работы с формой (закрыть, напечатать, выйти из приложения).
5. Создать макрос для автоматической загрузки кнопочной формы при открытии базы данных.

Лабораторная работа №5

Цель работы: приобретение практических навыков анализа и моделирования предметной области; ознакомление с работой специализированных CASE-средств. Приобретение начальных навыков работы с СУБД, работающими в рамках архитектуры «Клиент-Сервер». Перенос задачи в среду «Клиент-Сервер».
Содержание работы:

- Расширение предметной области.
- Анализ описания расширенной предметной области.
- Выбор структур таблиц и обоснование данного выбора.

- Наложение условий целостности.
- Определение ключей. Внешний ключ.
- Определение полей. Ограничения, налагаемые на поля.
- Наложение условий целостности.
- Работа с неопределенными значениями (Null).
- Ввод данных.
- Использование CASE-средств для создания базы данных по ее описанию.
- Изучение скриптов для создания базы данных для СУБД Oracle (или другой доступной серверной СУБД).
- Изучение текстов хранимых процедур.

Задания:

1. Проанализировать данные, описанные в расширенной предметной области. Определить необходимость создания таблиц для представления предметной области в рамках реляционной модели.
2. Для каждой таблицы:
   2.1. Определить условия на значения и сообщения об ошибках некоторых полей.
   2.2. Определить начальное значение для некоторых полей.
   2.3. Определить ключ.
   2.4. Определить внешний ключ (если он есть).
   2.5. Определить обязательные поля.
   2.6. Определить схему базы данных, связи между таблицами и условия целостности на таблицы, связанные отношением «один-ко-многим».
3. Используя CASE-средство, создать формализованное описание предметной области (диаграмма сущность-связь) и сгенерировать базу данных в формате одной из СУБД.
4. Ввести данные в таблицы. При вводе выяснить, что дает наложение условий на значения полей.
5. Проанализировать скрипты для создания базы данных в СУБД Oracle (или другой доступной СУБД).
6. Проанализировать текст готовых хранимых процедур.

Лабораторная работа №6

Цель работы: приобретение навыков внесения необходимых изменений в программный код после изменения модели предметной области и переноса задачи в среду «Клиент-Сервер».

Содержание работы:

- Модификация ранее созданных запросов с учетом изменений предметной и области и ориентацией на современные серверные СУБД (приведение к стандарту SQL-92).
- Модификация существующего интерфейса.

Задания:

1. Пояснить связь изменения постановки задачи с необходимостью модификации запросов и интерфейса.
2. Адаптировать имеющиеся запросы к расширенной предметной области.
3. Реализовать новые запросы, необходимость которых вызвана расширением предметной области.
4. Добавить новые интерфейсные формы, изменить существующие.

ВИДЫ ПРЕДМЕТНЫХ ОБЛАСТЕЙ

1. Страховая компания

Описание предметной области

Вы работаете в страховой компании. Вашей задачей является отслеживание финансовой деятельности компании.

Компания имеет различные филиалы по всей стране. Каждый филиал характеризуется названием, адресом и телефоном. Деятельность компании организована следующим образом: к Вам обращаются различные лица с целью заключения договора о страховании. В зависимости от принимаемых на страхование объектов и страхуемых рисков, договор заключается по определенному виду страхования (например, страхование автотранспорта от угона, страхование домашнего имущества, добровольное медицинское страхование). При заключении договора Вы фиксируете дату заключения, страховую сумму, вид страхования, тарифную ставку и филиал, в котором заключался договор.

Таблицы

Договоры (Номер договора, Дата заключения, Страховая сумма, Тарифная ставка, Код филиала, Код вида страхования).
Вид страхования (Код вида страхования, Наименование).
Филиал (Код филиала, Наименование филиала, Адрес, Телефон).

Развитие постановки задачи

Нужно учесть, что договоры заключают страховые агенты. Помимо информации об агентах (фамилия, имя, отчество, адрес, телефон), нужно еще хранить филиал, в котором работают агенты. Кроме того, исходя из базы данных, нужно иметь возможность рассчитывать заработную плату агентам. Заработная плата составляет некоторый процент от страхового платежа (страховой платеж это страховая сумма, умноженная на тарифную ставку). Процент зависит от вида страхования, по которому заключен договор.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

2. Гостиница

Описание предметной области

Вы работаете в гостинице. Вашей задачей является отслеживание финансовой стороны работы гостиницы.

Ваша деятельность организована следующим образом: гостиница предоставляет номера клиентам на определенный срок. Каждый номер характеризуется вместимостью, комфортностью (люкс, полулюкс, обычный) и ценой. Вашими клиентами являются различные лица, о которых Вы собираете определенную информацию (фамилия, имя, отчество и некоторый комментарий). Сдача номера клиенту производится при наличии свободных мест в номерах, подходящих клиенту по указанным выше параметрам. При поселении фиксируется дата поселения. При выезде из гостиницы для каждого места запоминается дата освобождения.

Таблицы

Клиенты (Код клиента, Фамилия, Имя, Отчество, Паспортные данные, Комментарий).
Номера (Код номера, Номер, Количество человек, Комфортность, Цена).
Поселение (Код поселения, Код клиента, Код номера, Дата поселения, Дата освобождения, Примечание).

Развитие постановки задачи

Необходимо хранить информацию не только по факту сдачи номера клиенту, но и осуществлять бронирование номеров. Кроме того, для постоянных клиентов, а также для определенных категорий

клиентов, предусмотрена система скидок. Скидки могут суммироваться.

Внести в структуру таблиц изменения, учитывающие этот факт, и изменить существующие запросы. Добавить новые запросы.

3. Ломбард

Описание предметной области

Вы работаете в ломбарде. Вашей задачей является отслеживание финансовой стороны работы ломбарда.

Деятельность Вашей компании организована следующим образом: к Вам обращаются различные лица с целью получения денежных средств под залог определенных товаров. У каждого из приходящих к Вам клиентов Вы запрашиваете фамилию, имя, отчество и другие паспортные данные. После оценивания стоимости принесенного в качестве залога товара Вы определяете сумму, которую готовы выдать на руки клиенту, а также свои комиссионные. Кроме того, определяете срок возврата денег. Если клиент согласен, то Ваши договоренности фиксируются в виде документа, деньги выдаются клиенту, а товар остается у Вас. В случае если в указанный срок не происходит возврата денег, товар переходит в Вашу собственность.

Таблицы

Клиенты (Код клиента, Фамилия, Имя, Отчество, Номер паспорта, Серия паспорта, Дата выдачи паспорта).
Категории товаров (Код категории товаров, Название, Примечание).
Сдача в ломбард (Код, Код категории товаров, Код клиента, Описание товара, Дата сдачи, Дата возврата, Сумма, Комиссионные).

Развитие постановки задачи.

После перехода прав собственности на товар, ломбард может продавать товары по цене, меньшей или большей, чем была заявлена при сдаче. Цена может меняться несколько раз, в зависимости от ситуации на рынке. (Например, владелец ломбарда может устроить распродажу зимних вещей в конце зимы). Помимо текущей цены, нужно хранить все возможные значения цены для данного товара.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

4. Реализация готовой продукции

Описание предметной области

Вы работаете в компании, занимающейся оптово-розничной продажей различных товаров. Вашей задачей является отслеживание финансовой стороны работы компании.

Деятельность Вашей компании организована следующим образом: Ваша компания торгует товарами из определенного спектра. Каждый из этих товаров характеризуется наименованием, оптовой ценой, розничной ценой и справочной информацией. В Вашу компанию обращаются покупатели. Для каждого из них Вы запоминаете в базе данных стандартные данные (наименование, адрес, телефон, контактное лицо) и составляете по каждой сделке документ, запоминая наряду с покупателем количество купленного им товара и дату покупки.

Таблицы

Товары (Код товара, Наименование, Оптовая цена, Розничная цена, Описание).
Покупатели (Код покупателя, Телефон, Контактное лицо, Адрес).
Сделки (Код сделки, Дата сделки, Код товара, Количество, Код покупателя, Признак оптовой продажи).

Развитие постановки задачи

Теперь ситуация изменилась. Выяснилось, что обычно покупатели в рамках одной сделки покупают не один товар, а сразу несколько. Также компания решила предоставлять скидки в зависимости от количества закупленных товаров и их общей стоимости.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

5. Ведение заказов

Описание предметной области

Вы работаете в компании, занимающейся оптовой продажей различных товаров. Вашей задачей является отслеживание финансовой стороны работы компании.

Деятельность Вашей компании организована следующим образом: Ваша компания торгует товарами из определенного спектра. Каждый из этих товаров характеризуется ценой, справочной информацией и признаком наличия или отсутствия доставки. В Вашу компанию

обращаются заказчики. Для каждого из них Вы запоминаете в базе данных стандартные данные (наименование, адрес, телефон, контактное лицо) и составляете по каждой сделке документ, запоминая наряду с заказчиком количество купленного им товара и дату покупки.

Таблицы

Заказчики (Код заказчика, Наименование, Адрес, Телефон, Контактное лицо).
Товары (Код товара, Цена, Доставка, Описание).
Заказы (Код заказа, Код заказчика, Код товара, Количество, Дата).

Развитие постановки задачи.

Теперь ситуация изменилась. Выяснилось, что доставка разных товаров может производиться разными способами, различными по цене и скорости. Нужно хранить информацию по тому, какими способами может осуществляться доставка каждого товара и информацию о том, какой вид доставки (а, соответственно, и какую стоимость доставки) выбрал клиент при заключении сделки.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

6. Бюро по трудоустройству

Описание предметной области

Вы работаете в бюро по трудоустройству. Вашей задачей является отслеживание финансовой стороны работы компании.

Деятельность Вашего бюро организована следующим образом: Ваше бюро готово искать работников для различных работодателей и вакансии для ищущих работу специалистов различного профиля. При обращении к Вам клиента-работодателя, его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в базе данных. При обращении к Вам клиента-соискателя, его стандартные данные (фамилия, имя, отчество, квалификация, профессия, иные данные) также фиксируются в базе данных. По каждому факту удовлетворения интересов обеих сторон составляется документ. В документе указываются соискатель, работодатель, должность и комиссионные (доход бюро).

Таблицы

Работодатели (Код работодателя, Название, Вид деятельности, Адрес, Телефон).

Сделки (Код соискателя, Код работодателя, Должность, Комиссионные).
Соискатели (Код соискателя, Фамилия, Имя, Отчество, Квалификация, Вид деятельности, Иные данные, Предполагаемый размер заработной платы).

Развитие постановки задачи.

Оказалось, что база данных не совсем точно описывает работу бюро. В базе фиксируется только сделка, а информация по открытым вакансиям не храниться. Кроме того, для автоматического поиска вариантов, необходимо вести справочник «виды деятельности».

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

7. Нотариальная контора

Описание предметной области

Вы работаете в нотариальной конторе. Вашей задачей является отслеживание финансовой стороны работы компании.

Деятельность Вашей нотариальной конторы организована следующим образом: Ваша фирма готова предоставить клиенту определенный комплекс услуг. Для наведения порядка Вы формализовали эти услуги, составив их список с описанием каждой услуги. При обращении к Вам клиента, его стандартные данные (название, вид деятельности, адрес, телефон) фиксируются в базе данных. По каждому факту оказания услуги клиенту составляется документ. В документе указываются услуга, сумма сделки, комиссионные (доход конторы), описание сделки.

Таблицы

Клиенты (Код клиента, Название, Вид деятельности, Адрес, Телефон).
Сделки (Код сделки, Код клиента, Код услуги, Сумма, Комиссионные, Описание).
Услуги (Код услуги, Название, Описание).

Развитие постановки задачи

Теперь ситуация изменилась. В рамках одной сделки клиенту может быть оказано несколько услуг. Стоимость каждой услуги фиксирована. Кроме того, компания предоставляет в рамках одной сделки различные виды скидок. Скидки могут суммироваться.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

8. Фирма по продаже запчастей

Описание предметной области

Вы работаете в фирме, занимающейся продажей запасных частей для автомобилей. Вашей задачей является отслеживание финансовой стороны работы компании.

Основная часть деятельности, находящейся в Вашем ведении, связана с работой с поставщиками. Фирма имеет определенный набор поставщиков, по каждому из которых известны название, адрес и телефон. У этих поставщиков Вы приобретаете детали. Каждая деталь наряду с названием характеризуется артикулом и ценой (считаем цену постоянной). Некоторые из поставщиков могут поставлять одинаковые детали (один и тот же артикул). Каждый факт покупки запчастей у поставщика фиксируется в базе данных, причем обязательными для запоминания являются дата покупки и количество приобретенных деталей.

Таблицы

Поставщики (Код поставщика, Название, Адрес, Телефон).
Детали (Код детали, Название, Артикул, Цена, Примечание).
Поставки (Код поставщика, Код детали, Количество, Дата).

Развитие постановки задачи

Теперь ситуация изменилась. Выяснилось, что цена детали может меняться от поставки к поставке. Поставщики заранее ставят Вас в известность о дате изменения цены и о его новом значении. Нужно хранить не только текущее значение цены, но и всю историю изменения цен.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

9. Курсы по повышению квалификации

Описание предметной области

Вы работаете в учебном заведении и занимаетесь организацией курсов повышения квалификации.

В Вашем распоряжении имеются сведения о сформированных группах студентов. Группы формируются в зависимости от

специальности и отделения. В каждой из них включено определенное количество студентов. Проведение занятий обеспечивает штат преподавателей. Для каждого из них у Вас в базе данных зарегистрированы стандартные анкетные данные (фамилия, имя, отчество, телефон) и стаж работы. В результате распределения нагрузки Вы получаете информацию о том, сколько часов занятий проводит каждый преподаватель с соответствующими группами. Кроме того, хранятся также сведения о виде проводимых занятий (лекции, практика), предмете и оплате за 1 час.

Таблицы

Группы (Номер группы, Специальность, Отделение, Количество студентов).
Преподаватели (Код преподавателя, Фамилия, Имя, Отчество, Телефон, Стаж).
Нагрузка (Код преподавателя, Номер группы, Количество часов, Предмет, Тип занятия, Оплата).

Развитие постановки задачи

В результате работы с базой данных выяснилось, что размер почасовой оплаты зависит от предмета и типа занятия. Кроме того, каждый преподаватель может вести не все предметы, а только некоторые.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

10. Определение факультативов для студентов

Описание предметной области

Вы работаете в высшем учебном заведении и занимаетесь организацией факультативов.

В Вашем распоряжении имеются сведения о студентах, включающие стандартные анкетные данные (фамилия, имя, отчество, адрес, телефон). Преподаватели Вашей кафедры должны обеспечить проведение факультативных занятий по некоторым предметам. По каждому факультативу существует определенное количество часов и вид проводимых занятий (лекции, практика, лабораторные работы). В результате работы со студентами у Вас появляется информация о том, кто из них записался на какие факультативы. Существует некоторый минимальный объем факультативных предметов, которые должен

прослушать каждый студент. По окончанию семестра Вы заносите информацию об оценках, полученных студентами на экзаменах.

Таблицы

Студенты (Код студента, Фамилия, Имя, Отчество, Адрес, Телефон).
Предметы (Код предмета, Название, Объем лекций, Объем практик, Объем лабораторных работ).
Учебный план (Код студента, Код предмета, Оценка).

Развитие постановки задачи

Теперь ситуация изменилась. Выяснилось, что некоторые из факультативов могут длиться более одного семестра. В каждом семестре для предмета устанавливается объем лекций, практик и лабораторных работ в часах. В качестве итоговой оценки за предмет берется последняя оценка, полученная студентом.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

11. Распределение учебной нагрузки

Описание предметной области

Вы работаете в высшем учебном заведении и занимаетесь распределением нагрузки между преподавателями кафедры.

В Вашем распоряжении имеются сведения о преподавателях кафедры, включающие наряду с анкетными данными сведения об их ученой степени, занимаемой административной должности и стаже работы. Преподаватели Вашей кафедры должны обеспечить проведение занятий по некоторым предметам. По каждому из них существует определенное количество часов. В результате распределения нагрузки у Вас должна получится информация следующего рода: «Такой-то преподаватель проводит занятия по такому-то предмету с такой-то группой».

Таблицы

Преподаватели (Код преподавателя, Фамилия, Имя, Отчество, Ученая степень, Должность, Стаж).
Предметы (Код предмета, Название, Количество часов).
Нагрузка (Код преподавателя, Код предмета, Номер группы).

Развитие постановки задачи

Теперь ситуация изменилась. Выяснилось, что все проводимые занятия делятся на лекционные и практические. По каждому виду

занятий устанавливается свое количество часов. Кроме того, данные по нагрузке нужно хранить несколько лет.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

12. Распределение дополнительных обязанностей

Описание предметной области

Вы работаете в коммерческой компании и занимаетесь распределением дополнительных разовых работ. Вашей задачей является отслеживание хода выполнения дополнительных работ.

Компания имеет определенный штат сотрудников, каждый из которых получает определенный оклад. Время от времени, возникает потребность в выполнении некоторой дополнительной работы, не входящей в круг основных должностных обязанностей сотрудников. Для наведения порядка в этой сфере деятельности Вы проклассифицировали все виды дополнительных работ, определившись с суммой оплаты по факту их выполнения. При возникновении дополнительной работы определенного вида Вы назначаете ответственного, фиксируя дату начала. По факту окончания Вы фиксируете дату и выплачиваете дополнительную сумму к зарплате с учетом Вашей классификации.

Таблицы

Сотрудники (Код сотрудника, Фамилия, Имя, Отчество, Оклад).
Виды работ (Код вида, Описание, Оплата за день).
Работы (Код сотрудника, Код вида, Дата начала, Дата окончания).

Развитие постановки задачи

Теперь ситуация изменилась. Выяснилось, что некоторые из дополнительных работ являются достаточно трудоемкими и, в то же время, срочными, что требует привлечения к их выполнению нескольких сотрудников. Также оказалось, что длительность работ в каждом конкретном случае составляет разную величину. Соответственно, нужно заранее планировать длительность работы и количество сотрудников, занятых для выполнения работы.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

13. Техническое обслуживание станков

Описание предметной области

Ваше предприятие занимается ремонтом станков и другого промышленного оборудования. Вашей задачей является отслеживание финансовой стороны деятельности предприятия.

Клиентами Вашей компании являются промышленные предприятия, оснащенные различным сложным оборудованием. В случае поломок оборудования они обращаются к Вам.

Ремонтные работы в Вашей компании организованы следующим образом: все станки проклассифицированы по странам-производителям, годам выпуска и маркам. Все виды ремонта отличаются названием, продолжительностью в днях, стоимостью. Исходя из этих данных, по каждому факту ремонта Вы фиксируете вид станка и дату начала ремонта.

Таблицы

Виды станков (Код вида станка, Страна, Год выпуска, Марка).
Виды ремонта (Код ремонта, Название, Продолжительность, Стоимость, Примечания).
Ремонт (Код вида станка, Код ремонта, Дата начала, Примечания).

Развитие постановки задачи

Теперь ситуация изменилась. Несложный анализ показал, что нужно не просто подразделять станки по типам, а иметь информацию о том, сколько раз ремонтировался тот или иной конкретный станок.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

14. Туристическая фирма

Описание предметной области

Вы работаете в туристической компании. Ваша компания работает с клиентами, продавая им путевки. Вашей задачей является отслеживание финансовой стороны деятельности фирмы.

Работа с клиентами в Вашей компании организована следующим образом: у каждого клиента, пришедшего к Вам, собираются некоторые стандартные данные – фамилия, имя, отчество, адрес, телефон. После этого Ваши сотрудники выясняют у клиента, куда он хотел бы поехать отдыхать. При этом ему демонстрируются различные варианты, включающие страну проживания, особенности местного

климата, имеющиеся отели разного класса. Наряду с этим, обсуждается возможная длительность пребывания и стоимость путевки. В случае если удалось договориться, и найти для клиента приемлемый вариант, Вы регистрируете факт продажи путевки (или путевок, если клиент покупает сразу несколько путевок), фиксируя дату отправления. Иногда Вы решаете предоставить клиенту некоторую скидку.

Таблицы

Маршруты (Код маршрута, Страна, Климат, Длительность, Отель, Стоимость).
Путевки (Код маршрута, Код клиента, Дата отправления, Количество, Скидка).
Клиенты (Код клиента, Фамилия, Имя, Отчество, Адрес, Телефон).

Развитие постановки задачи

Теперь ситуация изменилась. Фирма работает с несколькими отелями в нескольких странах. Путевки продаются на одну, две или четыре недели. Стоимость путевки зависит от длительности тура и отеля. Скидки, которые предоставляет фирма, фиксированы. Например, при покупке более 1 путевки, предоставляется скидка 5%. Скидки могут суммироваться.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

15. Грузовые перевозки

Описание предметной области

Вы работаете в компании, занимающейся перевозками грузов. Вашей задачей является отслеживание стоимости перевозок с учетом заработной платы водителей.

Ваша компания осуществляет перевозки по различным маршрутам. Для каждого маршрута Вы определили некоторое название, вычислили примерное расстояние и установили некоторую оплату для водителя. Информация о водителях включает фамилию, имя, отчество и стаж. Для проведения расчетов Вы храните полную информацию о перевозках (маршрут, водитель, даты отправки и прибытия). По факту некоторых перевозок водителям выплачивается премия.

Таблицы

Маршруты (Код маршрурта, Название, Дальность, Количество дней в пути, Оплата).
Водители (Код водителя, Фамилия, Имя, Отчество, Стаж).
Проделанная работа (Код маршрута, Код водителя, Дата отправки, Дата возвращения, Премия).

Развитие постановки задачи

Теперь ситуация изменилась. Ваша фирма решила ввести гибкую систему оплаты. Так, оплата водителям должна теперь зависеть не только от маршрута, но и от стажа водителя. Кроме того, нужно учесть, что перевозку могут осуществлять два водителя.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

16. Учет телефонных переговоров

Описание предметной области

Вы работаете в коммерческой службе телефонной компании. Компания предоставляет абонентам телефонные линии для междугородних переговоров. Вашей задачей является отслеживание стоимости междугородних телефонных переговоров.

Абонентами компании являются юридические лица, имеющие телефонную точку, ИНН, расчетный счет в банке. Стоимость переговоров зависит от города, в который осуществляется звонок, и времени суток (день, ночь). Каждый звонок абонента автоматически фиксируется в базе данных. При этом запоминаются город, дата, длительность разговора и время суток.

Таблицы

Абоненты (Код абонента, Номер телефона, ИНН, Адрес).
Города (Код города, Название, Тариф дневной, Тариф ночной).
Переговоры (Код переговоров, Код абонента, Код города, Дата, Количество минут, Время суток).

Развитие постановки задачи

Теперь ситуация изменилась. Ваша фирма решила ввести гибкую систему скидок. Так, стоимость минуты теперь уменьшается в зависимости от длительности разговора. Размер скидки для каждого города разный.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

17. Учет внутриофисных расходов

Описание предметной области

Вы работаете в бухгалтерии частной фирмы. Сотрудники фирмы имеют возможность осуществлять мелкие покупки для нужд фирмы, предоставляя в бухгалтерию товарный чек. Вашей задачей является отслеживание внутриофисных расходов.

Ваша фирма состоит из отделов. Каждый отдел имеет название. В каждом отделе работает определенное количество сотрудников. Сотрудники могут осуществлять покупки в соответствии с видами расходов. Каждый вид расходов имеет название, некоторое описание и предельную сумму средств, которые могут быть потрачены по данному виду расходов в месяц. При каждой покупке сотрудник оформляет документ, где указывает вид расхода, дату, сумму и отдел.

Таблицы

Отделы (Код отдела, Название, Количество сотрудников).
Виды расходов (Код вида, Название, Описание, Предельная норма).
Расходы (Код расхода, Код вида, Код отдела, Сумма, Дата).

Развитие постановки задачи

Теперь ситуация изменилась. Оказалось, что нужно хранить данные о расходах не только в целом по отделу, но и по отдельным сотрудникам. Нормативы по расходованию средств устанавливаются не в целом, а по каждому отделу за каждый месяц. Неиспользованные в текущем месяце деньги могут быть использованы позже.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

18. Библиотека

Описание предметной области

Вы являетесь руководителем библиотеки. Ваша библиотека решила зарабатывать деньги, выдавая напрокат некоторые книги, имеющиеся в небольшом количестве экземпляров. Вашей задачей является отслеживание финансовых показателей работы библиотеки.

У каждой книги, выдаваемой в прокат, есть название, автор, жанр. В зависимости от ценности книги Вы определили для каждой из них

залоговую стоимость (сумма, вносимая клиентом при взятии книги напрокат) и стоимость проката (сумма, которую клиент платит при возврате книги, получая назад залог). В библиотеку обращаются читатели. Все читатели регистрируются в картотеке, которая содержит стандартные анкетные данные (фамилия, имя, отчество, адрес, телефон). Каждый читатель может обращаться в библиотеку несколько раз. Все обращения читателей фиксируются, при этом по каждому факту выдачи книги запоминаются дата выдачи и ожидаемая дата возврата.

Таблицы

Книги (Код книги, Название, Автор, Залоговая стоимость, Стоимость проката, Жанр).
Читатели (Код читателя, Фамилия, Имя, Отчество, Адрес, Телефон).
Выданные книги (Код книги, Код читателя, Дата выдачи, Дата возврата).

Развитие постановки задачи

Теперь ситуация изменилась. Несложный анализ показал, что стоимость проката книги должна зависеть не только от самой книги, но и от срока ее проката. Кроме того, необходимо добавить систему штрафов за вред, нанесенный книге и систему скидок для некоторых категорий читателей.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

19. Прокат автомобилей

Описание предметной области

Вы являетесь руководителем коммерческой службы в фирме, занимающейся прокатом автомобилей. Вашей задачей является отслеживание финансовых показателей работы пункта проката.

В Ваш автопарк входит некоторое количество автомобилей различных марок, стоимостей и типов. Каждый автомобиль имеет свою стоимость проката. В пункт проката обращаются клиенты. Все клиенты проходят обязательную регистрацию, при которой о них собирается стандартная информация (фамилия, имя, отчество, адрес, телефон). Каждый клиент может обращаться в пункт проката несколько раз. Все обращения клиентов фиксируются, при этом по каждой сделке запоминаются дата выдачи и ожидаемая дата возврата.

Таблицы

Автомобили (Код автомобиля, Марка, Стоимость, Стоимость проката, Тип).
Клиенты (Код клиента, Фамилия, Имя, Отчество, Адрес, Телефон).
Выданные автомобили (Код автомобиля, Код клиента, Дата выдачи, Дата возврата).

Развитие постановки задачи

Теперь ситуация изменилась. Несложный анализ показал, что стоимость проката автомобиля должна зависеть не только от самого автомобиля, но и от срока его проката, а также от года выпуска. Также нужно ввести систему штрафов за возвращение автомобиля в ненадлежащем виде и систему скидок для постоянных клиентов.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

20. Выдача банком кредитов

Описание предметной области

Вы являетесь руководителем информационно-аналитического центра коммерческого банка. Одним из существенных видов деятельности Вашего банка является выдача кредитов юридическим лицам. Вашей задачей является отслеживание динамики работы кредитного отдела.

В зависимости от условий получения кредита, процентной ставки и срока возврата все кредитные операции делятся на несколько основных видов. Каждый из этих видов имеет свое название. Кредит может получить юридическое лицо (клиент), при регистрации предоставивший следующие сведения: название, вид собственности, адрес, телефон, контактное лицо. Каждый факт выдачи кредита регистрируется банком, при этом фиксируются сумма кредита, клиент и дата выдачи.

Таблицы

Виды кредитов (Код вида, Название, Условия получения, Ставка, Срок).
Клиенты (Код клиента, Название, Вид собственности, Адрес, Телефон, Контактное лицо).
Кредиты (Код вида, Код клиента, Сумма, Дата выдачи).

Развитие постановки задачи

Теперь ситуация изменилась. После проведения различных исследований выяснилось, что используемая система не позволяет отслеживать динамику возврата кредитов. Для устранения этого недостатка Вы приняли решение учитывать в системе еще и дату фактического возврата денег. Нужно еще учесть, что кредит может гаситься частями, и за задержку возврата кредита начисляются штрафы.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

21. Инвестирование свободных средств

Описание предметной области

Вы являетесь руководителем аналитического центра инвестиционной компании. Ваша компания занимается вложением денежных средств в ценные бумаги.

Ваши клиенты – предприятия, которые доверяют Вам управлять их свободными денежными средства на определенный период. Вам необходимо выбрать вид ценных бумаг, которые позволят получить прибыль и Вам и Вашему клиенту. При работе с клиентом для Вас весьма существенной является информация о предприятии – название, вид собственности, адрес и телефон.

Таблицы

Ценные бумаги (Код ценной бумаги, Минимальная сумма сделки, Рейтинг, Доходность за прошлый год, Дополнительная информация).
Инвестиции (Код инвестиции, Код ценной бумаги, Код клиента, Котировка, Дата покупки, Дата продажи).
Клиенты (Код клиента, Название, Вид собственности, Адрес, Телефон).

Развитие постановки задачи

При эксплуатации базы данных стало понятно, что необходимо хранить историю котировок каждой ценной бумаги. Кроме того, помимо вложений в ценные бумаги, существует возможность вкладывать деньги в банковские депозиты.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

22. Занятость актеров театра

Описание предметной области

Вы являетесь коммерческим директором театра, и в Ваши обязанности входит вся организационно-финансовая работа, связанная с привлечением актеров и заключением контрактов.

Вы поставили дело следующим образом: каждый год театр осуществляет постановку различных спектаклей. Каждый спектакль имеет определенный бюджет. Для участия в конкретных постановках в определенных ролях Вы привлекаете актеров. С каждым из актеров Вы заключаете персональный контракт на определенную сумму. Каждый из актеров имеет некоторый стаж работы, некоторые из них удостоены различных наград и званий.

Таблицы

Актеры (Код актера, Фамилия, Имя, Отчество, Звание, Стаж).

Спектакли (Код спектакля, Название, Год постановки, Бюджет).

Занятость актеров в спектакле (Код актера, Код спектакля, Роль, Стоимость годового контракта).

Развитие постановки задачи

В результате эксплуатации базы данных выяснилось, что в рамках одного спектакля на одну и ту же роль привлекается несколько актеров. Контракт определяет базовую зарплату актера, а по итогам реально отыгранных спектаклей актеру назначается премия. Кроме того, в базе данных нужно хранить информацию за несколько лет.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

23. Платная поликлиника

Описание предметной области

Вы являетесь руководителем службы планирования платной поликлиники. Вашей задачей является отслеживание финансовых показателей работы поликлиники.

В поликлинике работают врачи различных специальностей, имеющие разную квалификацию. Каждый день в поликлинику обращаются больные. Все больные проходят обязательную регистрацию, при которой в базу данных заносятся стандартные анкетные данные (фамилия, имя, отчество, год рождения). Каждый больной может обращаться в поликлинику несколько раз, нуждаясь в

различной медицинской помощи. Все обращения больных фиксируются, при этом устанавливается диагноз, определяется стоимость лечения, запоминается дата обращения.

Таблицы

Врачи (Код врача, Фамилия, Имя, Отчество, Специальность, Категория).
Пациенты (Код пациента, Фамилия, Имя, Отчество, Год рождения).
Обращения (Код обращения, Код врача, Код пациента, Дата обращения, Диагноз, Стоимость лечения).

Развитие постановки задачи

В результате эксплуатации базы данных выяснилось, что при обращении в поликлинику пациент обследуется и проходит лечение у разных специалистов. Общая стоимость лечения зависит от стоимости тех консультаций и процедур, которые назначены пациенту. Кроме того, для определенных категорий граждан предусмотрены скидки.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

24. Анализ динамики показателей финансовой отчетности различных предприятий

Описание предметной области

Вы являетесь руководителем информационно-аналитического центра крупного холдинга. Вашей задачей является отслеживание динамики показателей для предприятий Вашего холдинга.

В структуру холдинга входят несколько предприятий. Каждое предприятие имеет стандартные характеристики (название, реквизиты, телефон, контактное лицо). Работа предприятия может быть оценена следующим образом: в начале каждого отчетного периода на основе финансовой отчетности вычисляется по неким формулам определенный набор показателей. Принять, что важность показателей характеризуется некоторыми числовыми константами. Значение каждого показателя измеряется в некоторой системе единиц.

Таблицы

Показатели (Код показателя, Название, Важность, Единица измерения).
Предприятия (Код предприятия, Название, Банковские реквизиты, Телефон, Контактное лицо).

Динамика показателей (Код показателя, Код предприятия, Дата, Значение).

Развитие постановки задачи

В результате эксплуатации базы данных выяснилось, что некоторые показатели считаются в рублях, некоторые в долларах, некоторые в евро. Для удобства работы с показателями нужно хранить изменения курсов валют относительно друг друга.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

25. Учет телекомпанией стоимости прошедшей в эфире рекламы

Описание предметной области

Вы являетесь руководителем коммерческой службы телевизионной компании. Вашей задачей является отслеживание расчетов, связанных с прохождением рекламы в телеэфире.

Работа построена следующим образом: заказчики просят поместить свою рекламу в определенной передаче в определенный день. Каждый рекламный ролик имеет определенную продолжительность. Для каждой организации-заказчика известны банковские реквизиты, телефон и контактное лицо для проведения переговоров. Передачи имеют определенный рейтинг. Стоимость минуты рекламы в каждой конкретной передаче известна (определяется коммерческой службой, исходя из рейтинга передачи и прочих соображений).

Таблицы

Передачи (Код передачи, Название, Рейтинг, Стоимость минуты).
Реклама (Код рекламы, Код передачи, Код заказчика, Дата, Длительность в минутах).
Заказчики (Код заказчика, Название, Банковские реквизиты, Телефон, Контактное лицо).

Развитие постановки задачи

В результате эксплуатации базы данных выяснилось, что необходимо также хранить информацию об агентах, заключивших договоры на рекламу. Зарплата рекламных агентов составляет некоторый процент от общей стоимости рекламы, прошедшей в эфире.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

26. Интернет-магазин

Описание предметной области

Вы являетесь сотрудником коммерческого отдела компании, продающей различные товары через Интернет. Вашей задачей является отслеживание финансовой составляющей работы компании.

Работа Вашей компании организована следующим образом: на Интернет-сайте компании представлены (выставлены на продажу) некоторые товары. Каждый из них имеет некоторое название, цену и единицу измерения (штуки, килограммы, литры). Для проведения исследований и оптимизации работы магазина Вы пытаетесь собирать данные с Ваших клиентов. При этом для Вас определяющее значение имеют стандартные анкетные данные, а также телефон и адрес электронной почты для связи. В случае приобретения товаров на сумму свыше 5000р. клиент переходит в категорию «постоянных клиентов» и получает скидку на каждую покупку в размере 2%. По каждому факту продажи Вы автоматически фиксируете клиента, товары, количество, дату продажи, дату доставки.

Таблицы

Товары (Код товара, Название, Цена, Единица измерения).
Клиенты (Код клиента, Фамилия, Имя, Отчество, Адрес, Телефон, e-mail, Признак постоянного клиента).
Продажи (Код продажи, Код товара, Код клиента, Дата продажи, Дата доставки, Количество).

Развитие постановки задачи

В результате эксплуатации базы данных выяснилось, что иногда возникают проблемы, связанные с нехваткой информации о наличии нужных товаров на складе в нужном количестве. Кроме того, обычно клиенты в рамках одного заказа покупают не один вид товара, а несколько видов. Исходя из суммарной стоимости заказа, компания предоставляет дополнительные скидки.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

27. Ювелирная мастерская

Описание предметной области

Вы работаете в ювелирной мастерской. Ваша мастерская осуществляет изготовление ювелирных изделий для частных лиц на заказ. Вы работаете с определенными материалами (платина, золото, серебро, различные драгоценные камни и т.д.). При обращении к Вам потенциального клиента Вы определяетесь с тем, какое именно изделие ему необходимо. Все изготавливаемые Вами изделия принадлежат к некоторому типу (серьги, кольца, броши, браслеты), бывают выполнены из определенного материала, имеют некоторый вес и цену (включающую стоимость материалов и работы).

Таблицы

Изделия (Код изделия, Название, Тип, Код материала, Вес, Цена).
Материалы (Код материала, Название, Цена за грамм).
Продажи (Код изделия, Дата продажи, Фамилия покупателя, Имя покупателя, Отчество покупателя).

Развитие постановки задачи

В процессе опытной эксплуатации базы данных выяснилось, что ювелирное изделие может состоять из нескольких материалов. Кроме того, постоянным клиентам мастерская предоставляет скидки.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

28. Парикмахерская

Описание предметной области

Вы работаете в парикмахерской.

Ваша парикмахерская стрижет клиентов в соответствии с их пожеланиями и некоторым каталогом различных видов стрижки. Так, для каждой стрижки определены название, принадлежность полу (мужская, женская), стоимость работы. Для наведения порядка Вы, по мере возможности, составляете базу данных клиентов, запоминая их анкетные данные (фамилия, имя, отчество). Начиная с 5-ой стрижки, клиент переходит в категорию постоянных и получает скидку в 3% при каждой последующей стрижке. После того, как закончена очередная работа, в кассе фиксируются стрижка, клиент и дата производства работ.

Таблицы

Стрижки (Код стрижки, Название, Пол, Стоимость).
Клиенты (Код клиента, Фамилия, Имя, Отчество, Пол, Признак постоянного клиента).
Работа (Код работы, Код стрижки, Код клиента, Дата).

Развитие постановки задачи

Теперь ситуация изменилась. У Вашей парикмахерской появился филиал, и Вы хотели бы видеть, в том числе, и раздельную статистику по филиалам. Кроме того, стоимость стрижки может меняться с течением времени. Нужно хранить не только последнюю цену, но и все данные по изменению цены стрижки.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

29. Химчистка

Описание предметной области

Вы работаете в химчистке.

Ваша химчистка осуществляет прием у населения вещей для выведения пятен. Для наведения порядка Вы, по мере возможности, составляете базу данных клиентов, запоминая их анкетные данные (фамилия, имя, отчество). Начиная с 3-го обращения, клиент переходит в категорию постоянных клиентов и получает скидку в 3% при чистке каждой последующей вещи. Все оказываемые Вами услуги подразделяются на виды, имеющие название, тип и стоимость, зависящую от сложности работ. Работа с клиентом первоначально состоит в определении объема работ, вида услуги и, соответственно, ее стоимости. Если клиент согласен, он оставляет вещь (при этом фиксируется услуга, клиент и дата приема) и забирает ее после обработки (при этом фиксируется дата возврата).

Таблицы

Виды услуг(Код вида услуг, Название, Тип, Стоимость).
Клиенты (Код клиента, Фамилия, Имя, Отчество, Признак постоянного клиента).
Услуги (Код услуги, Код вида услуги, Код клиента, Дата приема, Дата возврата).

Развитие постановки задачи

Теперь ситуация изменилась. У Вашей химчистки появился филиал, и Вы хотели бы видеть, в том числе, и раздельную статистику по филиалам. Кроме того, вы решили делать надбавки за срочность и сложность работ.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

30. Сдача в аренду торговых площадей

Описание предметной области

Вы работаете в крупном торговом центре, сдающим в аренду коммерсантам свои торговые площади.

Вашей задачей является наведение порядка в финансовой стороне работы торгового центра.

Работы Вашего торгового центра построена следующим образом: в результате планирования Вы определили некоторое количество торговых точек в пределах Вашего здания, которые могут сдаваться в аренду. Для каждой из торговых точек важными данными являются этаж, площадь, наличие кондиционера и стоимость аренды в день. Со всех потенциальных клиентов Вы собираете стандартные данные (название, адрес, телефон, реквизиты, контактное лицо). При появлении потенциального клиента Вы показываете ему имеющиеся свободные площади. При достижении соглашения Вы оформляете договор, фиксируя в базе данных торговую точку, клиента, период (срок) аренды.

Таблицы

Торговые точки (Код торговой точки, Этаж, Площадь, Наличие кондиционера, Стоимость аренды в день).
Клиенты (Код клиента, Название, Реквизиты, Адрес, Телефон, Контактное лицо).
Аренда (Код аренды, Код торговой точки, Код клиента, Дата начала, Дата окончания).

Развитие постановки задачи

В результате эксплуатации базы данных выяснилось, что некоторые клиенты арендуют сразу несколько торговых точек. Помимо этого, Вам необходимо собирать информацию об ежемесячных платежах, поступающих Вам от арендаторов.

Внести в структуру таблиц изменения, учитывающие эти факты, и изменить существующие запросы. Добавить новые запросы.

ПРИМЕР ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа №1

Краткое задание

Дано словесное описание простой предметной области.

В лабораторной работе рассматривается задача о зачислении абитуриентов на бюджетные места в некоторый вуз Нижнего Новгорода. Абитуриенты сдают экзамены на один или несколько факультетов вуза. Известно расписание экзаменов: дата, предмет экзамена, факультет, на который экзамен сдается.. На экзаменах абитуриенты получают оценки. По каждому абитуриенту хранятся некоторые данные, в частности, номер и дата выдачи аттестата.

Требуется построить различные варианты инфологической модели данных (представления данных), сравнить предложенные варианты.

Пример выполнения

Рассмотрим несколько вариантов инфологической модели.

Вариант 1.

Представим всю информацию как характеристики одного объекта – экзаменационной оценки.

ОЦЕНКА
Предмет экзамена Дата экзамена Факультет Фамилия Имя Отчество Номер аттестата Дата выдачи аттестата Значение оценки

Видим, что информация об абитуриенте дублируется, т.е. при внесении информации о новой оценке мы должны заново вносить уже введенную ранее информацию по абитуриенту (Фамилия, Имя, Отчество, номер аттестата, дата выдачи аттестата). При вводе одной и той же информации, можно допустить ошибки. Соответственно база данных перейдет в противоречивое состояние.

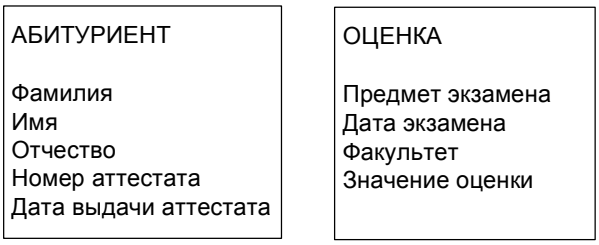
Предположим, что Сергеев Сергей Петрович сдал экзамен по математике на ВМК на оценку 5. Мы внесли информацию об этом (Сергеев, Сергей, Петрович, математика, ВМК, 5, № аттестата – 123123, дата выдачи аттестата – 21 июня 2003 года) в нашу базу данных. Через некоторое время данный абитуриент сдает информатику. Мы вносим информацию (Сергеев, Сергей, Петрович, информатика, ВМК, 5, № аттестата 123123, дата выдачи аттестата 22 июня 2004 года). При вводе была допущена ошибка – мы неправильно ввели дату выдачи аттестата.

Таким образом, наша база данных дает противоречивую информацию. По одним данным аттестат был получен Сергеевым 21 июня, по другим, 22 июня.

Кроме того, даже если информация была бы введена правильно, мы увеличиваем объем информации, что приводит к необходимости увеличения ресурсов и замедлению работы программного обеспечения.

Постараемся избавиться от данного недостатка и построим другую инфологическую модель.

Вариант 2.



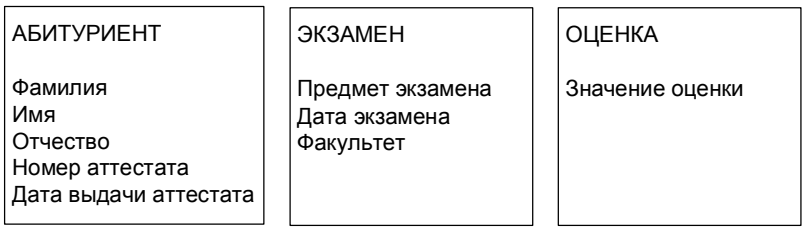
Между двумя сущностями должна существовать связь (абитуриент получает оценки).

АБИТУРИЕНТ <Получает> ОЦЕНКА

Для каждой полученной каждым абитуриентом оценки дублируется информация о предмете экзамена, дате экзамена, факультете.

Попробуем избавиться и от этого недостатка.

Вариант 3.



Между тремя сущностями существует две связи:

- АБИТУРИЕНТ <Получает> ОЦЕНКА;
- АБИТУРИЕНТ <Сдает> ЭКЗАМЕН.

В данной модели данных нет недостатков отмеченных в предыдущих двух моделях. Возьмем этот вариант модели за основной вариант при построении сетевой, иерархической и реляционной моделей.

Сетевая модель

Напомним, что для отображения связи в сетевой модели используется дополнительный файл (группа) связей (см. п.3.2 учебного пособия).

Общий вид сетевой модели представлен на рис. 1.

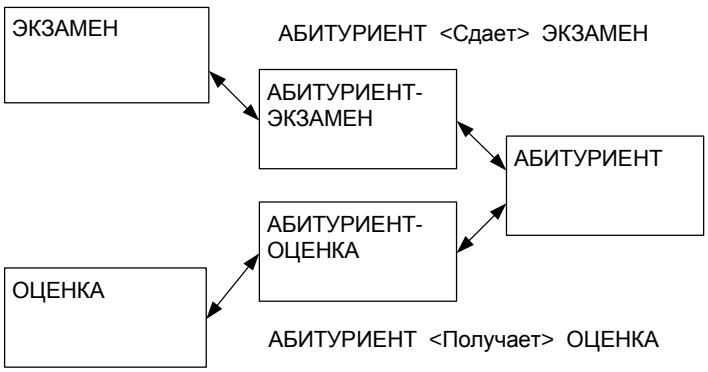


Рис.1. Общий вид сетевой модели

Представление связей между записями вышеуказанных групп рассмотрим на небольшом конкретном примере. Пусть абитуриент Сергеев сдал на факультет ВМК экзамены по математике и информатике и получил оценки 5 и 4. Абитуриент Смирнов сдал на факультет ВМК экзамены по математике и русскому языку и получил оценки 4 и 3. Тогда представление экземпляров записей и связей между ними будет иметь вид, представленный на рис. 2.

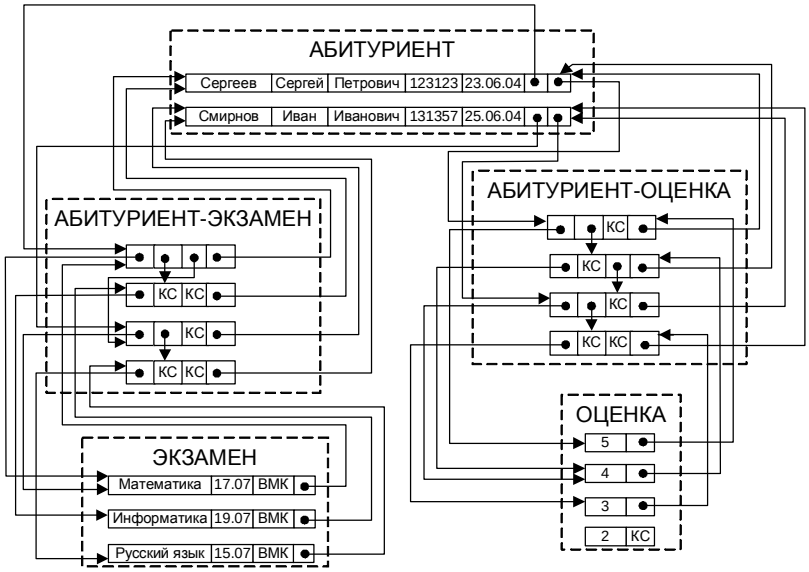


Рис.2. Сетевая модель данных

Поясним смысл полей записей дополнительных файлов на примере файла АБИТУРИЕНТ–ЭКЗАМЕН.

УК1	УК2	УК3	УК4

Указатель 1 (УК1) содержит адрес экземпляра записи файла ЭКЗАМЕН, т.е. на сдаваемый предмет. УК4 содержит адрес экземпляра записи файла АБИТУРИЕНТ, т.е. указывает на абитуриента, сдавшего этот экзамен. УК2 содержит адрес экземпляра записи дополнительного файла АБИТУРИЕНТ–ЭКЗАМЕН, который соответствует следующему экзамену, сдаваемому этим же абитуриентом. УК3 содержит адрес экземпляра записи дополнительного файла АБИТУРИЕНТ–ЭКЗАМЕН, который соответствует следующему абитуриенту, сдавшему этот экзамен. КС – признак конца соответствующего списка.

Аналогично определяются поля второго дополнительного файла АБИТУРИЕНТ–ОЦЕНКА. Двигаясь по указателям построенной модели, можно ответить на следующие запросы:

- «Какие экзамены сдал абитуриент Сергеев (Смирнов)?»
- «Какие абтуриенты сдали математику (информатику, русский язык)?»
- «Какие абитуриенты получили оценку 5 (4, 3)?»
- «Какие оценки получил абитуриент Сергеев (Смирнов)?»

Существенным недостатком построенной модели является невозможность ответа на запрос «По какому предмету абитуриент Сергеев (Смирнов) получил оценку 4?». Этот недостаток может быть устранен с помощью с помощью агрегации (см. п.2.7.2 учебного пособия).

Иерархическая модель

Общий вид иерархической модели представлен на рис. 3.

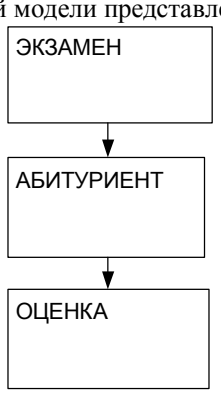


Рис.3. Общий вид иерархической модели

Представление экземпляров записей для вышеизложенного примера может, в частности, выглядеть следующим образом.

ЭКЗАМЕН			
Математика 17.07 ВМК		Информатика 19.07 ВМК	Русский язык 15.07 ВМК
АБИТУРИЕНТ			
Сергеев Сергей Петрович 123123 23.06.04	Смирнов Иван Иванович 131357 25.06.04	Сергеев Сергей Петрович 123123 23.06.04	Смирнов Иван Иванович 131357 05.06.04
ОЦЕНКА			
5	4	4	3

Примечание. Соответствующее дерево можно представить также с помощью указателей (по аналогии с сетевой моделью).

Реляционная модель

Реляционная модель представляется следующими отношениями:
Экзамены (Код экзамена, Предмет, Факультет, Дата).
Абитуриенты (Код абитуриента, Фамилия, Имя, Отчество, Номер аттестата, Дата выдачи аттестата).
Оценки (Код экзамена, Код абитуриента, Значение оценки).

Поля Код экзамена и Код абитуриента в таблице Оценки являются полями для реализации связи с соответствующими таблицами.

Лабораторная работа №2

Краткое задание

Даны три таблицы и генератор данных. Требуется в среде Access создать данные таблицы, наложив соответствующие условия и ограничения целостности, а также заполнить их данными.

Три таблицы, представляющие собой модель предметной области:
Абитуриенты (Номер абитуриента, Фамилия, Имя, Отчество, Номер аттестата, Дата выдачи аттестата).
Экзамены (Код экзамена, Предмет, Дата проведения экзамена, Тип экзамена, Факультет).
Оценки (Код экзамена, Номер абитуриента, Оценка).

Пример выполнения

Форматы полей и их свойства (приведены только те свойства полей, которые значимы и/или отличны от значений по умолчанию):

*Таблица **Абитуриенты***

ПОЛЕ	**НОМЕР**
Тип	Длинное целое
Размер	4
Примечание	Поле имеет специальный тип «Счетчик».
Новые значения	Последовательные
Примечание	Данное поле является ключом таблицы.
Индексированное поле	Да (совпадения не допускаются).
Поле	Фамилия
Тип	Текстовый;
Размер	20;
Обязательное поле	Да;
Пустые строки	Нет;
Индексированное поле	Нет.
Поле	Имя
Тип	Текстовый;
Размер	15;
Обязательное поле	Да;
Пустые строки	Нет;
Индексированное поле	Нет.
Поле	Отчество
Тип	Текстовый;
Размер	15;
Обязательное поле	Нет;
Индексированное поле	Нет.
Поле	Номер Аттестата.
Тип	Текстовый;
Размер	10;
Поле	Дата аттестата.
Тип	Дата/время;
Размер	8;
Формат поля	Краткий формат даты;

Обязательное поле	Нет.

Таблица ***Экзамены***

Поле	Код_экзамена
Тип	Длинное целое
Размер	4
Примечание	Поле имеет специальный тип «Счетчик»
Новые значения	Последовательные
Примечание	Данное поле является ключом таблицы
Индексированное поле	Да (совпадения не допускаются)
Поле	Предмет
Тип	Текстовый
Размер	30
Подстановка тип элемента управления	Поле со списком;
Тип источника строк	Список значений;
Источник строк	«математика»; «физика»; «русский язык и литература»; «информатика»;
Ограничиться списком	Да.
Поле	Дата_экзамена
Тип	Дата/время
Размер	8
Формат поля	Краткий формат даты
Обязательное поле	Да
Поле	Тип_экзамена
Тип	Текстовый(30)
Подстановка тип элемента управления	Поле со списком
Тип источника строк	Список значений
Источник строк	«письменно»; «устно»; «сочинение»
Ограничиться списком	Да
Поле	Факультет
Тип	Текстовый(5).
Подстановка тип элемента управления	Поле со списком
Тип источника строк	Список значений
Источник строк	«ВМК»; «ММ»; «ЭФ»; «ВШОПФ»

Ограничиться списком	Да

Таблица **Оценки**

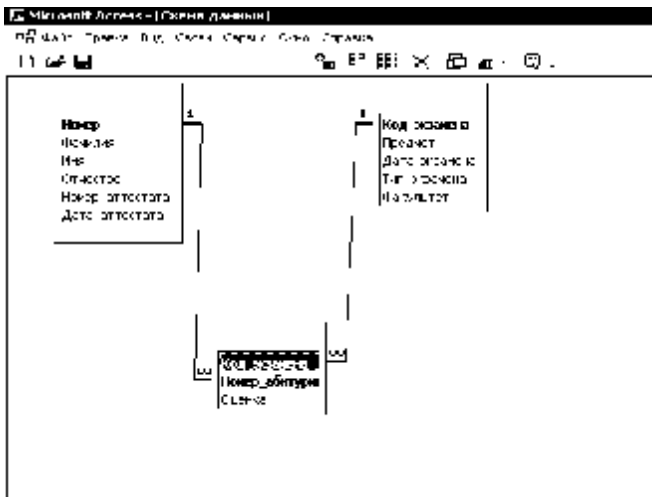
Поле	Код_экзамена
Тип	Длинное целое (4)
Обязательно поле	Да
Индексированное поле	Да (совпадения допускаются)
Поле	Номер_абитуриента
Тип	Длинное целое (4)
Индексированное поле	Да (совпадения допускаются)

Данные два поля составляют ключ таблицы.

Ключ реализован с помощью соответствующего индексного файла

Поле	Оценка
Тип	Числовой (одинарное с плавающей точкой)
Формат поля	Фиксированный
Число десятичных знаков	1
Условие на значение	>=2 And <=5
Сообщение об ошибке	Оценка должна быть между 2 и 5
Обязательное поле	Нет

Схема данных



Абитуриенты-Оценки

Номер 1–∞ Номер_абитуриента
Атрибуты: обеспечение целостности, каскадные обновления, каскадные удаления
Атрибуты: один-ко-многим

Экзамены-Оценки

Код_экзамена 1–∞ Код_экзамена
Атрибуты: обеспечение целостности, каскадные обновления, каскадные удаления
Атрибуты: один-ко-многим

Импорт данных

С помощью генератора данных генерируем три текстовых файла. Данные из файлов импортируем в Access. Получаем следующие три таблицы (приведены фрагменты таблиц).

Таблица «Абитуриенты»

Номер	Фамилия	Имя	Отчество	Номер_аттестата	Дата_аттестата
1	Сергеев	Сергей	Сергеевич	123123	21.06.2003
2	Иванов	Александр	Николаевич	321321	21.06.2003
3	Михайлов	Федор	Петрович	245542	21.06.2003

Таблица «Экзамены»

Код_экзамена	Предмет	Дата_экзамена	Тип_экзамена	Факультет
1	Математика	21.07.2003	письменно	ВМК
2	Физика	27.07.2003	письменно	ВМК
3	Математика	21.07.2003	письменно	ММ
4	Физика	27.07.2003	письменно	ММ
5	русский язык и литература	30.07.2003	сочинение	ВМК
6	русский язык и литература	30.07.2003	сочинение	ММ

Таблица «Оценки»

Код_экзамена	Номер_абитуриента	Оценка
1	1	4,50
1	2	4,00
1	3	3,50
2	1	4,00
2	2	4,50
3	1	4,00

Код_экзамена	Номер_абитуриента	Оценка
5	1	3,50
5	2	3,00

Лабораторная работа №3

Краткое задание

Выбрать требуемую информацию.

Пример выполнения

1. Вывести фамилию, имя, отчество всех абитуриентов.

```
SELECT фамилия, имя, отчество FROM абитуриенты;
```

2. Вывести названия различных предметов, которые сдаются в вуз.

```
SELECT DISTINCT предмет FROM экзамены;
```

3. Вывести всю возможную информацию об экзаменах, проводимых в вуз.

```
SELECT * FROM экзамены;
```

4. Вывести фамилии абитуриентов и оценки, которые они получили на различных экзаменах.

```
SELECT фамилия, оценка  FROM абитуриенты, оценки
  WHERE абитуриенты.номер =оценки.номер_абитуриента;
```

5. Вывести фамилии абитуриентов и их оценки по математике. Отсортировать вывод по оценкам, внутри оценок по фамилиям абитуриентов.

```
SELECT фамилия, оценка, предмет
  FROM абитуриенты, экзамены, оценки
  WHERE абитуриенты.номер = оценки.номер_абитуриента
    and оценки.код_экзамена = экзамены.код_экзамена
    and предмет = 'Математика'
  ORDER BY оценка desc, фамилия;
```

6. Вывести фамилии абитуриентов и оценки, которые были получены в июле 2003 года.

```
SELECT фамилия, оценка
  FROM абитуриенты, экзамены, оценки
  WHERE абитуриенты.номер = оценки.номер_абитуриента
```

```
    and оценки.код_экзамена = экзамены.код_экзамена
    and month(дата_экзамена) = 7 and year(дата_экзамена)
      = 2003;
```

7. Вывести фамилии, имена и отчества абитуриентов, сдавших математику лучше, чем на 3.

```
SELECT DISTINCT фамилия, имя, отчество
  FROM абитуриенты, экзамены, оценки
  WHERE абитуриенты.номер = оценки.номер_абитуриента
    and оценки.код_экзамена = экзамены.код_экзамена
    and предмет = 'математика'
    and оценка>3;
```

8. Какие оценки получил Сергеев Сергей Сергеевич на ВМК?

```
SELECT оценка
  FROM абитуриенты, экзамены, оценки
  WHERE абитуриенты.номер = оценки.номер_абитуриента
    and оценки.код_экзамена = экзамены.код_экзамена
    and фамилия = «Сергеев» and имя = «Сергей»
    and отчество = «Сергеевич»
    and факультет = «ВМК»;
```

9. Какие оценки получил абитуриент с кодом 2? а с кодом 3? а с некоторым кодом?

```
SELECT оценка FROM оценки
  WHERE номер_абитуриента = 2;
SELECT оценка FROM оценки
  WHERE номер_абитуриента = 3;
SELECT оценка FROM оценки
  WHERE номер_абитуриента = [введите номер абитуриента];
```

10. Вывести фамилию, инициалы и оценки абитуриентов за экзамен по математике на факультете ВМК, который проходил 21 июля. Фамилия и инициалы должны быть выведены в одном поле. Вывод должен быть отсортирован по вычислимому полю.

```
SELECT фамилия & « » & left(имя,1) & «. »
     & left( отчество,1) as абитуриент, оценка
FROM абитуриенты, экзамены, оценки
WHERE абитуриенты.номер = оценки.номер_абитуриента
    and оценки.код_экзамена = экзамены.код_экзамена
```

```
    and факультет = «ВМК»
    and предмет = «математика»
    and дата_экзамена = #7/21/2003#
ORDER BY фамилия & « » & left(имя,1) & «. »
         & left(отчество,1);
```

Внешнее объединение. Is Null, Is Not Null, Like. Оператор UNION.

11. Добавить абитуриента в таблицу. Пусть пока он не получил ни одной оценки. Вывести информацию по всем абитуриентам и оценки, которые они получили (или значение NULL, если они не получили пока ни одной оценки).

```
SELECT фамилия, оценка
  FROM абитуриенты
  LEFT JOIN оценки
  ON абитуриенты.номер = оценки.номер_абитуриента;
```

12. Вывести информацию (номер, фамилия, имя, отчество) обо всех абитуриентах, которые не имеют пока оценок.

```
SELECT фамилия, оценка
  FROM абитуриенты
  LEFT JOIN оценки
  ON абитуриенты.номер = оценки.номер_абитуриента
  WHERE оценка Is Null;
```

13. Вывести абитуриентов, которые сдавали какие-либо экзамены.

```
SELECT фамилия, оценка
  FROM абитуриенты LEFT JOIN оценки
  ON абитуриенты.номер=оценки.номер_абитуриента
  WHERE оценка is not null;
```

14. Вывести даты проведения экзаменов и даты выдачи аттестатов (?).

```
SELECT дата_аттестата FROM абитуриенты
UNION
SELECT дата_экзамена FROM экзамены;
```

15. Получить фамилии, имена и номера студентов, которых зовут Александр или Александра. Учесть, по возможности, что имя может быть набрано с ошибками или с ведущими пробелами (?).

```
SELECT фамилия,имя, номер
```

```
  FROM абитуриенты
  WHERE имя like «*Александр*» or  имя like «*Алекс*»
    or  имя like «*Ал*»
```

16. Какая средняя оценка по всем абитуриентам?

```
SELECT avg(оценка) as средняя_оценка FROM оценки;
```

17. Сколько экзаменов в расписании?

```
SELECT count(*) as количество_экзаменов FROM экзамены;
```

18. Сколько абитуриентов в нашей базе данных?

```
SELECT count(*) as количество_абитуриентов
  FROM абитуриенты;
```

19. Какая средняя оценка по математике на каждый факультет?

```
SELECT avg(оценка) as  средняя_оценка, факультет
  FROM оценки, экзамены, абитуриенты
  WHERE оценки.код_экзамена = экзамены.код_экзамена
    and оценки.номер_абитуриента = абитуриенты.номер
    and предмет = «математика»
  GROUP BY факультет;
```

20. Сколько оценок было поставлено на экзаменах на каждый факультет?

```
SELECT факультет, count(номер_абитуриента)
    as   количество_абитуриентов
  FROM экзамены, оценки
  WHERE оценки.код_экзамена = экзамены.код_экзамена
  GROUP BY факультет;
```

21. Какие средние оценки по каждому факультету по каждому предмету?

```
SELECT факультет, предмет, avg(оценка) as средняя_оценка
  FROM экзамены, оценки
  WHERE оценки.код_экзамена = экзамены.код_экзамена
  GROUP BY факультет, предмет;
```

Создать группирующий запрос, в котором определяются условия, причем сначала выполняются вычисления, а затем происходит отбор.

22. На какие факультеты средний балл абитуриентов по всем экзаменам выше, чем 4?

```
SELECT avg(оценка), факультет FROM экзамены, оценки
  WHERE оценки.код_экзамена = экзамены.код_экзамена
  GROUP BY факультет
  HAVING avg(оценка) > 4;
```

23. По каким предметам средний балл абитуриентов меньше 4?

```
SELECT avg(оценка), предмет FROM экзамены, оценки
  WHERE оценки.код_экзамена = экзамены.код_экзамена
  GROUP BY предмет
  HAVING avg(оценка) < 4;
```

Создать группирующий запрос, в котором определяются условия, причем сначала происходит отбор, а затем выполняются вычисления.

24. Среди абитуриентов, сдававших математику, подсчитать количество положительных оценок по каждому факультету.

```
SELECT count(оценка) as количество_оценок, факультет
  FROM оценки, экзамены
  WHERE оценки.код_экзамена = экзамены.код_экзамена
    and предмет = «математика»
    and оценка > 2
  GROUP BY факультет;
```

25. Подсчитать среднюю оценку абитуриентов по каждому факультету, за июльские экзамены.

```
SELECT avg(оценка) as средняя_оценка, факультет
  FROM оценки, экзамены
  WHERE оценки.код_экзамена = экзамены.код_экзамена
     and month(дата_экзамена) = 7
  GROUP BY факультет;
```

26. Какие баллы набрали абитуриенты на каждый факультет?

```
SELECT sum(оценка) as набранный_балл, факультет,
       номер_абитуриента
  FROM оценки, экзамены
  WHERE оценки.код_экзамена = экзамены.код_экзамена
  GROUP BY факультет, номер_абитуриента;
```

27. Вывести информацию по абитуриентам, получившим пятерки по математике или физике.

```
SELECT фамилия & « » & left(имя,1) & «. »
     & left( отчество,1) as абитуриент, оценка, предмет
  FROM абитуриенты, экзамены, оценки
  WHERE абитуриенты.номер = оценки.номер_абитуриента
     and оценки.код_экзамена = экзамены.код_экзамена
     and предмет in ( «математика»,»физика»)
  ORDER BY фамилия & « » & left(имя,1) & «. »
     & left(отчество,1);
```

28. Какие студенты сдали экзамены (получили оценку строго больше 2) с кодом 1 или 2 или 3?

```
SELECT DISTINCT фамилия, имя, отчество
  FROM абитуриенты, экзамены, оценки
  WHERE абитуриенты.номер = оценки.номер_абитуриента
    and оценки.код_экзамена = экзамены.код_экзамена
    and оценки.код_экзамена in (1,2,3)
    and оценка>2
  ORDER BY фамилия, имя;
```

29. Какие абитуриенты получили больше других пятерок?

```
SELECT номер_абитуриента, count(оценка)
  FROM  оценки
  WHERE оценка = 5
  GROUP BY номер_абитуриента
  HAVING count(оценка) > ANY
   (SELECT count(оценка)
     FROM оценки
     WHERE оценка = 5
     GROUP BY номер_абитуриента);
```

30. Какие абитуриенты сдали математику лучше, чем в среднем сдали различные предметы абитуриенты, поступающие на ВМК?

```
SELECT DISTINCT номер_абитуриента
  FROM  оценки, экзамены
  WHERE оценки.код_экзамена = экзамены.код_экзамена
    and предмет = «математика»
    and оценка >
```

```
    (SELECT avg(оценка)
      FROM оценки, экзамены
      WHERE оценки.код_экзамена = экзамены.код_экзамена
           and факультет = «ВМК»);
```

31. По каким предметам проводилось меньше экзаменов, чем по другим?

```
SELECT предмет
  FROM экзамены
  GROUP BY предмет
  HAVING count(*) < ANY
   (SELECT count(*)
      FROM экзамены
      GROUP BY предмет);
```

32. Какие абитуриенты сдали все экзамены, которые есть в расписании?

```
SELECT номер_абитуриента
  FROM оценки
  GROUP BY номер_абитуриента
  HAVING count(*) =
   (SELECT count(*) FROM Экзамены);
```

33. Найти экзамены, которые пока никто не сдавал.

```
SELECT код_экзамена FROM экзамены
  WHERE not  exists
    (SELECT * FROM Оценки
      WHERE код_экзамена = экзамены.код_экзамена);
```

34. Найти абитуриентов, которые получили только «отлично» при поступлении на более чем один факультет?

```
SELECT DISTINCT  номер_абитуриента, факультет
  FROM абитуриенты, оценки t1, экзамены  t2
  WHERE абитуриенты.номер = t1.номер_абитуриента
      and оценка = 5
      and t2.код_экзамена = t1.код_экзамена
      and EXISTS(
       SELECT *
       FROM Оценки, экзамены
       WHERE оценки.код_экзамена = экзамены.код_экзамена
```

```
and номер_абитуриента = t1.номер_абитуриента
and факультет <>  t2.факультет);
```

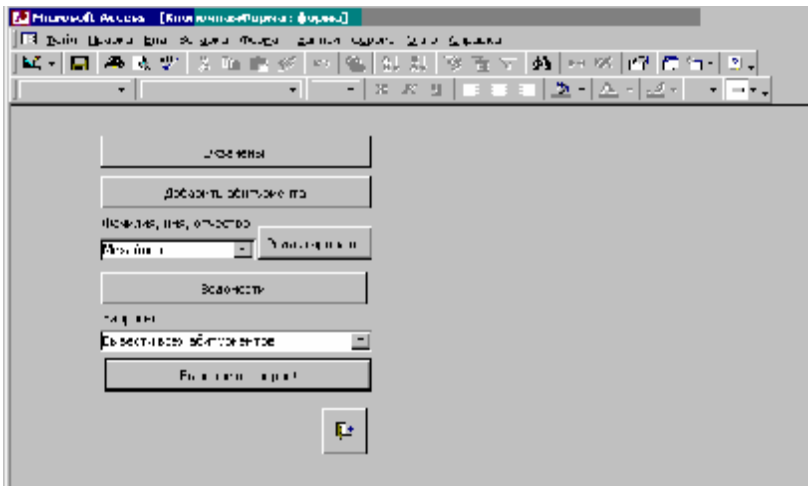
Лабораторная работа №4

Краткое задание

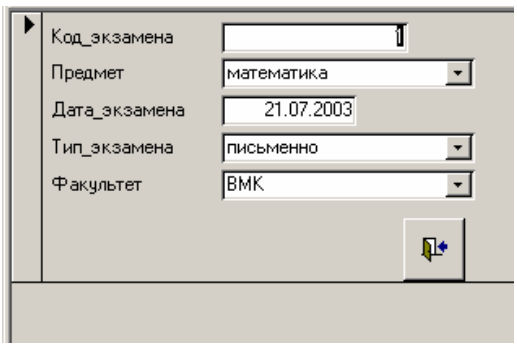
Построить интерфейс для созданной базы данных.

Пример выполнения

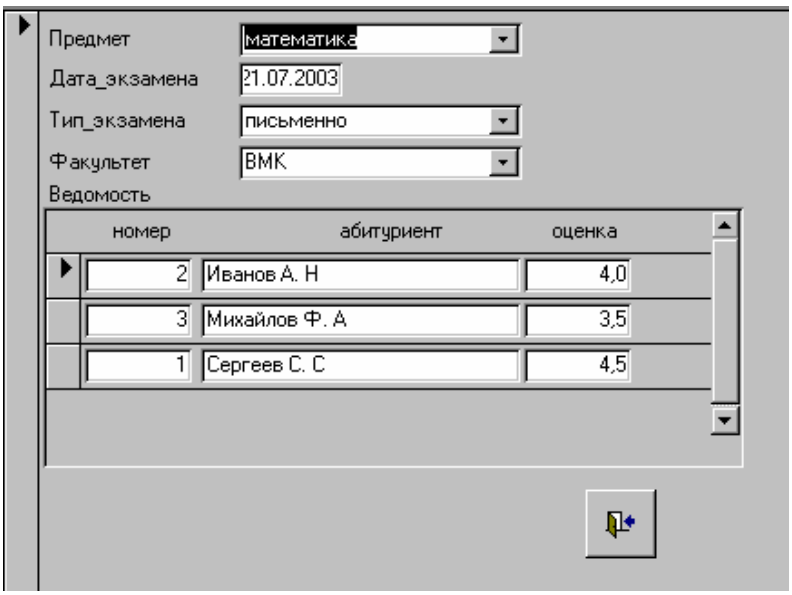
Кнопочная форма – главное меню.



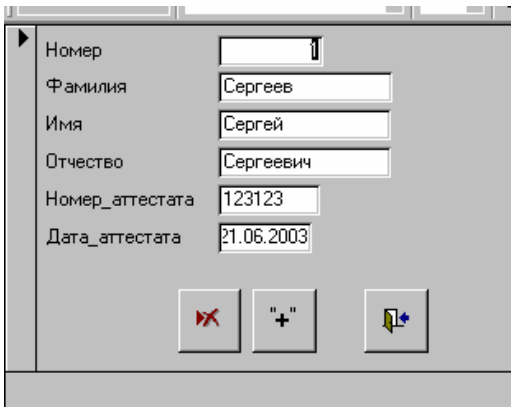
Форма редактирования и добавления экзаменов.



Форма просмотра экзаменационных ведомостей.



Форма просмотра информации об абитуриенте и добавления абитуриентов.



При выполнении данной лабораторной работы были написаны некоторые алгоритмы на языке VBA.

Лабораторная работа №5

Краткое задание

Развить предметную область. Построить диаграммы «Сущность-Связь». Привести таблицы к третьей нормальной форме.

Пример выполнения

Описание расширенной предметной области

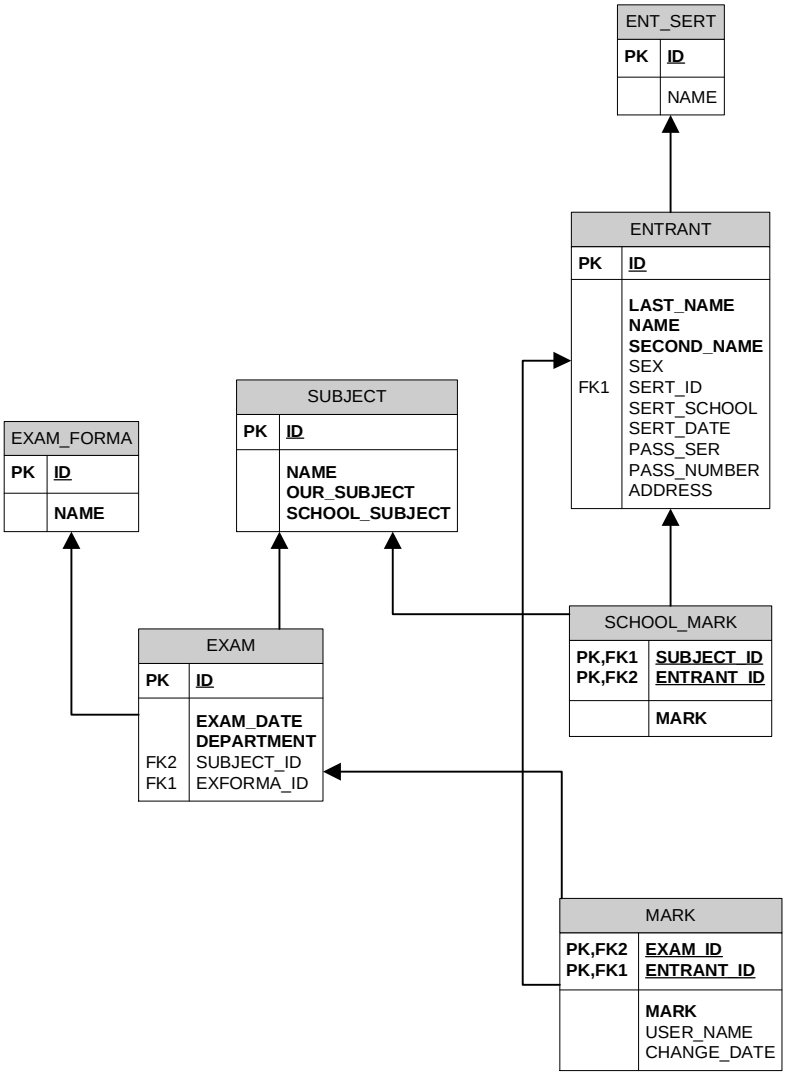
Вуз проводит прием граждан на 1 курс. Желающие поступить (в дальнейшем, абитуриенты) подают документы и сдают экзамены. По результатам экзаменов, абитуриенты, набравшие наибольшее количество баллов, зачисляются в вуз. Абитуриенты могут одновременно подать документы на несколько факультетов, но в этом случае, они должны сдавать экзамены на каждый из выбранных факультетов. Абитуриенты сдают в приемную комиссию аттестат о среднем полном образовании или диплом техникума (колледжа) о среднем специальном образовании. Каждому абитуриенту присваивается номер, который заносится в его личное дело, зачетную книжку и другие документы.

При внесении информации об оценках необходимо хранить имя пользователя, вносящего информацию и дату (время) внесения.

Состав хранимой информации

Необходимо хранить и обрабатывать следующие данные. Номер, присвоенный абитуриенту при поступлении, фамилию, имя, отчество, пол абитуриента. Документ об образовании (школьный аттестат, диплом об окончании техникума или ПТУ). Номер документа, кем и когда выдан документ об образовании. Паспортные данные, адрес проживания. Оценки по школьным предметам из документа об образовании. Расписание экзаменов - факультет, предмет, тип экзамена, дата экзамена. Тип экзамена - устный, письменный, сочинение, тест. Данные об оценках абитуриента на каждый факультет, на который он поступает (экзамен, оценка).

Диаграммы «Сущность-Связь»



Таблицы в ЗНФ

Абитуриенты (Номер, Имя, Отчество, Фамилия, Пол, Код документа об образовании, Выдан школой (учреждением), Дата выдачи, Серия паспорта, Номер паспорта, Адрес проживания).
Entrant (Id, Name, Second_name, Last_name, Sex, Sert_id, Sert_school, Sert_date, Pass_ser, Pass_number, Address).

Документы об образовании (Код, Название).
Ent_sert (Id, Name).

Предметы (Код, Название, Школьный предмет, Вузовский предмет).
Subject (Id, Name, Our_subject, School_subject).

Школьные оценки (Код предмета, Код абитуриента, Оценка).
School_mark (Subject_id, Entrant_id, mark).

Экзаменационные оценки (Код экзамена, Код абитуриента, Оценка, Имя пользователя, Дата внесений изменений).
Mark (Exam_id, Entrant_id, Mark, User_name, Change_date).

Экзамены (Код экзамена, Предмет, Дата проведения экзамена, Тип экзамена, Факультет).
Exam (Id, Subj_id, Exforma_id, Exam_date, Department).

Типы экзамена (Код, Название).
Exam_forma (Id, Name).

Скрипты для создания объектов базы данных в СУБД Oracle

```
CREATE TABLE «STUDENT».«ENTRANT» (
  «ID» NUMBER(10) NOT NULL,
  «LAST_NAME» VARCHAR2(20 byte) NOT NULL,
  «NAME» VARCHAR2(15 byte) NOT NULL,
  «SECOND_NAME» VARCHAR2(15 byte) NOT NULL,
  «SEX» VARCHAR2(1 byte) DEFAULT 'м',
  «SERT_ID» NUMBER(10) DEFAULT 1,
  «SERT_SCHOOL» NUMBER(7), «SERT_DATE» DATE,
  «PASS_SER» VARCHAR2(4 byte),
  «PASS_NUMBER» VARCHAR2(6 byte),
  «ADDRESS» VARCHAR2(70 byte),
  CONSTRAINT «ST_ENT_PK» PRIMARY KEY(«ID»)
  USING INDEX
```

```
  TABLESPACE «USERS»
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0) PCTFREE 10 INITRANS
    2 MAXTRANS 255)
  TABLESPACE «USERS» PCTFREE 10 PCTUSED 0 INITRANS 1
    MAXTRANS  255
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0)
  LOGGING

CREATE TABLE «STUDENT».«ENT_SERT» (
  «ID» NUMBER(10) NOT NULL,
  «NAME» VARCHAR2(100 byte),
  CONSTRAINT «ENT_SERT_PK» PRIMARY KEY(«ID»)
  USING INDEX
  TABLESPACE «USERS»
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0) PCTFREE 10 INITRANS 2
    MAXTRANS 255)
  TABLESPACE «USERS» PCTFREE 10 PCTUSED 0 INITRANS 1
    MAXTRANS 255
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0)
  LOGGING

CREATE TABLE «STUDENT». «EXAM» («ID» NUMBER(10) NOT NULL,
  «EXAM_DATE» DATE NOT NULL,
  «DEPARTMENT» VARCHAR2(5 byte) NOT NULL,
  «SUBJECT_ID» NUMBER(10), «EXFORMA_ID» NUMBER(10),
  CONSTRAINT «EXAM_DEP_CHECK»
  CHECK(department in ('BMK','MM')),
  CONSTRAINT «EXAM_EXFORMA_FK» FOREIGN KEY(«EXFORMA_ID»)
  REFERENCES «STUDENT».«EXAM_FORMA»(«ID»),
  CONSTRAINT «EXAM_PK» PRIMARY KEY(«ID»),
  CONSTRAINT «EXAM_SUBJ_FK» FOREIGN KEY(«SUBJECT_ID»)
  REFERENCES «STUDENT».«SUBJECT»(«ID»))
  TABLESPACE «USERS» PCTFREE 10 PCTUSED 0 INITRANS 1
     MAXTRANS 255
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
```

```
    2147483645 PCTINCREASE 0)
  LOGGING

CREATE TABLE «STUDENT».«EXAM_FORMA» (
  «ID» NUMBER(10) NOT NULL,
  «NAME» VARCHAR2(50 byte) NOT NULL,
  CONSTRAINT «EXAM_FORMA_PK» PRIMARY KEY(«ID»)
  USING INDEX
  TABLESPACE «USERS»
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0) PCTFREE 10 INITRANS 2
    MAXTRANS 255)
  TABLESPACE «USERS» PCTFREE 10 PCTUSED 0 INITRANS 1
    MAXTRANS 255
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0)
  LOGGING

CREATE TABLE «STUDENT».«MARK» (
  «EXAM_ID» NUMBER(10) NOT NULL,
  «ENTRANT_ID» NUMBER(10) NOT NULL,
  «MARK» NUMBER(5, 2) NOT NULL,
  «USER_NAME» VARCHAR2(50 byte),
  «CHANGE_DATE» DATE,
  CONSTRAINT «MARK_PK»   PRIMARY KEY(«EXAM_ID»,
    «ENTRANT_ID»),
  CONSTRAINT «ST_ENT_FK» FOREIGN KEY(«ENTRANT_ID»)
  REFERENCES «STUDENT».«ENTRANT»(«ID»),
  CONSTRAINT «ST_EXAM_FK» FOREIGN KEY(«EXAM_ID»)
  REFERENCES «STUDENT».«EXAM»(«ID»))
  TABLESPACE «USERS» PCTFREE 10 PCTUSED 0 INITRANS 1
    MAXTRANS  255
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0)
  LOGGING

CREATE TABLE «STUDENT». «SCHOOL_MARK» (
  «SUBJECT_ID» NUMBER(10) NOT NULL,
  «ENTRANT_ID» NUMBER(10) NOT NULL,
```

```
  «MARK» NUMBER(3, 1) NOT NULL,
  CONSTRAINT «SCHOOL_MARK_PK»   PRIMARY KEY(«SUBJECT_ID»,
    «ENTRANT_ID»)
  USING INDEX
  TABLESPACE «USERS»
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0) PCTFREE 10 INITRANS 2
    MAXTRANS 255,
  CONSTRAINT «SMAKR_SUBJ_FK» FOREIGN KEY(«SUBJECT_ID»)
  REFERENCES «STUDENT».«SUBJECT»(«ID»),
  CONSTRAINT «SMARK_ENT_FK» FOREIGN KEY(«ENTRANT_ID»)
  REFERENCES «STUDENT».«ENTRANT»(«ID»))
  TABLESPACE «USERS» PCTFREE 10 PCTUSED 0 INITRANS 1
     MAXTRANS   255
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0)
  LOGGING

CREATE TABLE «STUDENT».«SUBJECT» (
  «ID» NUMBER(10) NOT NULL,
  «NAME» VARCHAR2(40 byte) NOT NULL,
  «OUR_SUBJECT» NUMBER(1) DEFAULT 0 NOT NULL,
  «SCHOOL_SUBJECT» NUMBER(1) DEFAULT 0 NOT NULL,
  CONSTRAINT «SUBJECT_PK» PRIMARY KEY(«ID»)
  USING INDEX
  TABLESPACE «USERS»
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0) PCTFREE 10 INITRANS 2
    MAXTRANS 255)
  TABLESPACE «USERS» PCTFREE 10 PCTUSED 0 INITRANS 1
    MAXTRANS 255
  STORAGE ( INITIAL 64K NEXT 0K MINEXTENTS 1 MAXEXTENTS
    2147483645 PCTINCREASE 0)
  LOGGING
```

Текст хранимой процедуры (триггера), отвечающего за корректное заполнение полей дата (время) внесения (изменения) оценки, имя пользователя, вносящего изменения.

```
CREATE OR REPLACE TRIGGER «STUDENT».»MARK_TRG» BEFORE
```

```
INSERT
OR UPDATE OF «MARK» ON «MARK» FOR EACH ROW begin
  :new.user_name := USER;
  :new.change_date := SYSDATE;
end;
```

Лабораторная работа №6

Краткое задание

Сформулировать вопросы к расширенной предметной области. Создать соответствующие запросы, привести их текст на языке SQL, а также некоторые результаты работы.

Пример выполнения

1. Вывести фамилию, имя, отчество всех абитуриентов.

```
SELECT last_name, name, second_name
  FROM entrant;
```

2. Вывести названия различных предметов, которые сдаются в вуз.

```
SELECT distinct name FROM subject, exam
  WHERE subject.id = exam.subject_id
```

3. Вывести всю возможную информацию об экзаменах, проводимых в вуз.

```
SELECT exam.id, subject.name, department, exam_date,
     exam_forma.name
  FROM exam, subject, exam_forma
  WHERE exam.subject_id = subject.id
    and exam.exforma_id = exam_forma.id;
```

4. Вывести фамилии абитуриентов и оценки, которые они получили на различных экзаменах.

```
SELECT last_name, mark, exam_id
  FROM entrant, mark
  WHERE entrant.id = mark.entrant_id
  ORDER BY last_name;
```

5. Вывести фамилии абитуриентов и их оценки по математике. Отсортировать вывод по оценкам, внутри оценок по фамилиям абитуриентов.

```
SELECT last_name, mark
  FROM entrant, mark, exam, subject
  WHERE entrant.id = mark.entrant_id
      and mark.exam_id = exam.id
      and exam.subject_id = subject.id
      and subject.name = 'Математика'
  ORDER BY mark, last_name
```

6. Вывести фамилии, имена и отчества абитуриентов, сдавших математику лучше, чем на 3.

```
SELECT last_name, entrant.name, second_name, mark
  FROM entrant, mark, exam, subject
  WHERE entrant.id = mark.entrant_id
     and mark.exam_id = exam.id
     and exam.subject_id = subject.id
     and subject.name = 'Математика' and mark >3;
```

7. Какие оценки получил Сергеев Сергей Сергеевич на ВМК?

```
SELECT mark
  FROM entrant, mark, exam
  WHERE entrant.id = mark.entrant_id
       and mark.exam_id = exam.id and department = 'ВМК'
       and name = 'Сергей' and second_name = 'Сергеевич'
       and last_name = 'Сергеев';
```

8. Вывести фамилию, инициалы и оценки абитуриентов за экзамен по математике на факультете ВМК. Фамилия и инициалы должны быть выведены в одном поле. Вывод должен быть отсортирован по вычислимому полю.

```
SELECT last_name || ' ' || entrant.name || ' ' ||
     second_name as fio
  FROM entrant, mark, exam, subject
  WHERE entrant.id = mark.entrant_id
     and mark.exam_id = exam.id
     and subject.id = exam.subject_id
     and department = 'ВМК'
     and subject.name = 'Математика'
  ORDER BY fio;
```

9. Добавить абитуриента в таблицу. Пусть пока он не получил ни одной оценки. Вывести информацию по всем абитуриентам и оценки, которые они получили (или значение NULL, если они не получили пока ни одной оценки).

```
SELECT last_name, mark
  FROM entrant, mark
  WHERE entrant.id = mark.entrant_id (+) ;
```

10. Вывести информацию (номер, фамилия, имя, отчество) обо всех абитуриентах, которые не имеют пока оценок.

```
SELECT last_name, mark
  FROM entrant, mark
  WHERE entrant.id = mark.entrant_id (+)
     and mark Is Null;
```

11. Вывести абитуриентов, которые сдавали какие-либо экзамены.

```
SELECT last_name, mark
  FROM entrant, mark
  WHERE entrant.id = mark.entrant_id (+)
      and mark Is Not Null;
```

12. Получить фамилии, имена и номера студентов, которых зовут Александр или Александра. Учесть, по возможности, что имя может быть набрано с ошибками или с ведущими пробелами.

```
SELECT last_name
  FROM entrant
  WHERE name like '%Алекс%';
```

13. Какая средняя оценка по всем абитуриентам?

```
SELECT avg(mark) as avg_mark
  FROM mark;
```

14. Сколько экзаменов в расписании?

```
SELECT count(id) FROM exam;
```

15. Сколько абитуриентов в нашей базе данных?

```
SELECT count(*) FROM entrant;
```

16. Какая средняя оценка по математике на каждый факультет?

```
SELECT avg(оценка) as  средняя_оценка, факультет
SELECT avg(mark), department
  FROM mark, exam, subject
  WHERE mark.exam_id = exam.id
     and exam.subject_id = subject.id
     and subject.name = 'Математика'
  GROUP BY department;
```

17. Какие средние оценки по каждому факультету по каждому предмету?

```
SELECT avg(mark), department, subject.name
  FROM mark, exam, subject
  WHERE mark.exam_id = exam.id
     and exam.subject_id = subject.id
     and subject.name = 'Математика'
  GROUP BY department, subject.name;
```

18. На какие факультеты средний балл абитуриентов по всем экзаменам выше, чем 4?

```
SELECT avg(mark), department
  FROM mark, exam
  WHERE mark.exam_id = exam.id
  GROUP BY department
  HAVING avg(mark) > 4;
```

19. Среди абитуриентов, сдававших математику, подсчитать количество положительных оценок по каждому факультету.

```
SELECT count(mark), department
  FROM mark, subject, exam
  WHERE mark.exam_id = exam.id
     and exam.subject_id = subject.id and mark > 2
  GROUP BY department;
```

20. Какие баллы набрали абитуриенты на каждый факультет?

```
SELECT entrant_id, department, sum(mark)
  FROM exam, mark
  WHERE mark.exam_id = exam.id
  GROUP BY entrant_id, department;
```

21. Вывести информацию по абитуриентам, получившим пятерки по математике или физике.

```
SELECT last_name, entrant.name, second_name, mark,
    subject.name
  FROM entrant, mark, subject, exam
  WHERE mark.entrant_id = entrant.id
     and mark.exam_id = exam.id
     and exam.subject_id = subject.id
     and mark = 5 and subject.name in
       ('Математика','Физика');
```

22. Какие студенты сдали экзамены (получили оценку строго больше 2) с кодом 1 или 2 или 3?

```
SELECT distinct last_name, entrant.name, second_name
  FROM entrant, mark, subject, exam
  WHERE mark.entrant_id = entrant.id
     and mark.exam_id = exam.id
     and exam.subject_id = subject.id
     and mark > 2 and exam.id in (1,2,3);
```

23. Какие абитуриенты получили больше других пятерок?

```
SELECT entrant_id, count(*)
  FROM mark
  WHERE mark = 5
  GROUP BY entrant_id
  HAVING count(*) >
    ANY (
      SELECT count(*) FROM mark
      WHERE mark = 5
      GROUP BY entrant_id);
```

24. Какие абитуриенты сдали математику лучше, чем в среднем сдали различные предметы абитуриенты, поступающие на ВМК?

```
SELECT distinct entrant_id
  FROM mark, exam, subject
  WHERE mark.exam_id = exam.id
      and exam.subject_id = subject.id
      and subject.name = 'Математика'
      and mark > (
```

```
SELECT avg(mark) FROM mark, exam
 WHERE mark.exam_id = exam.id
     and department = 'ВМК');
```

25. По каким предметам проводилось меньше экзаменов, чем по другим?

```
SELECT subject.name
  FROM exam, subject
  WHERE exam.subject_id = subject.id
  GROUP BY subject.name
  HAVING count(*) < ANY (
    SELECT count(*) FROM exam GROUP BY subject_id );
```

26. Какие абитуриенты сдали все экзамены, которые есть в расписании?

```
SELECT entrant_id FROM mark
  GROUP BY entrant_id
  HAVING count(*) = (SELECT count(*) FROM exam);
```

27. Найти экзамены, которые пока никто не сдавал.

```
SELECT id FROM exam
  WHERE not exists
    (SELECT * FROM mark
       WHERE mark.exam_id = exam.id);
```

28. Найти абитуриентов, которые получили только «отлично» при поступлении на более чем один факультет?

```
SELECT last_name, department as d
  FROM entrant, mark m1, exam e1
  WHERE m1.entrant_id = entrant.id
     and m1.exam_id = e1.id
     and mark = 5
     and exists (
       SELECT * FROM mark, exam
         WHERE mark.exam_id = exam.id
             and entrant_id = m1.entrant_id
             and department <> e1.department);
```