БПОУ ВО «Грязовецкий политехнический техникум»

**Рассмотрен**

цикловой комиссией по общепрофессиональным дисциплинам и профессиональным модулям отделения «Электрификация и автоматизация сельского хозяйства»

**Согласовано**

зам. директора по ОМР
_____ Е. А. Ткаченко
*«30» августа 2017 г.*

*Протокол № 1 от «30» августа 2017 г.*

Председатель комиссии:
_____ Т. В. Невзорова

# ПРАКТИЧЕСКИЕ (ЛАБОРАТОРНЫЕ) РАБОТЫ

**ОП.05.Основы программирования и баз данных**
Специальность 09.02.02. Компьютерные сети

Грязовец
2017 г.

## Лабораторная работа №1.

## Составление разветвляющихся и циклических алгоритмов и программ.

**Цель работы:** закрепить практические навыки работы с системой Borland Pascal, научиться правильно использовать различные операторы циклов; научиться составлять программы решения задач с использованием циклических структур.

### Общие сведения

Алгоритм называется циклическим, если он содержит многократное выполнение одних и тех же операторов при различных значениях промежуточных данных. Число повторений этих операторов может быть задано в явной (цикл с известным заранее числом повторений) или неявной (цикл с неизвестным заранее числом повторений) форме.

# Операторы цикла

#### Цикл с параметром

Оператор цикла применяется при выполнении расчетов или других действий, повторяющихся определенное количество раз. Оператор имеет вид:

```
For  i:= N1    To    N2  Do  "оператор";
```

либо

```
For  i:= N1  DownTo  N2  Do "оператор";
```

Здесь i - параметр цикла (переменная порядкового типа),
N1, N2 - начальное и конечное значения параметра цикла i.
N1, N2 могут быть константами, переменными или выражениями порядкового типа.
Напомним, что "оператор" может иметь вид: Begin "операторы" end;
В случае связки "To" цикл выполняется при условии N1 <= N2 и происходит с единичным возрастанием параметра цикла i от N1 до N2. В случае связки DownTo цикл выполняется при условии N1 >= N2 и происходит с единичным уменьшением параметра цикла i от N1 до N2.
В операторе цикла не разрешается присваивать параметру цикла какое-либо значение.
После окончания цикла значение параметра цикла "i" неопределенно.
Оператор цикла часто применяется для суммирования значений некоторой последовательности чисел или значений функции при известном числе операций суммирования. Напомним некоторые определения, связанные с расчетом суммы последовательности.
Сумма членов последовательности величин

$$a_1, a_2, a_3, \ldots, a_n$$

называется конечной суммой

$$S_n = a_1 + a_2 + a_3 + \ldots + a_n$$

Для некоторых последовательностей известны формулы расчета конечных сумм, например:
при $a_n = a_n\text{-}1 + d$; $S_n = (a_1 + a_n)*n/2$; - арифметическая прогрессия,

при $a_n = a_n\text{-}1 * q$; $S_n = (a_1 - a_n*q)/(1-q)$; - геометрическая прогрессия,
где d и q - постоянные числа.
Здесь N-ый член последовательности выражается через (N-1)-ый член. Такие зависимости называются реккурентными.
Конечная сумма последовательности может быть неизвестна, тогда для ее расчета применяется алгоритм суммирования членов последовательности в цикле от 1 до N. Приведем пример расчета конечной суммы последовательности: 12 + 32 + 52 +. . . + $(2*N-1)_2$; $S_n = N*(4*N_2-1)/3$;

```
PROGRAM SUM_K;                      { расчет конечной суммы }
var
  a, S, Sn, i, N : word;
Begin
  write('Введите число членов суммы N=');
  readln(N);
  S:= 0;
  For i:= 1 to N do
    begin                           {  цикл суммирования }
      a := Sqr(2*i-1);
      S:= S+a
    end;
  Sn := N*(4*N*N-1) div 3;
  Writeln('Конечная сумма S=',  S:10:2);
  Writeln('Расчет конечной суммы по формуле Sn=',  Sn:10:2);
  Writeln('Нажми Enter');
  ReadLn
End.
```

В некоторых случаях "N"-ый член последовательности определяется через сумму предыдущих членов, например,

$$a_n = p*S_{n-1},$$

тогда

$$S_n = S_{n-1} + a_n = S_{n-1}*(1+p),$$

и конечную сумму можно рассчитать по формуле:

$$S_n = S_0*(1+p)N,$$

где "$S_0$" - начальная сумма.
Рассмотрим программу вычисления конечной суммы денежного вклада в банк через N месяцев при ежемесячной процентной ставке "pr" (5% соответствует pr=5).

```
PROGRAM VKLAD;        { расчет конечной суммы вклада в банк }
var
  S, Sn, pr : Real;
  i, N      : Integer;
Begin
  Write('Введите начальную сумму вклада S=');
  readln(S);
  Write('Введите процент по вкладу pr=');
  readln(pr);
  Write('Введите количество месяцев вклада N=');
  readln(N);
  For i:= 1 to N do S:= S*(1+pr/100);     { цикл произведений }
  Writeln('Конечная сумма вклада S=',  S:10:2);
  { Оператор для расчета "Sn" напишите самостоятельно }
  Writeln('Расчет конечной суммы вклада по формуле Sn=', Sn:10:2);
  Writeln('Нажмите Enter');
```

```
  readln
End.
```

Часто применяются вложенные операторы цикла. Например, если необходимо провести все варианты расчета при изменении нескольких параметров в заданных диапазонах.

Составим программу расчета функции y = A*sin(x) - cos(x)/A; при изменении аргумента "x" в диапазоне от 0 до Pi с шагом Pi/100 и при изменении параметра "A" в диапазоне от 1 до 3 с шагом 0.5.

```
Program tabl;
var
  y, x, a, dx : real;
  i, j: integer;
Begin
  Writeln(' Расчет по формуле:  y=A*sin(x)-cos(x)/A; ');
  Writeln('-----------------------------------------------');
  Writeln('|   X   |  A=1.0 | A=1.5 | A=2.0 | A=2.5 | A=3.0 |');
  Writeln('-----------------------------------------------');
  dx := pi/100;
  for i:= 0 to 100 do
     begin    { внешний цикл изменения аргумента "X" }
       x:= dx*i;
       Write( x:8:4 );
       for j := 1 to 5 do
         begin{ вложеннный цикл изменения параметра "A" }
           A := 0.5*(j+1);
           y := A*sin(x)-cos(x)/A;    Write(y:8:4)
         end;
       Writeln;                                    {перевод курсора на
новую строчку}
       if ((i+1) mod 20) = 0 then readln{задержка прокрутки экрана до
нажатия Enter}
     end;
  readln;
End.
```

**Операторы цикла с условием**

В Турбо-Паскале применяются два оператора цикла с условием:

```
While  "условие"  DO  "оператор";
```

- цикл с предусловием: проверка условия перед каждым выполнением "оператора",

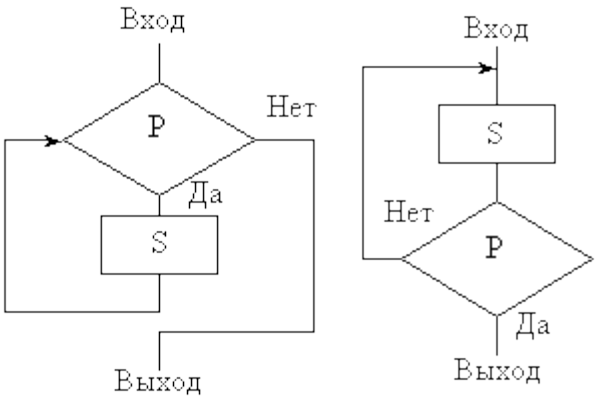
```
Repeat  "операторы"  Until  "условие";
```

- цикл с постусловием: проверка условия после каждого выполнения "операторов".

Здесь "условие" - выражение логического типа (Boolean).

Схема выполнения операторов имеет вид:

**Цикл WHILE** **Цикл REPEAT**




В цикле While "оператор" выполняется если условие верно (True), если условие ложно (False), то цикл заканчивается, т. е. цикл While повторяется пока выполняется условие. Цикл While начинается проверкой условия, поэтому, если начальное условие ложно, то "оператор" не выполняется ни разу. Для включения в тело цикла нескольких операторов применяется составной оператор: Begin "операторы" end.
Цикл Repeat повторяется, если условие ложно (False), и заканчивается, если условие верно (True), т. е. цикл Repeat повторяется до выполнения условия.
Цикл Repeat заканчивается проверкой условия, поэтому "операторы" выполняются не менее одного раза. В теле цикла может записываться более одного оператора.
Циклы с условием обычно используются в тех случаях, если количество повторений блока операторов заранее не известно, например, при расчете суммы членов бесконечного ряда с заданной погрешностью.
Сумма членов бесконечной последовательности

$$a_1, a_2, a_3, \ldots, a_n, \ldots$$

называется бесконечным рядом и записывается в виде:

$$a_1 + a_2 + a_3 + \ldots + a_n + \ldots$$

Здесь $a_n$ - общий член ряда.
Сумма конечного числа членов ряда называется частичной суммой и обозначается "$S_n$".
Если сумма членов бесконечного ряда имеет конечный предел "S", то ряд называется сходящимся. Для некоторых рядов получены формулы расчета суммы членов ряда. Например, сумма членов числового ряда:

$$1 + 1/32 + 1/52 + \ldots + 1/(2*N-1)<sup2< sup=""> + ... </sup2<>$$

имеет предел $S = Pi^2/8$ и общий член $a_n = images/(2*N-1)^2$, где N = 1, 2, 3, ...
Для сходящегося ряда вычисляется последовательность частичных сумм с заданной погрешностью. Абсолютная погрешность расчетов определяется по формуле $Eps=abs(S-S_n)$, либо $Eps=abs(a_n)$, если значение S неизвестно.
Относительная погрешность расчетов определяется по формуле $Eps\_o=abs((S-S_n)/S)$, либо $Eps\_o=abs(a_n/S_n)$.
Частичные суммы вычисляются по формуле: $S_n = S_{n-1} + a_n$

Для знакопеременного ряда следует добавить k1=-1, а в цикле: k1:=-k1, $a_n$=k1*$a_n$. В некоторых случаях "N"-ый член ряда выражается через "N-1"-ый, например, для ряда:

$$1 + 1/2! + 1/4! + 1/6! + ... + 1/(2*N)! + ... ; N = 0, 1, 2, ...$$

общий член ряда вычисляется по формуле: $a_n = a_{n-1}*k$;
Параметр $k = a_n/a_{n-1}$ - коэффициент роста вычисляется предварительно (до написания программы).
Для данного ряда

$$a_n = 1/(2*N)! = 1/( 1*2*...*(2*N-2)*(2*N-1)*2*N)$$
$$a_{n-1} = 1/(2*(N-1))! = 1/((2*N-2))! = 1/(1*2*...*(2*N-2))$$
$$k = a_n/a_{n-1} = 1/((2*N-1)*2*N)$$

Здесь N! = 1*2*3*...*N; - вычисление факториала числа "N", причем 0! = 1.
Расчет частичных сумм производится в цикле с условием, например, для данного ряда операторами:

```
  N  := 0;
  a  := 1;
  SN := 1;
  S  := (e+1)/e;
  e := 2.7182828;
  Repeat
    N := N+1;
    k := 1/((2*N-1)*2*N);
    a := a*k;
    SN := SN+a;
    Writeln('Частичная сумма Sn= ',  Sn:11:6);
  Until abs(S-Sn) < eps;              { eps - допустимая погрешность
расчетов}
  Writeln('Сумма ряда S = ', SN:11:6);
```

**Операторы ограничения и прерывания цикла**

Данные операторы применяются внутри операторов цикла с параметром или условием. Операторы имеют вид:

```
     Continue;   -  ограничение цикла,
     Break;      -  прерывание цикла.
```

Операторы Continue и Break позволяют производить действия не для всех операторов внутри цикла. Действие оператора Continue заключается в передаче управления на начало цикла, при этом контролируется условие выхода из цикла. Действие оператора Break заключается в передаче управления оператору, следующему за последним оператором цикла, при этом не контролируется условие выхода из цикла. Во вложенных циклах операторы Continue и Break действуют только на цикл в котором они записаны. Приведем пример использования операторов для блокировки несанкционированного доступа в программу.

```
  For i := 1 to 3 do
    begin
      Write( 'Введите ПАРОЛЬ:' );    Readln(S); {S и Parol - переменные
одного типа}
      If S = Parol  Then  Break                                  {
прерывание цикла }
      else  If   i <> 3  Then Continue;                         {
ограничение цикла }
      Writeln( 'Доступ к программе ЗАПРЕЩЕН' );
      Writeln( 'Нажмите Enter' );
```

```
      Readln;
      Halt                                                    {
прерывание программы }
    end;                                                     {
продолжение программы }
```

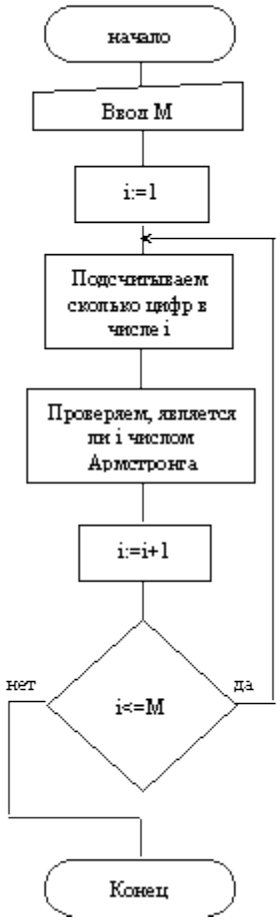
## Примеры

**Пример1**: На промежутке от 1 до M найти все числа Армстронга. Натуральное число из n цифр называется числом Армстронга, если сумма его цифр, возведенных в n-ю степень, равна самому числу.

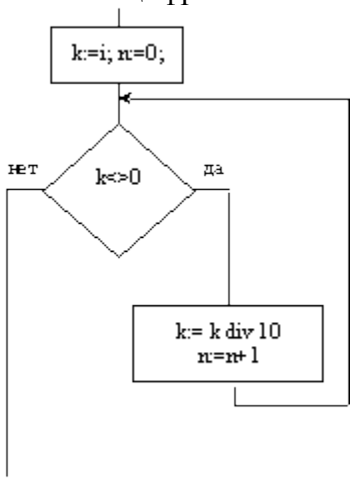
**Этапы решения задачи:**

1. Математическая модель: xO[1;M], x=
2. Составим блок схему программы:

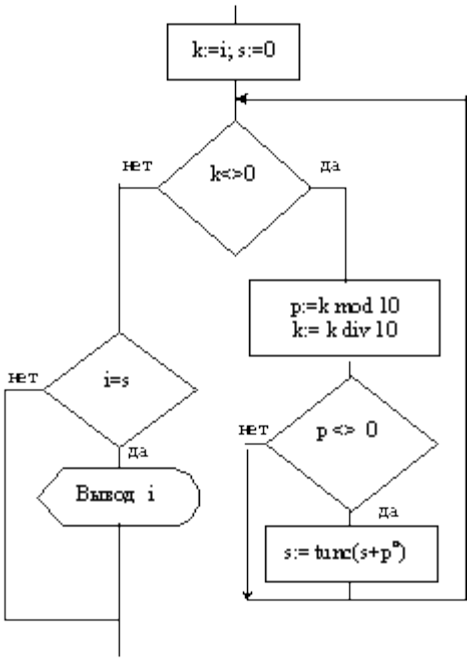

Распишем составные части блока"Находим все числа Армстронга на заданном промежутке и печатаем их"



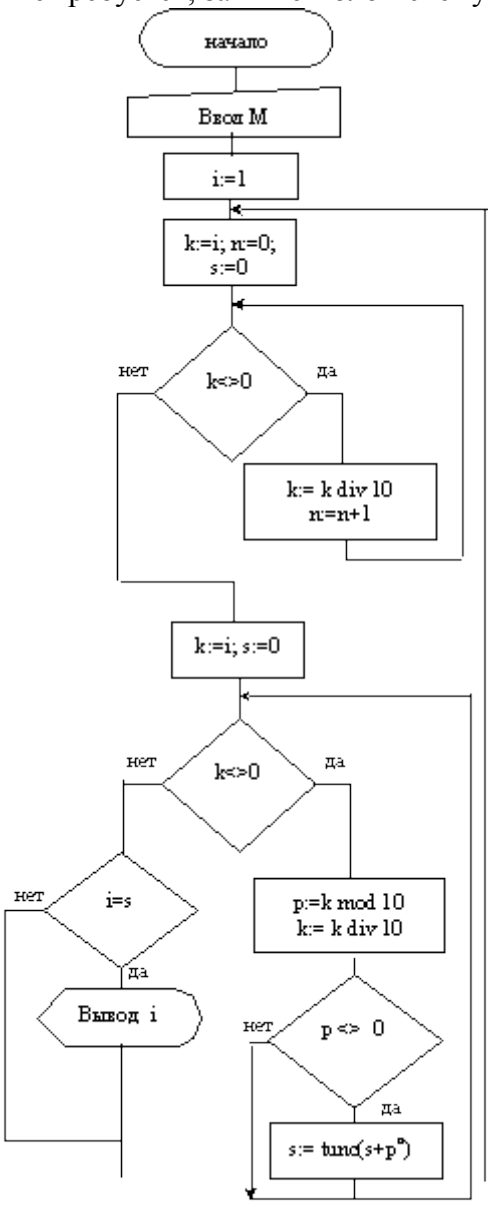

Опишем блок "Подсчитываем сколько цифр в числе i"




Опишем блок "Проверяем, является ли i числом Армстронга"




Дальнейшая детализация не требуется, запишем блок-схему целиком:




Дальнейшей детализации не требуется, переведем программу на язык Паскаль.

```
PROGRAM Primer_1;
var i,k,s,p,n: Integer;
   BEGIN
     Write('Введите M ');
   Readln(m);
          For i:=1 to M do
        begin
          s:=0; k:=i; n:=0;
          While k<>0 do
            begin k:=k DIV 10; n:=n+1 end;
              k:=i;
              While k<>0 do
              begin p:=k MOD 10; k:=k DIV 10;
                If p<>0 then s:=Trunc (s+Exp(n*Ln(p)))
              end;
              If s=f then WriteLn (f)
       end;
END.
```

## Контрольные вопросы

1. Как записывается и как работает оператор FOR?
2. Для организации каких циклов применим оператор FOR?
3. В чем отличие оператора WHILE от оператора REPEAT?
4. Как программируются циклические алгоритмы с явно заданным числом повторений цикла?
5. Как программируются циклические алгоритмы с заранее неизвестным числом повторений цикла?
6. Напишите оператор цикла, который не выполняется ни разу.
7. Напишите оператор цикла, который выполняется неограниченное число раз.
8. Замените оператор "Repeat A Until B" равносильным фрагментом программы с оператором While.

## Задачи

1. Найти все двузначные числа, сумма цифр которых не меняется при умножении числа на 2,3,4,5,6,7,8,9.
2. Найти все трехзначные числа, сумма цифр которых равна данному целому числу.
3. Найти все трехзначные числа, средняя цифра которых равна сумме первой и третьей цифр.
4. Найти все трехзначные числа, которые можно представить разностью между квадратом числа, образованного первыми двумя цифрами и квадратом третьей цифры.
5. Найти все двузначные числа, сумма квадратов цифр которых делится на 17.
6. Найти все трехзначные числа, представимые в виде сумм факториалов своих цифр.
7. Найти двузначное число, обладающее тем свойством, что куб суммы его цифр равен квадрату самого числа.
8. Найти двузначное число, равное утроенному произведению его цифр.
9. В каких двузначных числах удвоенная сумма цифр равна их произведению?

10. Можно ли заданное натуральное число M представить в виде суммы квадратов двух натуральных чисел? Написать программу решения этой задачи.

**Вычисление выражений:**
Дано натуральное n. Вычислить:

11. $(1+\frac{1}{1^2})+(1+\frac{1}{2^2})+(1+\frac{1}{3^2})+\ldots+(1+\frac{1}{n^2})$;
12. $\frac{1}{\sin 1}+\frac{1}{\sin 1+\sin 2}+\ldots+\frac{1}{\sin 1+\ldots \sin n}$;

Дано действительное число x, натуральное число n. Вычислить:

13. x ( x - n )( x - 2 n )( x - 3 n )…( x - n2 );
14. $\frac{1}{x}+\frac{1}{x(x+1)}+\ldots+\frac{1}{x(x+1)\ldots(x+n)}$;
15. $\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots+\frac{x^n}{n!}$;

Дано натуральное n. Вычислить:

16. $\prod_{i=1}^{n}(2+1/i!)$;
17. $\sum_{i=1}^{n}(1+i)/i!$;

Вычислить приближенно значение бесконечной суммы (справа от каждой суммы дается ее точное значение, с которым можно сравнить полученный ответ):

18. $1+\frac{1}{2^2}+\frac{1}{3^2}+\frac{1}{4^2}+\ldots=\frac{\pi^2}{6}$;
19. $\frac{1}{1\cdot 3}+\frac{1}{2\cdot 4}+\frac{1}{3\cdot 5}+\ldots=\frac{3}{4}$;
20. $1+\frac{x^1}{1!}+\frac{x^2}{2!}+\frac{x^3}{3!}+\ldots=e^x$;

Нужное приближение считается полученным, если вычислена сумма нескольких первых слагаемых, и очередное слагаемое оказалось по модулю меньше данного положительного числа e.

21. Даны два целых числа A и B (A < B). Вывести все целые числа, расположенные между данными числами (не включая сами эти числа), в порядке их возрастания, а также количество N этих чисел.
22. Даны два целых числа A и B (A < B). Вывести все целые числа, расположенные между данными числами (включая сами эти числа), в порядке их убывания, а также количество N этих чисел.
23. Дано вещественное число A и целое число N (> 0). Вывести A в степени N: AN = A·A·...·A (числа A перемножаются N раз).
24. Дано вещественное число A и целое число N (> 0). Вывести все целые степени числа A от 1 до N.
25. Дано вещественное число A и целое число N (> 0). Вывести 1 + A + A2 + A3 + ... + AN.

26. Дано вещественное число A и целое число N (> 0). Вывести 1 - A + A2 - A3 + ... + (-1)NAN.
27. Дано целое число N (> 1). Вывести наименьшее целое K, при котором выполняется неравенство 3K > N, и само значение 3K.
28. Дано целое число N (> 1). Вывести наибольшее целое K, при котором выполняется неравенство 3K < N, и само значение 3K.
29. Дано вещественное число A (> 1). Вывести наименьшее из целых чисел N, для которых сумма 1 + 1/2 + ... + 1/N будет больше A, и саму эту сумму.
30. Дано вещественное число A (> 1). Вывести наибольшее из целых чисел N, для которых сумма 1 + 1/2 + ... + 1/N будет меньше A, и саму эту сумму.
31. Дано целое число N (> 0). Вывести произведение 1·2·...·N. Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и выводить его как вещественное число.
32. Дано целое число N (> 0). Если N - нечетное, то вывести произведение 1·3·...·N; если N - четное, то вывести произведение 2·4·...·N. Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и выводить его как вещественное число.
33. Дано целое число N (> 2) и две вещественные точки на числовой оси: A, B (A < B). Отрезок [A, B] разбит на равные отрезки длины H с концами в N точках вида A, A + H, A + 2H, A + 3H, ..., B. Вывести значение H и набор из N точек, образующий разбиение отрезка [A, B].
34. Дано целое число N (> 2) и две вещественные точки на числовой оси: A, B (A < B). Функция F(X) задана формулой F(X) = 1 - sin(X). Вывести значения функции F в N равноотстоящих точках, образующих разбиение отрезка [A, B]: F(A), F(A + H), F(A + 2H), ..., F(B).
35. Дано число D (> 0). Последовательность чисел AN определяется следующим образом: A1 = 2, AN = 2 + 1/AN-1, N = 2, 3, ... Найти первый из номеров K, для которых выполняется условие |AK - AK-1| < D, и вывести этот номер, а также числа AK-1 и AK.
36. Дано число D (> 0). Последовательность чисел AN определяется следующим образом: A1 = 1, A2 = 2, AN = (AN-2+ AN-1)/2, N = 3, 4, ... Найти первый из номеров K, для которых выполняется условие |AK AK-1| < D, и вывести этот номер, а также числа AK-1 и AK.

**Задачи повышенной сложности**

1. Определить, является ли заданное число совершенным , т.е. равным сумме всех своих (положительных) делителей, кроме самого этого числа (например, число 6 совершенно: 6=1+2+3).
2. Дано натуральное k. Напечатать k-ю цифру последовательности 12345678910111213 14..., в которой выписаны подряд все натуральные числа.
3. Дано натуральное k. Напечатать k-ю цифру последовательности 149162536..., в которой выписаны подряд квадраты всех натуральных чисел.
4. Дано натуральное k. Напечатать k-ю цифру последовательности 1123581321..., в которой выписаны подряд все числа Фибоначчи.
5. Вычислить, многократно применяя итерационную формулу $x_{i+1} = x_i + \frac{1}{3} \cdot \left( \frac{A}{x_i^2} - x_i \right)$. Начальное приближение выбрать самостоятельно.

   Прекратить вычисления, если разность двух последовательных итераций станет меньше, чем произведение последнего приближения на .

## Лабораторная работа № 2.

## Составление алгоритмов и программ с использованием массивов

**Цель работы:** научиться правильно описывать различные массивы, уметь инициализировать массивы, распечатывать содержимое массива; научится решать задачи на использование массивов.

# Общие сведения:

Массив - это структурированный тип данных, который используется для описания упорядоченной совокупности фиксированного числа элементов одного типа, имеющих общее имя. Для обозначения элементов массива используются имя переменной-массива и индекс.

# Массивы

Массив - упорядоченные данные одного типа. Возможно создание массива, включающего массив другого типа. Массивом часто обозначают характеристики обьектов одного типа, имеющих одинаковые единицы измерения. Массив состоит из элементов, имеющих порядковые номера, т. е. элементы массива упорядоченны. Таким образом, если обьекты одного типа обозначить именем, например "A", то элементы обьекта будут A[1], A[2] и т. д. В квадратных скобках указан номер элемента. Порядковый номер элемента массива, обычно не несет никакой информации о значении элемента, а показывает расположение элемента среди других. К элементам массива можно обращаться только по их номеру (индексу). Значения элементам массива присваиваются также как и другим переменным с учетом типа массива. Если элементы массива имеют один индекс, то массив называется одномерным или линейным, либо массив - вектор. Значения элементов одномерного массива обычно выводят на экран или бумагу в виде столбца или строки. В некоторых случаях удобно элементы массива пронумеровывать двуми независимыми индексами, такие массивы называются двумерными или матрицами. Значения элементов двумерного массива обычно выводят на экран в виде таблицы. Если элементы массива имеют три независимых индекса, то массив называется трехмерным. Значения элементов трехмерного массива обычно выводят на экран в виде набора таблиц.

#### Линейные массивы

Линейным массивом можно обозначить, например, оценки учеников класса. Каждая оценка является значением элемента массива оценок "A" и имеет порядковый номер (индекс). В Турбо-Паскале значение индекса указывается в квадратных скобках после имени массива. Можно создать массив фамилий "S" учеников класса. Значением элемента массива будет фамилия ученика, а индексом - порядковый номер по списку. Пусть дан список фамилий учеников и их оценки:

| № | Фамилии | Оценки |
|---|---|---|
| 1 | Иванов | 5 |
| 2 | Петров | 4 |

| | | |
|---|---|---|
| 3 | Сидоров | 5 |
| 4 | Титов | 5 |
| ... | ... | ... |
| 30 | Якупов | 4 |

**Описание массивов:**

```
Var A : array[1..30] of byte;
    S : array[1..30] of string; {или}
    SO: array[1..30] of string[12];
```

**Присвоение значений элементам массива:**

```
 "A" - A[1]:= 5;   A[2]:= 4;                    и т. д.
 "S " - S[1]:= 'Иванов'; S[2]:= 'Петров';       и т. д.
```

Приведем таблицу обозначений и соответствия элементам массива, их значений и индексов:

| **Номер элемента индекса** | 1 | 2 | 3 | 4 | 1 | 30 |
|---|---|---|---|---|---|---|
| **Элементы массива "S"** | S[ 1 ] | S[ 2 ] | S[ 3 ] | S[ 4 ] | S[ i ] | S[ 30 ] |
| **Значения элементов** | Иванов | Петров | Сидоров | Титов | ... | Якупов |
| **Элементы массива "A"** | A[ 1 ] | A[ 2 ] | A[ 3 ] | A[ 4 ] | A[ i ] | A[ 30 ] |
| **Значения элементов** | 5 | 4 | 5 | 5 | ... | 4 |

Если известна зависимость, по которой изменяются значения элементов массива, то присвоение значений удобно проводить в операторах цикла с параметром или с условием. Например, присвоим значения элементам массива "y" по зависимости: y=sin(x), где x= Pi * i/180, 0<= i <=180.

```
For i:= 0 to 180 Do y[i]:= sin(Pi * i/180);
```

Присвоим случайные значения в диапазоне от -30 до +40 ста элементам массива "R":

```
Randomize;
for i:= 1 to 100 Do R[i]:= - 30 + Random(71);
```

Присвоим значения семи элементам массива "A" оператором Readln:

```
For i:= 1 to 7 Do
  begin
    Write('Введите A[',i,']= ');
    Readln(A[i]);
  end;
```

При выводе массива на экран удобно размещать данные в виде таблицы - в несколько колонок. Для вывода обозначений переменных ("шапки таблицы") можно использовать операторы вывода символов в цикле, например:

```
For j:=1 to 66  do  Write('-'); Writeln;
For j:=1 to 3 do Write('|    Фамилия    |   оценка  |'); Writeln;
For j:=1 to 66  do  Write('-'); Writeln;
```

- шапка для вывода в три пары колонок значений переменных "S" и "A". Шапка занимает 66 позиций (по ширине экрана в текстовом режиме размещается 79 символов и пробел). Оператор Writeln; переводит курсор на новую строчку.

Вывод значений ста элементов массивов "S" и "A" в три пары колонок, произведем операторами:

```
For i:= 1 to 100 do
  begin
    Write('|', s[i]:11,'|', a[i]:8, '|');
    if (i mod 3) = 0 Then Writeln;
    if (i mod 60) = 0 then Readln;
  end;
```

В этом случае данные таблицы полностью не умещаются на экране и можно задержать прокрутку экрана при выводе данных, применяя оператор Readln после вывода, например, 20 строк.

В цикле удобно определять сумму элементов массива, наибольший (наименьший) элемент и создавать новые массивы, удовлетворяющие некоторому условию, например:

```
  s := 0;
  for i := 1 to 100 do s:= s + a[i];   { s - сумма элементов массива}
  a_max := a[1];
  for i:= 1 to 100 do     { поиск наибольшего элемента a[j] }
  if a[i] > a_max then begin  a_max:= a[i];  j:= i; end;
  j := 0; k := 0;
  for i:=1 to 100 do {создание новых массивов с элементами: b[j]>=0,
c[k]<0}
  if a[i] >= 0 then
    begin
      j := j+1;
      b[j] := a[i];
    end
  else
    begin
      k := k+1;
      c[k] := a[i]
    end;
  j := 0;
  k := 8;
  for i:= 1 to 100  do {создание массива номеров "M" для элементов:
a[i] > a[k]}
    if a[i] > a[k] then
      begin
        j := j+1;
        M[j] := i;
      end;
```

**Двумерные массивы**

Массивы, рассмотренные выше, имеют элементы, упорядоченные по одному индексу и называются одномерными массивами или векторами. Массив может быть двумерным, трехмерным и т. д. Двумерные массивы имеют элементы, упорядоченные по двум индексам и часто называются матрицами. В Турбо-Паскале при описании многомерного массива диа-пазоны изменения индексов перечисляются через запятые, например:

```
   Var
      A: array[1..30, 1..7] of byte;
```

Рассмотрим пример работы с двумерными массивами. Обозначим массивом оценки учеников класса по нескольким предметам. Каждая оценка является значением элемента массива оценок "A" и имеет порядковый номер (два индекса). Поставим в соответствие первому индексу номер фамилии в списке учеников, а второму - номер предмета, по которому получена оценка. Тогда двумерный массив оценок можно представить в виде таблицы: каждый элемент a[i, j] находится на пересечении I-ой строки и J-го столбца.
Исходные данные могут быть представлены в виде таблицы оценок:
Годовые оценки по предметам:

| | | **1** | **2** | **3** | **4** | **5** | **6** |
|---|---|---|---|---|---|---|---|
| **№** | **Фамилия** | Физика | Химия | Алгебра | Геометрия | История | Биология |
| 1 | Иванов | 4 | 5 | 3 | 4 | 5 | 5 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 2 | Петров | 4 | 5 | 4 | 3 | 4 | 4 |
| 3 | Сидоров | 5 | 5 | 3 | 4 | 5 | 4 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 30 | Якупов | 4 | 3 | 4 | 5 | 4 | 5 |

Можно создать одномерные массивы фамилий "S" учеников класса и наименований предметов "P" . Значением элемента массива "P" будет наименование предмета, а индексом - порядковый номер предмета, например: 1 - физика, 2 - химия, 3 - алгебра, 4 - геометрия, 5 - история, 6 - биология. Приведенная выше таблица может быть представлена в виде набора элементов ( число строк = N, число столбцов = M ):

| **Номер строки "I"** | **Номер столбца "J"** | | **1** | **2** | **3** | **4** | **...** | **J** | **...** | **M** |
|---|---|---|---|---|---|---|---|---|---|---|
| | **Массив "S"** | **Массив "P"** | P[1] | P[2] | P[3] | P[4] | ... | P[J] | ... | P[M] |
| **1** | S[1] | Массив "A" | a[1, 1] | a[1, 2] | a[1, 3] | a[1, 4] | ... | a[1, j] | ... | a[n, m] |
| **2** | S[2] | | a[2, 1] | a[2, 2] | a[2, 3] | a[2, 4] | ... | a[2, j] | ... | a[2, m] |
| **3** | S[3] | | a[3, 1] | a[3, 2] | a[3, 3] | a[3, 4] | ... | a[3, j] | ... | a[3, m] |
| **4** | S[4] | | a[4, 1] | a[4, 2] | a[4, 3] | a[4, 4] | ... | a[4, j] | ... | a[4, m] |
| **...** | ... | | ... | ... | ... | ... | ... | ... | ... | ... |
| **I** | S[I] | | a[i, 1] | a[i, 2] | a [i, 3] | a [i, 4] | ... | a[i, j] | ... | a[i, m] |
| **...** | ... | | ... | ... | ... | ... | ... | ... | ... | ... |
| **N** | S[N] | | a[n, 1] | a[n, 2] | a[n, 3] | a[n, 4] | ... | a[n, j ] | ... | a[n, m] |

Массив оценок можно задать с использованием функции Random, например:

```
for i:= 1 to N do
  for j:= 1 to M do A[i, j]:= random(4) + 2;
```

Для вывода наименований предметов ( "шапка" таблицы ) можно использовать операторы:

```
Writeln;
Write('Фамилия\\Предметы:|');
For i:= 1 to M do write(P[i]:9,'  |');
```

Для вывода элементов массива "A" на экран удобно использовать вложенный цикл:

```
for i:= 1 to N do
  begin
    writeln;
    write(S[i]:19,  '  |';
    for j:= 1 to M do write(A[i,j]:7, '     |')
 end;
```

Для расчета массива "SS" - сумм "M" элементов в каждой из "N" строк массива "A" (NxM) можно применить операторы:

```
for i:= 1 to N do
  begin
    SS[i]:= 0;
    for j:= 1 to M do SS[i]:= SS[i] + A[i,j];
```

```
    end;
```
Здесь для каждого индекса "i" от 1 до N происходит суммирование элементов A[i, j] по индексу "j" от 1 до M.
При модификации массива "A" изменяется расположение данных в исходном массиве, например, в случае вставки данных из линейного массива "B" в колонку с номером "M1" необходимо сдвинуть данные в колонках J >= M1 используя операторы:
```
  for i:= 1 to N do
    begin
      for j:=M+1 downto M1+1 do A[i,j]:=A[i,j-1];
      A[i,M1]:=B[i];
    end;
```
Если порядковый номер предмета изменится, то необходимо изменить расположение оценок в массиве "A", например, перестановку колонок с оценками по физике и химии можно сделать операторами:
```
  for j:= 1 to N do
    begin
      a1:=A[1,j];
      A[1,j]:=A[2,j];
      A[2,j]:=a1;
    end;
```
Примечание: при модификации массива "A" не забудьте соответственно изменять расположение данных в сопутствующих массивах, например, "P" или "S".
При создании новых массивов согласно некоторым условиям выбираются данные из элементов исходного массива. Например, для создания массива "B" из четных столбцов массива "A" можно использовать операторы:
```
  for j:= 1 to M do
   If (j Mod 2) = 0 then for i:= 1 to N do B[i,j Div 2]:= A[i,j];
```
Для создания массива "B", состоящего из строк массива "A", удовлетворяющих условию A[i, 1] > C, где C - заданное число, можно использовать операторы:
```
  k:= 0;
  for i:= 1 to N do
    If A[i,1] > C then
      begin
        k:= k + 1;
        for j:= 1 to M do B[k,j]:= A[i,j];
      end;
```
Для сравнения значений элементов массива удобно строить столбиковые диаграммы (гистограммы). Например, для вывода "N" значений положительных элементов массива "SS" в виде горизонтальной гистограммы можно использовать операторы:
```
  k:= 30/S_max;                { k - нормирующий масштабный коэффициент
}
                               { S_max - наибольший элемент массива
"SS" }
  for i:=1 to N do
    begin
      writeln;                 { переход к новому столбику }
      yg := round(k*SS[i]);    { yg - длина столбика гистограммы }
      for j:=1 to yg do write(#220); { вывод символа '_' с кодом 220 }
    end;
```
Добавив операторы вывода порядкового номера и значений SS[i] получаем


гистограмму в виде:

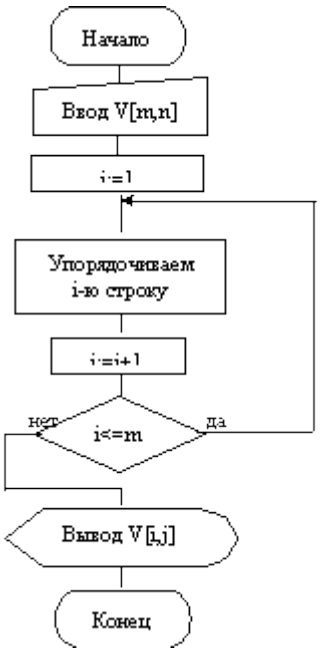
## Примеры:

**Пример1:** Дан двумерный массив. В каждой строке все его элементы, не равные нулю, переписать (сохраняя порядок) в начало строки, а нулевые элементы - в конец массива. Новый массив не заводить.

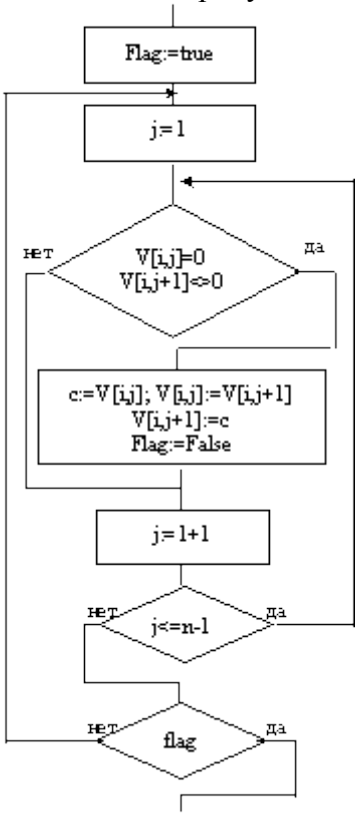
**Этапы решения задачи:**
1. Суть одного из алгоритмов решения данной задачи состоит в том чтобы "просматривать" массив построчно и находить в каждой строке пару (0:число), а затем менять их местами между собой и так до тех пор пока в строке таких пар не окажется.
2. Напишем программу на псевдо паскале:

```
program example1;
var
V:array[1..100,1..100] of integer;
   m,n, i,j, c: integer;
     flag: boolean;
begin
     <ввод размерности массива m*n>
     <заполнение ячеек массива>
for i:=1 to m do
      repeat
         flag:= true;
            for j:=1 to n-1 do
              if (v[i,j]=0) and (v[i,j+1]<>0) then begin
                <поменять их местами>
                  flag:= false;
                end;
             until flag;
         <Печать массива>
      readln;
   end.
```

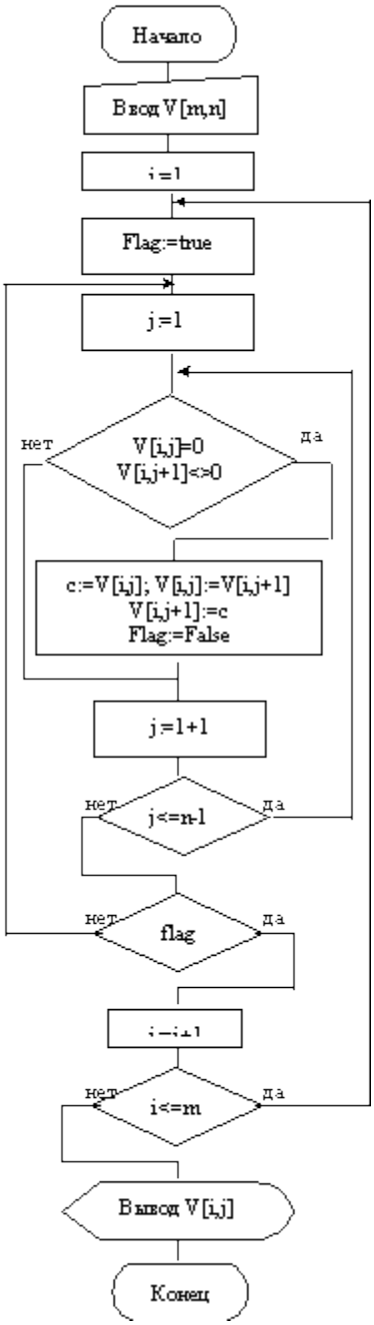
3.Составим блок схему алгоритма



Детализируем блок "Упорядочиваем 1-ю строку"




Блок схема алгоритма целиком:




4.Приведем программу на языке Паскаль:

```
program example1;
  var
    V:array[1..100,1..100] of integer;
      m,n, i,j, c: integer;
         flag: boolean;
begin
       write('Введите размерность массива m-n> ');
     readln(m,n);
         for i:= 1 to m do
            for j:= 1 to n do begin
               write('V[',i,',',j,']= ');
                    readln(V[i,j]);
  end;
    for i:=1 to m do
      repeat
          flag:= true;
              for j:=1 to n-1 do
```

```
          if (v[i,j]=0) and (v[i,j+1]<>0) then begin
         c:=v[i,j]; v[i,j]:=v[i,j+1]; v[i,j+1]:=c;
      flag:= false;
  end;
     until flag;
      for i:= 1 to m do begin
         for j:= 1 to n do write(V[i,j]:2);
         writeln
     end;
  readln;
end.
```

## Контрольные вопросы

1. Каким образом определяются переменные типа массив (одномерный и двумерный)?
2. Как осуществляется доступ к отдельному элементу одномерного и двумерного массива?
3. Каким образом выводятся элементы массива на экран?
4. Приведите пример фрагмента программы, который выводит на экран двумерный массив в виде матрицы.
5. Сколько чисел можно записать в шестимерный массив X : Array[0..1, 0..1, 0..1, 0..1, 0..1, 0..1] of Integer?

### Задания

1. Дан массив размера N. Вывести его элементы в обратном порядке.
2. Дан массив размера N. Вывести вначале его элементы с четными индексами, а затем - с нечетными.
3. Дан целочисленный массив A размера 10. Вывести номер первого и последнего из тех его элементов A[i], которые удовлетворяют двойному неравенству: A[1] < A[i] < A[10]. Если таких элементов нет, то вывести 0.
4. Дан целочисленный массив размера N. Преобразовать его, прибавив к четным числам первый элемент. Первый и последний элементы массива не изменять.
5. Дан целочисленный массив размера N. Вывести вначале все его четные элементы, а затем - нечетные.
6. Поменять местами минимальный и максимальный элементы массива размера 10.
7. Заменить все отрицательные элементы целочисленного массива размера 10 на минимальное значение элементов массива.
8. Дан массив размера N. Осуществить сдвиг элементов массива вправо на одну позицию.
9. Дан массив размера N и число k (0 < k < 5, k < N). Осуществить циклический сдвиг элементов массива влево на k позиций.
10. Проверить, образуют ли элементы целочисленного массива размера N арифметическую прогрессию. Если да, то вывести разность прогрессии, если нет - вывести 0.
11. Дан массив ненулевых целых чисел размера N. Проверить, чередуются ли в нем положительные и отрицательные числа. Если чередуются, то вывести 0, если нет, то вывести номер первого элемента, нарушающего закономерность.

12. Дан массив размера N. Определить количество участков, на которых его элементы монотонно возрастают.
13. Дан массив размера N. Определить количество его промежутков монотонности (то есть участков, на которых его элементы возрастают или убывают).
14. Дан целочисленный массив размера N. Определить максимальное количество его одинаковых элементов.
15. Дан целочисленный массив размера N. Если он является перестановкой, то есть содержит все числа от 1 до N, то вывести 0, в противном случае вывести номер первого недопустимого элемента.
16. Дан целочисленный массив размера N. Назовем серией группу подряд идущих одинаковых элементов, а длиной серии - количество этих элементов (длина серии может быть равна 1). Вывести массив, содержащий длины всех серий исходного массива.
17. Дано число k ($0 < k < 11$) и матрица размера 4 x 10. Найти сумму и произведение элементов k-го столбца данной матрицы.
18. Дана матрица размера а х б. Найти суммы элементов всех ее четных строк и нечетных столбцов.
19. Дана матрица размера а х б. Найти минимальное значение в каждой строке.
20. Дана матрица размера а х б. В каждой строке найти количество элементов, больших среднего арифметического всех элементов этой строки.
21. Дана матрица размера а х б. Преобразовать матрицу, поменяв местами минимальный и максимальный элемент в каждой а) строке б) столбце.
22. Дана целочисленная матрица размера а х б. Вывести номер ее первой строки, содержащего равное количество положительных и отрицательных элементов (нулевые элементы не учитываются). Если таких строк нет, то вывести 0.
23. Дана целочисленная матрица размера M x N. Найти количество ее строк и столбцов, все элементы которых различны.
24. Дана квадратная матрица порядка M. Найти сумму элементов ее главной и побочной диагонали.
25. Дана квадратная матрица порядка M. Заменить нулями элементы матрицы, лежащие а) ниже главной диагонали, б) выше главной диагонали, в) нижепобочной диагонали.
26. Дана квадратная матрица порядка M. Повернуть ее на 90,180,270 градусов в положительном направлении.
27. Даны два числа k1 и k2 и матрица размера а х б. Поменять местами столбцы матрицы с номерами k1 и k2.
28. Дано число k и матрица размера а х б. Удалить столбец матрицы с номером k.
29. Даны целые числа a1, a2, a3. Получить целочисленную матрицу [bij]i,j=1,2,3, для которой bij=ai-3aj.
30. Получить [aij]i=1,…10; j=1,…12 - целочисленную матрицу, для которой aij=i+2j.
31. Дано натуральное число n. Получить действительную матрицу [aij]i, j=1,…n, для которой aij= $\frac{1}{i+j}$ .
32. Дана действительная квадратная матрица порядка n. Найти наибольшее из значений элементов, расположенных в заштрихованной части матрицы.

33. Дана квадратная вещественная матрица размерности n. Найти количество нулевых элементов, стоящих: выше главной диагонали; ниже главной диагонали; выше и ниже побочной.
34. Дана вещественная матрица размерности n * m. По матрице получить логический вектор, присвоив его k-ому элементу значение True , если выполнено указанное условие и значение False иначе: - все элементы k столбца нулевые; - элементы k строки матрицы упорядочены по убыванию; - k строка массива симметрична.
35. Дана вещественная матрица размерности n * m. Сформировать вектор b, в котором элементы вычисляются как: - произведение элементов соответствующих строк; - среднее арифметическое соответствующих столбцов; - разность наибольших и наименьших элементов соответствующих строк; - значения первых отрицательных элементов в столбце.
36. Дан двухмерный массив A[1..m,1..n]. Написать программу построения одномерного массива B[1..m], элементы которого соответственно равны а) суммам элементов строк, б) произведениям элементов строк, в) наименьшим средних арифметических элементов строк.
37. Расположить элементы данного массива в обратном порядке (первый элемент меняется с последним, второй - с предпоследним и т.д. до середины; если массив содержит нечетное количество элементов, то средний остается без изменения).
38. В данном массиве поменять местами элементы, стоящие на нечетных местах, с элементами, стоящими на четных местах.

**Задачи повышенной сложности**

1. В массиве A[1..N,1..N] определить номера строки и столбца какой-нибудь седловой точки. Некоторый элемент массива называется седловой точкой, если он является одновременно наименьшим в своей строке и наибольшим в своем столбце.
2. Массив A[1..5,1..7] содержит вещественные числа. Требуется ввести целое число K и вычислить сумму элементов A[I,J], для которых I+J=K. Прежде, однако следует убедиться, что значение K позволяет найти решение, в противном случае нужно напечатать сообщение об ошибке.
3. Дан массив A[1..N,1..N]. Составить программу, которая прибавила бы каждому элементу данной строки элемент, принадлежащий этой строке и главной диагонали.
4. Дана матрица NxM. Переставляя ее строки и столбцы, переместить наибольший элемент в верхний левый угол. Определить можно ли таким же образом поместить минимальный элемент в нижний правый угол.
5. Заполнить двухмерный массив T[1..n,1..n] последовательными целыми числами от 1 до n 52 0, расположенными по спирали, начиная с левого верхнего угла и продвигаясь по часовой стрелке:

```
1   2   3   4   5   6
20   21   22   23   24   7
19   32   33   34   25   8
18   31   36   35   26   9
30   29   28   27   10
```

```
15   14   13   12   11
```

6. Элемент двухмерного массива называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером NxN найти максимум среди всех локальных минимумов.

## Лабораторная работа №3.

## Составление алгоритмов и программ по обработке строк.

Цель работы: познакомить с понятием "стринг" и выработать навыки работы с символьной информацией в языке программирования Pascal научиться использовать строки символов и множества при решении задач.

### Общие сведения

Переменные типа String аналогичны массивам типа Char. Их отличием является то, что число символов (длина строки) может динамически меняться в интервале от единицы до заданного верхнего значения.
Тип String (строка) в Турбо Паскале широко используется для обработки текстов. Этот тип является стандартным и во многом похож на одномерный массив символов Array [0..N] of Char. Значение N соответствует количеству символов в строке и может меняться от 0 до 255. Символы, входящие в строку, занимают позиции с 1 до N. Начальный байт строки с индексом 0 содержит информацию о ее длине, т.е. это символ с кодом, равным длине строки.
Можно, также описывать переменные типа String[K], где K - целое число не больше 255. Так определяются строки с длиной не больше K. Этот тип уже не является стандартным. С символами строки можно работать как с элементами массива из символов, но в отличие от массивов, строки можно вводить целиком, сравнивать друг с другом и сцеплять операцией "+".
Сравнение строк выполняется посимвольно в соответствии с их кодами до первого несовпадения. Если одна из строк закончилась до первого несовпадения, то она считается меньшей. Пустая строка меньше любой строки. ПРИМЕР: Сравнение строк.

```
'abcd' > 'abcD' { 'd'>'D' }
'abcd' > 'abc' { 'd'>'' }
'abc' < 'axxc' { 'b'<'x' }
'abcd' = 'abcd'
```

Переменная строкового типа (String) может рассматриваться как массив элементов символьного типа (Char). Например, если в программе определены переменные S: string; C: char; и задано S:='Москва', то S[1]='M', S[2]='o' и т. д. и возможно присвоение, например: C:= S[1]; Таким образом строка может рассматриваться как линейный массив символов. Элементы массива, составляющие строку можно переставлять местами и получать новые слова, например:

```
for i:= 1 to N div 2 do
  begin
    C:= S[i];
    S[i]:= S[N-i+1];
    S[N-i+1]:= C
    Writeln(S);
```

```
    end;    { исходное слово выведется справа налево:  "авксоМ" }
```
Здесь N:= ord(S[0]); - число символов в переменной "S" хранится в переменной S[0]. Функция "ord" преобразует символьный тип в целый. N div 2 - количество перестановок для слова из "N" символов. В переменной "C" запоминается значение i-го элемента, который меняется с элементом, симметричным относительно середины строки.

Можно производить поиск и замену заданного символа в строке, например:
```
  for i:=1 to N do if S[i]=' ' then writeln      ('найден символ
пробел');
  for i:=1 to N do if S[i]='/' then S[i]:='\'; {замена символа "/" на
"\"}
```
Заменяя или переставляя символы в строке по определенной схеме (закону) можно зашифровать строку. Для дешифровки используется, как правило, схема обратной перестановки или замены символов. Например:
```
  for i:=1 to N do S[i]:= chr(ord(S[i])+2);
  {преобразование исходных символов  в символы с кодом большим на две
единицы}
```
Напомним, что все используемые в MS-DOS символы имеют ASCII коды от 0 до 255. Здесь удобно также использовать функции Pred(C); и Succ(C);
Существует ряд стандартных функций и процедур для работы со строками.

- Функция Length(s) выдает длину строки s.
- Функция Concat(s1,s2,..,sn) возращает строку s1+s2+..+sn.
- Функция Copy(s,p,k) возвращает фрагмент строки s, который начинается в позиции p и имеет длину k.
- Функция Pos(s1,s) ищет первое вхождение подстроки s1 в строку s и возвращает номер первого символа s1 в строке s или 0 если не нашли.
- Процедура Delete(s,p,k) удаляет из строки s фрагмент, который начинается в позиции p и имеет длину k.
- Процедура Insert(s,s1,p) вставляет в строку s1 подстроку s, начиная с заданной позиции p.

Турбо паскаль позволяет производить преобразования числовых значений в строковые и наоборот. Для этого используются процедуры Str(X:n:d,S) и Val(S,X,e). Первая получает их числа X строку S с изображением этого числа, в которой не менее n символов и из них d знаков после запятой. Параметры n и d необязательные. Вторая процедура получает из строки S число X. При успешном результате e=0.
ПРИМЕР: Работа со строками.
```
  var s,x,y,z : string;
begin
  x := 'turbo';
  y := 'pascal';
  z := x+' '+y; { z='turbo pascal' }
  s := ''; { пустая строка }
  for c:='a' to 'z' do s:=s+c; { s='abcd..xyz' }
  writeln(s);
end.
```

## Примеры

**Пример1.** Дан текст, слова в котором, могут разделяться пробелами, запятыми, точками и т.д. Требуется напечатать все слова с удвоенной буквой "н".
***Этапы решения задачи:***

1. Разобьем задачу на несколько блоков
   а) Формирование тела программы, объявление переменных;
   б) Ввод текста;
   в) Очистка текста от "ненужных" символов до первого слова;
   г) Вычисление длины первого слова;
   д) Поиск в слове буквы "н";
   е) Подсчет стоящих рядом букв "н";
   ж) Печать найденного слова;
   з) Удаление первого слова;
   и) Если текст не закончился возвращение к пункту (в).
2. Реализуем эти блоки на Паскале

```
а)

  program example1;
   var st, st1:string;
     i,j,k,n:integer;
      flag:boolean;
     const
   znak=[' ','.',',',':',';','!','?'];
 begin
end.
Назначение переменных:
       t- содержит введенный текст
           st1 - хранит первое слово текста
               i,j,k,n - вспомогательные переменные
             flag - указывает, что данное слово искомое
б)          writeln('Введите текст');
                readln(st);
в)         repeat
               while st[1] in znak do delete(st,1,1);
г) i:=1        while (not (st[i] in znak)) and (i<=length(st)) do
inc(i);
                st1:=copy(st,1,i-1);
         flag:= false;
д)     while (pos('н',st1)>0) and (not flag) do begin
е)     j:=pos('н',st1); n:=j; k:=0;
       while st1[n]='н' do begin inc(n); inc(k); end;
       if k=2 then flag:= true;
       delete(st1,j,k)
  end;
ж)    if flag then writeln(copy(st,1,i-1));
з)     delete(st,1,i);
и)    until st='';

Приведем программу целиком:
program example1;
var st, st1:string;
       i,j,k,n:integer;
       flag:boolean;
const
     znak=[' ','.',',',':',';','!','?'];
   begin
     writeln('Введите текст');
     readln(st);
     repeat
```

```
          while st[1] in znak do delete(st,1,1);
        i:=1;
      while (not (st[i] in znak)) and (i<=length(st)) do inc(i);
      st1:=copy(st,1,i-1);
      flag:= false;
      while (pos('н',st1)>0) and (not flag) do begin
        j:=pos('н',st1); n:=j; k:=0;
           while st1[n]='н' do begin inc(n); inc(k); end;
        if k=2 then flag:= true;
        delete(st1,j,k)
  end;
  if flag then writeln(copy(st,1,i-1));
     delete(st,1,i);
   until st='';
     readln;
end.
```

## Контрольные вопросы

1. Как описываются строковые переменные?
2. Какая максимальная длина строки допустима в Pascal?
3. Какие операции допустимы над строковыми данными?
4. В чем отличие строковой переменной от массива символов?
5. Какие стандартные процедуры и функции для работы со строками вы знаете?
6. Что выведет функция Copy(x,Pos(' ',x)+1,18), если x='Сила есть - ума не надо'?
7. Чему равно значение x[0] после присваивания x:='вопрос'?

## Задания

1. Вывести строку длины N (N - четное), которая состоит из чередующихся символов C1 и C2, начиная с C1.
2. Дана строка. Вывести строку, содержащую те же символы, но расположенные в обратном порядке.
3. Дана строка. Если она представляет собой запись целого числа, то вывести 1; если вещественного (с дробной частью), то вывести 2; если строку нельзя преобразовать в число, то вывести 0.
4. Дана строка S и число N. Преобразовать строку S в строку длины N следующим образом: если длина строки S больше N, то отбросить первые символы, если длина строки S меньше N, то в ее начало добавить символы "." (точка).
5. Даны два числа: N1 и N2, и две строки: S1 и S2. Получить из этих строк новую строку, объединив N1 первых символов строки S1 и N2 последних символов строки S2.
6. Даны две строки: S1 и S2. Определить количество вхождений строки S2 в строку S1.
7. Даны строки S1, S2 и символ C. После каждого вхождения символа C в строку S1 вставить строку S2.
8. Даны две строки: S1 и S2. Удалить из строки S1 все подстроки, совпадающие с S2. Если таких подстрок нет, то вывести S1 без изменений.
9. Даны три строки: S1, S2, S3. Заменить в строке S1 первое1|последнее2|все3 вхождения строки S2 на S3.

10. Дана строка, состоящая из русских слов, разделенных пробелами (одним или несколькими). Определить количество слов в строке.
11. Дана строка, состоящая из русских слов, разделенных пробелами (одним или несколькими). Определить количество слов, которые а) начинаются и заканчиваются одной и той же буквой б) содержат хотя бы одну букву "А".
12. Дана строка, состоящая из русских слов, разделенных пробелами (одним или несколькими). Определить длину самого короткого и длинного слова.
13. Дана строка, состоящая из русских слов, разделенных пробелами (одним или несколькими). Вывести строку, содержащую эти же слова (разделенные одним пробелом), но расположенные в обратном порядке.
14. Дана строка-предложение на русском языке. Подсчитать количество содержащихся в строке знаков препинания.
15. Дана строка-предложение, содержащая избыточные пробелы. Преобразовать ее так, чтобы между словами был ровно один пробел.
16. Дана строка, содержащая полное имя файла, то есть имя диска, список каталогов (путь), собственно имя и расширение. Выделить из этой строки имя файла.
17. Дана строка, содержащая полное имя файла. Выделить из строки название последнего каталога (без символов "\"). Если файл содержится в корневом каталоге, то вывести символ "\".
18. Дана строка-предложение. Зашифровать ее, поместив вначале все символы, расположенные на четных местах, а затем, в обратном порядке, все символы, расположенные на нечетных местах (например, строка "Программа" превратится в "ргамамроП").
19. Дана строка, содержащая несколько круглых скобок. Если скобки расставлены правильно (то есть каждой открывающей соответствует одна закрывающая), то вывести число 0. В противном случае вывести или номер позиции, в которой расположена первая ошибочная закрывающая скобка, или, если закрывающих скобок не хватает, число -1.

***Обработка текста:*** В следующих заданиях под словом "текст" понимается строка символов, слова в которой, разделены пробелами, ",",".","!","?",";",":" (одним или несколькими).

20. Дан текст. а) Подсчитать количество слов в данной строке. б) Подсчитать количество букв а в последнем слове данной строки. в) Найти количество слов, начинающихся с буквы б. г) Найти количество слов, у которых первый и последний символы совпадают между собой. д) Найти длину самого короткого слова.
21. Составить программу циклической перестановки букв в словах текста так, что i-я буква слова становится i+1-ой, а последняя - первой.
22. В каждом слове текста замените "а" на букву "е", если "а" стоит на четном месте, и заменить букву "б" на сочетание "ак", если "б" стоит на нечетном месте.
23. Гжатск получил новое название - город Гагарин. А в рязанской областной типографии еще не просохли гранки небольшой книги о родине первого космонавта. Конечно, книгу нужно было переделать... Написать программу, осуществляющую в некотором тексте замену слова "Гжатск" словом "Гагарин" (учесть, что слова имеют разную длину!)
24. Дан текст, содержащий от 2 до 30 слов, в каждом из которых от 2 до 10 латинских букв; между соседними словами - не менее одного пробела. Напечатать все слова, отличные от последнего слова, предварительно преобразовав каждое из них по

следующему правилу: 1) перенести первую букву в конец слова; 2) перенести последнюю букву в начало слова.

25. Отредактировать заданное предложения текста, удаляя из него все слова с нечетными номерами и переворачивая слова с четными номерами. Например, HOW DO YOU DO -> OD OD
26. Дан текст. Напечатать все слова, отличные от последнего слова, предварительно преобразовав каждое из них по следующему правилу: 1) оставить в слове только первые вхождения каждой буквы; 2) если слово нечетной длины, то удалить его среднюю букву
27. Написать программу для подсчета суммы мест, на которых в словах текста стоит заданная буква.
28. Составить таблицу слов данного текста, начинающихся с буквы "А", с указанием числа повторений каждого слова.
29. Составить программу для вычеркивания из слов текста всех букв, стоящих на нечетных местах после буквы "а". Задачи на смекалку
30. Составить программы для перевода арабских чисел в римские и для обратной операции. Например, 255 = CCLV = сто + сто + пятьдесят + пять Замечание. Подобными алгоритмами перевода чисел из одной системы в другую мы пользуемся по нескольку раз на дню, когда ведем денежные расчеты. Сумма денег - это арабское число, которому соответствует определенный набор банкнот и монет (аналоги римских цифр).
31. Автоморфными называются числа, которые содержатся в последних разрядах их квадрата. Например:, 52=25, 252=625. Составить программу для нахождения нескольких автоморфных чисел.
32. Подсчитать, сколько букв надо исправить в слове X, чтобы получилось слово Y (X,Y - слова одинаковой длины).
33. Какое минимальное число букв необходимо заменить в слове X с тем, чтобы оно стало перевертышем?
34. Составить программу для подсчета числа одинаковых букв в словах X и Y равной длины, стоящих на одних и тех же местах.
35. Задано определенное количество конкретных сочетаний букв (например, УЩ, ЮЩ и др.). Определить, сколько таких групп символов содержится в тексте, вводимом с клавиатуры.
36. С клавиатуры вводится текст. Подсчитать и вывести на печать количество слов текста, начинающихся с гласной.
37. Для запоминания числа p иногда используют "магические" фразы, например: "это я знаю и помню прекрасно Пи многие знаки мне лишни напрасны" или "кто и шутя и скоро пожелаетъ Пи узнать число ужъ знаетъ". Число букв в каждом слове любой из данных фраз представляет собою некоторую цифру числа : "это"-3, "я"-1, "знаю"-4 и т.д. Составить программу, которая по указанному алгоритму будет выводить на печать число, используя любой текст.
38. Для заданного текста определить длину содержащейся в нем максимальной серии символов, отличных от латинских букв.
39. Записать программу, выясняющую, можно ли из букв слова X составить слово Y.

**Задачи повышенной сложности**

1. Зашифровать введенную с клавиатуры строку, поменяв местами первый символ со вторым, третий с четвертым и т. д. Затем провести дополнительную шифровку результата смещением кода. Провести дешифровку.

2. Составить процедуру создания текстового окна, окаймленного рамкой из псевдографических символов. В параметры процедуры ввести координаты левого верхнего угла, размеры и цвет окна, а также цвет рамки.
3. Составить программу, организующую перемещение текстового окна 8x8 по экрану. См. задачу 2. Движение начинается по нажатию клавиши и заканчивается либо по нажатию клавиши, либо при достижении окном края экрана. Варианты движения: а) из левого верхнего угла в правый нижний угол. При неточном "попадании" в нижний угол смещать окно по одной из сторон до точной остановки в углу. б)из левого нижнего угла в правый верхний. в) из центра экрана к одной из боковых сторон. При достижении края размер окна по направлению движения должен уменьшаться до минимального.

### Лабораторная работа №4,5,6,7.

**ТЕМА: составление алгоритмов и программ с использованием внутренних процедур и функций.**

**Цель работы:** познакомиться с понятиями "процедура" и "функция" в языке программирования Pascal, рассмотреть их сходства и различия, закрепить практические навыки работы с системой TURBO Pascal на примере реализации алгоритмов при помощи процедур и функций, научиться применять метод последовательной детализации в практическом программировании; применять процедуры и функции при решении задач.

# Процедуры и функции

Практически во всех алгоритмических языках имеется возможность программирования функций и процедур - блоков операторов, оформленных в виде подпрограмм. Разработка функций и процедур необходима при многократном использовании в разных местах программы или в нескольких программах блока операторов, выполняющих однотипные действия, например, расчет значений сложной функции при различных значениях аргумента. В Турбо - Паскале имеется также возможность создавать библиотеки (модули), состоящие из специальных процедур и функций, отличных от поставляемых в пакете ( модули System, Crt, Graph).
Процедуры (подпрограммы) и функции, определяемые программистом, приводятся в разделе описания основной программы. Процедуры и функции имеют заголовок, раздел описания и раздел операторов.
**Заголовок процедуры** состоит из служебного слова **Procedure**, имени процедуры и списка параметров,

**например:**
```
Procedure Name_P(p1, p2,...: "тип";  Var p3, p4,...: "тип";...);
```
Заголовок функции состоит из служебного слова Function, имени функции и списка параметров, кроме того указывается тип возвращаемого функцией значения,

**например:**
```
Function Name_F("список формальных параметров"):"тип результа-та";
```

Здесь:
**Function** и **Procedure** - служебные слова,
**Name_F, Name_P** - имена функции и процедуры соответственно,
**p1, p2** - имена формальных параметров-значений,
**p3, p4** - имена формальных параметров-переменных,
**. . .** - многоточие означает возможность перечисления большего числа параметров.

В дальнейшем, если не оговаривается особо, все сказанное к процедуре относится также и к функции.

Тип возвращаемого функцией значения может быть простым, строковым или типом-указателем. Тип формальных параметров может быть любым, но должен указываться только идентификатором (именем типа). Таким образом, имя типа формального параметра - массива должно быть задано предварительно в операторе Type, например: Type M= array[1..100]of real; Затем тип массива может указываться в заголовке процедуры, например: Procedure Name_P(p: M); Тип формальных параметров описывается только в заголовке процедуры. Список формальных параметров может отсутствовать, например. процедура Randomize; не имеет параметров.
Если в результате выполнения нескольких операторов получается одно значение переменной, то эти операторы можно включить в описание функции. Например, функция Sin(x); возвращает значение, которое присваивается переменной Y:=sin(x); (эта, и другие стандартные функции описаны в модуле System, который подключается к программе автоматически).
Если в результате выполнения нескольких операторов производится некоторое действие или расчет нескольких переменных, то эти операторы лучше включить в описание процедуры. Например, процедура ClrScr; из модуля CRT очищает экран.

Вызов процедуры осуществляется в разделе выполнения основной программы или других процедур (вложенные процедуры). Программа (процедура) внутри которой вызывается другая процедура называется внешней по отношению к вызываемой процедуре.
При вызове процедуры вместо формальных параметров подставляются фактические параметры, значения которых используются в процедуре.

**Например:**

```
Name_P(p11, p22,..., p33, p44,...);  - вызов процедуры Name_P,
Y:= Name_F("список фактических параметров"):  - вызов функции Name_F,
```

Здесь:
**p11, p22, . . .** - имена или значения переменных,
**p33, p44, . . .** - имена переменных, значения которых возвращаются в программу.
**Y** - переменная, которой присваивается значение возвращаемое функцией.

Типы соответствующих формальных и фактических параметров должны совпадать, а имена могут совпадать или быть различными. Вместо параметров-значений можно подставлять имена переменных, значения переменных или выражения, вместо параметров-переменных подставляются имена переменных. Функция и параметры-переменные возвращают во внешнюю программу значения, полученные после окончания работы функции или процедуры. Изменения параметров-значений в процедуре носит локальный характер, во внешней программе соответствующие фактические параметры не изменяются. Если не требуется передавать во внешнюю программу новые значения, то следует использовать параметры-значения, а не параметры-переменные.

В процедуре можно использовать локальные метки, константы и переменные, описав их в разделе описания процедуры. Локальные имена не должны совпадать с именами формальных параметров, а их значения не передаются во внешнюю программу. Метки, константы и переменные, описанные во внешней программе раньше, чем сама процедура, называются глобальными по отношению к вызываемой процедуре. Если локальные и глобальные имена совпадают, то в процедуре используются локальные значения, а во внешней программе - глобальные значения, т. е. локальные и глобальные идентификаторы независимы. Если имя глобальной переменной уникально (в процедуре не описывается переменная с таким же именем) и ее значение в процедуре изменяется, то оно изменяется и во внешней программе. Вызывая в программе процедуру программист использует ее имя и параметры не анализируя, а часто и не зная содержимого процедуры. Поэтому в целях универсальности процедур следует все значения в процедуру передавать через список параметров, а переменные внутри процедуры описывать, т. е. делать их локальными.

Приведем пример процедуры вывода на экран визитной карточки программиста.

```
Program NP_1;
Var
  Dat, Fam: string;  { Fam: глобальная переменная }
Procedure VIZ(D_R :string);           { D_R  - формальный параметр }
Var
  S_t: string;{ S_t: локальная переменная }
Begin
   Writeln('| ------------------------------------------ |');
   Writeln('|  Разработчик программы:',    Fam:14,'         |');
   Writeln('|                                             |');
   Writeln('|  г.  Анжеро-Судженск   ',    D_R:14,'         |');
   Writeln('|  Телефон:   2-99-76                         |');
   Writeln('| -------------------------------------------|');
   Write('   Комментарий: ');
   Readln(S_t)
end;
Begin
  Fam:='И.И.Иванов';
  Dat:='06.12.95';    {Dat - фактический параметр}
  VIZ(Dat);           { вызов процедуры }
  Readln;
END.
```

Если процедура описана в другом файле с именем, например, F_PR. pas, то ее можно подключить к программе, указав в разделе описания директиву: **{$I F_PR. pas}**

Приведем пример использования стандартных процедур модуля DOS для вывода текущей даты и времени:

```
uses DOS;                                          { подключение модуля
DOS }
Procedure Date_Time;
var y, m, d, d_w:word;  h, min, sec, hund: word;{локальные параметры }
begin
  GetDate(y,m,d,d_w); {вызов процедуры DOS,  возвращающей параметры
даты }
  GetTime(h,min,sec,hund);  { процедура,  возвращающая параметры
времени }
  writeln('сегодня: ' );
  writeln('_':10,   d, ' число');
  writeln('_':10,   m, ' месяц');
  writeln('_':10,   y, ' год'  );
  writeln('день недели: ', d_w );    { d_w= 0 - воскресенье,  и т. д. }
  writeln('Время: ' );
  writeln('_':6,   h, ' часов' );
```

```
  writeln('_':6,   min, ' минут' );
  writeln('_':6,   sec, ' секунд' ); readln
end;
Begin
  Date_Time
end.
```

В практических задачах часто пишутся процедуры, возвращающие значения элементов массивов. Приведем пример процедуры расчета "N" значений функции, например, Y= 4*Sin(x)+7*Cos(x); в заданном диапазоне x1<=x<=x2, при N<=100 и равномерной разбивке диапазона.

```
type r_1000= array[1. . 1000] of real;              { задается тип
r_1000 }
var Z: r_1000;  x1, x2: real;   n: word;
Procedure Mas_Y(var Y:r_1000; x1,x2:real; n:word); {Y - параметр-
перемен-ная}
var i: word;   x,  dx: real;                         { локальные
параметры }
begin
    If (n>1000) or (n<2) then
      begin
        writeln('Длина массива >1 и не должна превышать 1000');
        Readln;
        Halt
      end;
    i:=0;
    x:=x1;
    dx:=(x2-x1)/(n-1);     { dx - шаг изменения аргумента }
    If dx<= 0 then
      begin
        writeln('x2 должно быть больше x1');
        Readln;
        Halt
      end;
  While x<="" pre="">
```

Здесь тип формального параметра "Y" задается в разделе описания типов внешней программы и совпадает с типом фактического параметра "Z", значения элементов которого возвращаются во внешнюю программу.

Оператор Halt прерывает выполнение всей программы,  даже если он используется внутри процедуры.  Применение оператора Exit внутри процедуры вызывает прерывание процедуры,  но не внешней программы.
Приведем пример процедуры вывода массива чисел в файл:

```
Type M_30x30_r= array[1..30, 1..30] of real;  { задается тип  M_30x30_r
}
var  x: M_30x30_r;
     i, j, n, m: byte;
{-------------------------------------------------------------------------
--}
Procedure Wr_M(a: M_30x30_r;  name_f: string;  n, m: byte);
Var
  i, j: byte;                          { a - массив NxM,   n<=30,  m<=30
}
  f: text;                             { name_f - имя файла }
begin
  assign(f, name_f);
  rewrite(f);
  For i:= 1 to n do
    begin
      writeln(f);
      For j:= 1 to m do write(f, a[i,j]:6:2)
    end;
```

```
  close(f)
end;
{-------------------------------------------------------------------------
--}
Begin
  N:= 10;                          { создание симметричной матрицы }
  for i:= 1 to N do
    for j:= i to N do
      x[i, j]:= 0.5 + random(50); { заполнение верхней треугольной
матрицы }
  for i:= 1 to N do
    for j:= i to N do
      x[j,i]:= x[i,j]; { заполнение нижней,  симметричной части матрицы
}
End.
```

Для правильного считывания данных, записанных в файл бесформатным выводом необходима запись пробела для разделения чисел.

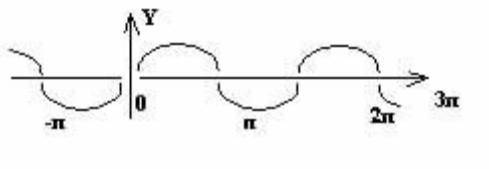
Приведем пример функции для расчета высоты треугольника по заданным значениям его сторон.

```
Program TR;
Var a, b, c, ha, hb, hc: real;
{-------------------------------------------------------------------------
------}
Function  H_TR(a, b, c: real): real;        { a, b, c - Стороны
треугольника }
Var p, s: real;
Begin
  If (a<0) or (b<0) or (c<0) Then
    begin
      Writeln('Стороны треугольника >0 ?');
      readln;
      Halt
    end;
  If (a>(b+c)) or (b>(a+c)) or (c>(a+b)) Then
    begin
      Writeln('a<(b+c),  b<(a+c),  c<(a+b) ?');
      readln;
      Halt
    end;
  p:= (a+b+c)/2;                   { полупериметр }
  s:= Sqrt(p*(p-a)*(p-b)*(p-c));   { площадь }
  H_TR:= 2*s/a;                    { Присвоение функции значения }
End;
{-----------------------------------------------------------------}
Begin
  Writeln('Введите значения сторон треугольника a,b,c');
  Readln(a,b,c);
  ha:= H_TR(a, b, c);
  hb:= H_TR(b, a, c);
  hc:= H_TR(c, b, a);
  Writeln('Высоты  треугольника:');
  Writeln('ha=',ha:-10:4, 'hb=',hb:-10:4, 'hc=',hc:-10:4);
  Readln
End.
```

В программе трижды вызывается функция расчета высоты треугольника для различных комбинаций фактических параметров, что позволяет вычислить все высоты треугольника.

Приведем пример использования функции для расчета суммы членов степенного ряда, представляющего тригонометрическую функцию Y= Sin(x).

```
PROGRAM fun_sin;
var  y,  y1,  x1: real;
{------------------------------------------------------------------------
---}
Function Sin_r( x: real): real;
Var a,  k,  y: real;  i: longint;
begin
  if abs(x) > 2*Pi Then x:= 2*pi*Frac(x/(2*Pi));     {учет
периодичности }
  if abs(x) > 2*Pi Then x:= x - 2*pi*Int(x/(2*Pi));            {
функции }
  if abs(x) > Pi Then x:= Pi - ABS(x);         { учет асимметрии
функции }
  i:= 0;  a:= x;  y:= a;
  while abs(a)>0.0000001 do
    begin
      i:=i+1;
      k:=-x*x/(2*i*(2*i+1));
      a:= a*k;
      y:= y + a
    end;
  Sin_r:= y;                      { присвоение функции ее значения }
end;
{------------------------------------------------------------------------
---}
Begin
  write('Введите значение аргумента: x1= ');
  Readln(x1);
  Y:= Sin_r(x1);            { вызов функции,  разработанной
программистом }
  Y1:= Sin(x1);             { вызов стандартной функции }
  writeln('значение аргумента: x1= ',  x1);
  writeln('расчетное значение функции: Sin_r(x1)= ',  y :-11:8);
  writeln('контрольный результат:     Sin(x1) =  ',  y1:-11:8);
  writeln('Нажмите Enter');
  readln;
end.
```




В описании функции обязателен оператор присвоения функции ее значения. Изменение величины параметра-значения "x" в теле функции не отражается на его значении во внешней программе. Функция возвращает значение, которое во внешней программе присваивается переменной "y".

## Контрольные вопросы

1. Для чего нужны в программе процедуры и функции?
2.

3. В чем отличие между процедурой и функцией?
4.
5. Чем отличаются формальные и фактические параметры?
6.
7. Чем отличаются параметры-значения и параметры-переменные?
8.
9. Как объявляются глобальные и локальные переменные? Каково правило видимости этих переменных?
10.
11. Почему при обращении к процедуре, аргумент, передаваемый параметру-переменной, может быть только переменной, а не константой или выражением?
12.

## Пример

**Пример1.** Найти сумму положительных элементов в массиве.

***Этапы решения задачи:***

1. Алгоритм решения довольно прост - в цикле будем "пробегать" массив, сравнивая его ячейки с 0 и суммировать, если они >0.
2.
3. Составим блок-схему программы
4.


Уточним из каких блоков состоит блок*"Суммирование положительных ячеек"*


Содержание этих блоков простое, поэтому не стоит их уточнять.

5. Напишем программу на языке Паскаль
6.

```
program example;
   type
    Tarray = array[1..100] of integer;
     Var v: Tarray;
    N,i,s:integer;
  Procedure vvod_data(var m:Tarray;n:integer);
  Var i:integer;
Begin
  Writeln('Введите ',n,' чисел через пробел');
   For i:= 1 to n do read(m[i]);
End;
  Function summ(m:TArray):integer;
  Var s:integer;
Begin
   S:=0;
    For i:= 1 to n do if m[i]>0 then s:= s+m[i];
   Summ:=s;
End;
begin
      write('Введите размерность массива N= '); readln(n);
     vvod_Data(v,n);
    s:= summ(v);
   writeln('Сумма= ',s);
end.
```

## Задания

1. Описать функции Min2(A,B) и Max2(A,B)2 вещественного типа, находящие минимальное и максимальное из двух вещественных чисел A и B.

2. Описать процедуру Minmax(A,B), записывающую в переменную A минимальное из значений A и B, а в переменную B - максимальное из этих значений.

3. Описать функцию Fact(N) целого типа, вычисляющую значение факториала N! = 1·2·...·N (N > 0 - параметр целого типа). С помощью этой фунции вычислить факториалы 10 данных чисел.

4. Описать процедуру SumDigit(N,S), находящую сумму цифр S целого числа N (N - входной, S - выходной параметр). Используя эту процедуру, найти суммы цифр пяти данных чисел.

5. Описать функцию Polynom(A,N,X) вещественного типа, находящую значение полинома P в вещественной точке X. Полином P задается параметрами N (степень полинома, 0 < N < 8) и A (коэффициенты полинома, вещественный массив размера N+1): P(X) = A[1]·XN + A[2]·XN-1 + ... + A[N]·X + A[N+1]. Используя эту функцию, найти значения заданного полинома в пяти данных точках.

6. Описать функцию Min(A,N) и Max(A,N)2 вещественного типа, находящую минимальный и максимальный элемент массива A, состоящего из N вещественных чисел.

7. Описать функцию NMin(A,N) и NMax(A,N)2 целого типа, находящую номер минимального и максимального2 элемента массива A (массив состоит из N вещественных чисел).

8. Описать процедуру NMinmax(A,N,NMin,NMax), находящую номера минимального и максимального элемента массива A из N вещественных чисел.

9. Описать процедуру Invert(A,N), меняющую порядок следования элементов массива A из N вещественных чисел на противоположный ("инвертирование массива").

10. Описать процедуру Smooth(A,N), заменяющую каждый элемент вещественного массива A размера N на его среднее арифметическое со своими соседями ("сглаживание массива"). С помощью этой процедуры выполнить пятикратное сглаживание данного массива A размера N, выводя на экран результаты каждого сглаживания.

11. Описать функцию SumLine(A,M,N,k) вещественного типа, вычисляющую сумму элементов вещественной матрицы A размера M x N, расположенных в k-й строке (если [k > M]1|[k > N]2, то функция возвращает 0).

12. Описать функцию SumCol(A,M,N,k)2 вещественного типа, вычисляющую сумму элементов вещественной матрицы A размера M x N, расположенных в k-й столбце (если [k > M]1|[k > N]2, то функция возвращает 0).

13. Описать функцию IsIdent(S) целого типа, проверяющую, является ли строка S допустимым идентификатором Паскаля. При утвердительном ответе возвращается 0. Если S является пустой строкой, то возвращается -1, если строка начинается с цифры, то возвращается -2. Если S содержит недопустимые символы, то возвращается номер первого недопустимого символа. Проверить с помощью этой функции пять данных строк.

14. Описать функцию FillStr(S,Len) строкового типа, возвращающую строку длины Len, заполненную повторяющимися копиями строки-шаблона S (последняя копия строки-шаблона может входить в результирующую строку частично).

15. Описать процедуру Trim(S), удаляющую в строке S начальные и конечные пробелы.

16. Описать функцию PosLast(subS,S) целого типа, возвращающую номер позиции, с которой в строке S содержится последнее вхождение подстроки subS. Если в строке S отсутствуют подстроки subS, то функция возвращает 0.

17. Даны действительные числа x1, y1, x2, y2, …,x10, y10. Найти периметр десятиугольника, вершины которого имеют соответственно координаты (x1, y1), (x2, y2), …, (x10, y10). (Определить процедуру вычисления расстояния между двумя точками, заданными своими координатами.)
18.
19. Даны действительные числа a, b, c, d, e - стороны пятиугольника. Найти площадь пятиугольника. (Определить процедуру вычисления площади треугольника по его сторонам.)
20.
21. Даны три символьные матрицы.
22.
a) ту матрицу, где есть хотя бы одна гласная - транспонировать;
23.
b) в той матрице, на главной диагонали которой все цифры, найти наименьшую и удалить соответствующую строку.

24.
25. Написать программу вычисления P по формуле: где n - заданное натуральное число.
26.
27. Описать функцию Stepen (x,n) от вещественного x и целого n, вычисляющую (посредством умножения) величину xn, и использовать ее для вычисления b=2.7k+(a+1)-5.
28.
29. Даны отрезки a,b,c и d. Для каждой тройки этих отрезков, из которых можно построить треугольник, напечатать площадь данного треугольника. Определить процедуру Plo(x,y,z), печатающую площадь треугольника со сторонами x,y и z, если такой треугольник существует.
30.
31. Пусть процедура Socr(a,b,p,q) от целых параметров (b№0)приводит дробь $\frac{a}{b}$ к несократимому виду $\frac{p}{q}$ Описать данную процедуру и использовать ее для приведения дроби $1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\Lambda+\frac{1}{20}$ к несократимому виду $\frac{c}{d}$
32.
33. Даны длины a,b и c сторон некоторого треугольника. Найти медианы треугольника, сторонами которого являются медианы исходного треугольника. Длина медианы, проведенной к стороне a, равна $\frac{1}{2}\sqrt{2b^2+2c^2-a^2}$
34.
35. Даны координаты вершин двух треугольников. Определить, какой из них имеет большую площадь.
36.
37. Даны координаты вершин треугольника и координаты некоторой точки внутри него. Найти расстояние от данной точки до ближайшей стороны треугольника. (При определении расстояний учесть, что площадь треугольника вычисляется и через три его стороны, и через основание и высоту.).
38.
39. Три прямые на плоскости заданы уравнениями akx+bky=ck, k=1,2,3. Если эти прямые попарно пересекаются и образуют треугольник, тогда найти его площадь.
40.
41. Два натуральных числа называются "дружественными", если каждое из них равно сумме всех делителей другого, за исключением его самого (таковы, например, числа 220 и 284). Напечатать все пары "дружественных" чисел, не превосходящих заданного натурального числа.
42.
43. Дано четное число n > 2. Проверить для этого числа гипотезу Гольдбаха. Эта гипотеза (по сегодняшний день не опровергнутая и полностью не доказанная) заключается в том, что каждое четное n, большее двух, представляется в виде суммы двух простых чисел. Воспользоваться функцией распознавания простых чисел.
44.
45. Дано натуральное число n. Выяснить, является ли оно полным квадратом. Определить функцию, позволяющую распознавать полные квадраты.
46.
47. Дан массив A[1..50], элементы которого отличны от нуля. Расположить их в таком порядке, чтобы первыми были все положительные элементы, а затем - все отрицательные, причем порядок следования как положительных, так и отрицательных

элементов должен сохраниться (при решении задачи новый массив не заводить!).

48.

49. Преобразовать массив S, "поворачивая" его вокруг центра на 90, 180, 270 градусов против часовой стрелки.

50.

51. Рассматривая массивы X, Y и Z как представление некоторых множеств из объектов типа индекс (X[k]=TRUE, если элемент k принадлежит множеству X, и X[k]=FALSE иначе, и т.п.), реализовать следующую операцию над этими массивами-множествами: переменной t присвоить значение TRUE, если множество X является подмножеством множества Y, и значение FALSE иначе.

52.

53. Элемент двухмерного массива называется локальным минимумом, если он строго меньше всех имеющихся у него соседей. Подсчитать количество локальных минимумов заданной матрицы размером NxN найти максимум среди всех локальных минимумов.

54.

55. Составить функцию для нахождения точного значения суммы натуральных чисел, в десятичной записи которых более 20 знаков. Указание. Исходные данные и ответ представить в виде массивов цифр.

56.

57. Даны две строки символов. Символ будем называть общим, если он встречается и в первой, и во второй строке. Пусть K1 - число вхождений в первую строку общего символа, а K2 - во вторую. Минимальное из чисел K1, K2 будем называть числом общности. Вывести все общие символы с указанием для них числа общности.

58.

**Задачи повышенной сложности**

1. Рассмотрим произвольное натуральное число и найдем сумму его цифр, затем сумму цифр полученного числа и так далее, пока не получим однозначное число. Назовем это число цифровым корнем. Требуется написать программу, которая для заданного N (N<10100) находит его цифровой корень.
2. 
3. Задано N натуральных чисел a1,a2,…,aN (1J N J 20), каждое из которых находится в интервале от 1 до 10000. Необходимо определить количество натуральных делителей произведения a1*a2*…*aN.
4. 
5. Написать функцию поиска корней полинома степени n. Исходными параметрами будут числа a1, a2,…,an. Комплексные корни учитывать.
6. 
7. Требуется написать программу, которая выводит в порядке возрастания все правильные несократимые дроби, знаменатели которых не превосходят N (2<=N<=500).
8. 
9. На экране компьютера, работающего в операционной системе Windows, было открыто N (N <= 20) окон, положение каждого из которых однозначно определяется четверкой натуральных чисел - X1 Y1 X2 Y2 - координатами левого верхнего и правого нижнего угла окна. Очевидно, что окна, открытые позже, могут частично или полностью перекрывать открытые ранее. Окно считается видимым, если виден хотя бы один образующий его пиксел.

## Лабораторная работа № 8

## Тема: Создание и использование собственного модуля

**Цель работы**: формирование знаний и умений по работе с модулями. Приобретение навыков создания собственных модулей.

**Краткие теоретические сведения**

Богатство алгоритмических возможностей Паскаля в значительной степени достигается благодаря использованию модулей.

*Модуль* представляет собой набор констант, типов данных, переменных, процедур и функций. Каждый модуль по своей структуре аналогичен *отдельной программе*.

*Модуль - программная единица, текст которой компилируется независимо (автономно).*

Вместе с тем, структура модуля позволяет использовать его как своеобразную *библиотеку описаний*. Модули являются достаточно гибким и удобным инструментальным средством при разработке больших программах комплексов рамках совместной технологии разработки программного обеспечения (структурное программирование и др.).

Кроме того, использование модулей позволяет практически обойти известное для 16-разрядной ПЭВМ *ограничение на размер кодового сегмента* (как известно, размер *кодового сегмента* отдельной программы не должен превышать *64 Кбайт*). Это достигается благодаря тому, что каждому модулю при выполнении при выполнении программы отводится свой отдельный сегмент оперативной памяти.

Паскаль располагает 8-мью стандартными (встроенными) модулями. Это *System, Dos, Overlay, Graph, CRT, Printer, Turbo3, Graph3*. Два последних модуля предназначены для поддержки совместимости программ, написанных на Турбо-Паскале версии 3.0.

Все перечисленные стандартные модули (кроме Graph, Graph3, Turbo3) объединены и сохранены в файле *TURBO.TPL.*

Модуль *SYSTEM* поддерживает все стандартные процедуры и функции, обеспечивает ввод-вывод данных, обработку строк, динамическое распределение оперативной памяти и ряд других возможностей Турбо-Паскаля. Модуль *SYSTEM* подключается к любой программе автоматически.

Модель *DOS* содержит многочисленные процедуры и функции, многие из которых по своему действию эквивалентны командам MS-DOS (*GetTime*, *DiskSize* и др.).

Модуль *OVERLAY* обеспечивает поддержку систем оверлеев.

Модуль *CRT* поддерживает ряд стандартных процедур и функций, которые обеспечивают работу с экраном дисплея в текстовом режиме, управление звуком и работу с клавиатурой.

Модуль *PRINTER* определяет драйвер печатающего устройства и позволяет организовывать вывод информации на принтер.

Модуль *GRAPH* обеспечивает работу с экраном дисплея в графическом режиме.

Наряду с использованием стандартных модулей каждый программист имеет возможность организации собственных модулей.

*Структура любого следующего модуля имеет вид:*

*ЗАГОЛОВОК МОДУЛЯ*

**Unit** < имя модуля >;

*ИНТЕРФЕЙСНАЯ ЧАСТЬ*

**Interface**

**Uses** < список используемых модулей >

{ открытые объявления}

**Type**

**Var**

**Procedure**

**Function**

*РЕАЛИЗАЦИОННАЯ ЧАСТЬ*

**Implementation**

**Uses** < список используемых модулей >

{ собственные объявления }

**Type**

**Var**

{ процедуры и функции }

**Procedure**

**Function**

*ИНИЦИАЛИЗАЦИОННАЯ ЧАСТЬ*

**Begin**

**…** {Основной блок модуля}

**End**.

Имя модуля записывается за ключевым словом *UNIT*. При выборе имени модуля необходимо учитывать одну особенность: *имя модуля должно совпадать с именем файла, в котором он хранится.*

Далее записывается раздел интерфейсной части (за ключевым словом *Interface*). Эта часть модуля является доступной (“видимой”) для любой программы, использующей этот модуль. То есть объявленные в этом разделе константы, типы данных, переменные, процедуры и функции, могут использоваться в любой другой программе. В свою очередь, в разделе интерфейса могут указываться другие используемые модули (их список следует за ключевым словом *Uses*). При этом все объекты, объявленные в интерфейсах этих модулей могут быть использованы в любом объявлении в интерфейсе данного модуля.

*Примечание.* Для “видимых” процедур и функций в интерфейсном разделе приводится только их заголовки. Полностью эти процедуры и функции записываются в разделе реализации.

Следующим структуре модуля описывается раздел реализации (за ключевым словом *Implementation*). В этом разделе могут использоваться все объекты, описанные в разделе интерфейса. Вместе с тем, здесь могут объявляться свои константы, типы данных, переменные процедуры и функции. Они могут быть использованы *только в данном разделе реализации* и в этом смысле являются “не видимыми”. Это же ограничение относится и к интерфейсам других модулей, список которых следует за ключевым словом *Uses* (в отличие от аналогичного списка в разделе интерфейса данного модуля). Таким образом, различие всех описаний содержащихся в разделе

интерфейса и реализации заключается в сфере их использования (первые - доступны извне, вторые -только внутренние).

Раздел реализации модуля начинается ключевым словом *Implementation* и заканчивается *end*. Но если между ними появляется ключевое слово *begin*, то получившийся составной оператор *begin…end* становится разделом инициализации модуля. Раздел инициализации обычно используется для открытия файлов и для формирования значений структур данных и переменных.

При выполнении программы, использующей модули, их разделы инициализации вызываются раньше загрузки самой программы. Для того чтобы использовать модули в программах их имена следует указать в разделе подключения модулей *Uses*.

Например, вы разработали программу, которая наряду со стандартным модулей *Crt* использует ваш разработанный собственный модуль с именем *Modul.* Тогда, в программе следует указать список используемых модулей в следующем виде:

```
Program Pr;
Uses   Crt, Modul;
. . .
```

Модули транслируются отдельно. В отличие от основных программ результатом трансляции которых будут файлы с расширением *EXE* модули получают расширение *TPU*. Полученные в результате трансляции *TPU - файлы* можно подсоединить к стандартному файлу *TURBO.TPL* с помощью утилиты *TPUMOVER.EXE*. Если этого не делать, то при трансляции самой программы все используемые модули (*TPU - файлы*) присоединяться к ней автоматически. Если в каком-либо из используемых модулей были внесены изменения, то при трансляции программы, все модифицируемые модули также будут предварительно перетранслированы (эту функцию реализует интегрированная среда).

### Пример разработки собственного модуля

Разработку собственного модуля рассмотрим на следующем примере:

Пусть дано задание: разработать личную библиотеку, включив в нее процедуры:

- ввода элементов числовой матрицы размером N*N;
- транспонирования матрицы;
- вывода результирующей матрицы.

В основной программе ввести размер матрицы N.

Начнем разработку модуля, который будет носить название Matrix. Программно это будет выглядеть так:

```
Unit Matrix;
 {Зарезервированное слово Unit служит для указания имени библиотеки. Это имя
 должно совпадать с именем PAS-файла библиотеки (т.е библиотека Matrix должна
 находиться с файле Matrix.Pas), а иначе компилятор даст ошибку при попытке
 использования такой библиотеки}

Interface
{Секция Interface содержит описания общедоступных типов данных, констант,
 процедур и функций. Т.е. все, что будет здесь находиться можно будет
 использовать при подключении данной библиотеки.}

Type
  TMatrix = array [1..10,1..10] of Integer;          { Квадратная матрица }
  procedure MatrInput  (Var m : TMatrix; n : Integer); { ввод матрицы }
  procedure MatrOutput (Var m : TMatrix; n : Integer); { вывод матрицы }
  procedure MatrTransp (Var m : TMatrix; n : Integer); { транспонирование }

Implementation
{Секция Implementation содержит реализацию тел процедур и функций, описанных
 в Interface. Также здесь могут содержаться типы данных, константы, процедуры
 и функции, необходимые для работы, но которые не будут видны программе при
 подключении библиотеки.}
{Процедура обмена местами двух элементов матрицы (x1,y1) и (x2,y2).
 Эта процедура используется при транспонировании матрицы, но ее
 нельзя вызвать при подключении библиотеки, т.к. она не объявлена
 в секции Interface.}
procedure Swap (Var m : TMatrix; x1,y1,x2,y2 : Integer);
 var
   temp : Integer;
 begin
   temp := m[x1,y1];
   m[x1,y1] := m[x2,y2];
   m[x2,y2] := temp;
 end;
```

```
{Ввод матрицы с клавиатуры. Параметры процедуры здесь не указаны, т.к.
они есть в секции Interface }
procedure MatrInput;
 var
   i,j : Integer;
 begin
   for i:=1 to n do
    begin
     Write(i:3,'-я строка : ');
     for j:=1 to n do Read(m[i,j]);
     ReadLn;
    end;
 end;
{Транспонирование матрицы.}
procedure MatrTransp;
 var
   i,j : Integer;
 begin
   for i:=1 to n-1 do
    for j:=i+1 to n do
     Swap (m,i,j,j,i);
 end;
{Вывод матрицы на экран.}
procedure MatrOutput;
 var
   i,j : Integer;
 begin
   for i:=1 to n do
    begin
     Write(i:3,'-я строка : ');
     for j:=1 to n do Write (m[i,j]:4);
     WriteLn;
    end;
 end;
{Эта секция может использоваться для инициализации работы библиотеки.}
Begin
End.
```

Создание модуля закончено. Теперь необходимо создать файл, который будет содержать текст основной программы, в которой будет подключаться разработанный выше модуль.

```
{Это отдельный файл, содержащий основную программу}
Uses
  Crt, {Библиотека стандартных процедур управления экраном и
клавиатурой}
```

```
  Matrix; {Наш разработанный модуль-библиотека работы с квадратными
матрицами (личная)}
Var
  m : TMatrix;  { Объявляем матрицу - максимальный размер 10*10 }
  n : Integer;  { Размер матрицы }
Begin
  { Повторяем ввод размера, пока не будет введено корректное значение}
  repeat
    ClrScr;
    Write('Введите размер матрицы (1..10) : ');
    ReadLn(n);
  until (n >= 1) and (n <= 10);
WriteLn;
WriteLn('Введите матрицу размера',n,'*',n,'по строкам:');
MatrInput (m,n); {вызов процедуры ввода матрицы, определенной в модуле
Matrix}
  {Транспонируем ее }
  MatrTransp (m,n); {вызов процедуры транспонирования матрицы,
определенной в модуле Matrix}
  { Выведем результат на экран }
  WriteLn;
  WriteLn('Транспонированная матрица :');
  MatrOutput (m,n); {вызов процедуры вывода матрицы, определенной в
модуле Matrix}
End.
```

**Порядок выполнения работы**

1. Изучить теоретические сведения по теме “ Модули в Паскале”.
2. Получить у преподавателя индивидуальное задание. Разработать личную библиотеку, включив в нее процедуры, определенные в задании.
3. Показать работающую программу преподавателю.
4. Ответить на контрольные вопросы.

**Контрольные вопросы**

1. Стандартные модули в Паскале.
2. Структура модуля.
3. Ключевые слова Unit, Interface, Implementation. Описание каждого раздела.
4. Концепция разработки собственного модуля. Пример программы.

# Лабораторная работа № 9
# ТЕМА: Составление программ с применением указателей.

Все глобальные переменные и типизированные константы размещаются в одной непрерывной области оперативной памяти, которая называется сегментом данных. Длина сегмента определяется архитектурой процессора 8086 и составляет 64 Килобайта (65536 байт), что может вызвать определённые трудности при описании и обработке больших массивов данных. С другой стороны объём стандартной памяти - 640 Килобайт. Выход - использовать динамическую память.

**Динамическая память** - это оперативная память ЭВМ, предоставляемая Турбо-Паскалевой программе при её работе, за вычетом сегмента данных (64 К), стека (обычно 16 К) и собственно тела программы. По умолчанию размер динамической памяти определяется всей доступной памятью ЭВМ и, как правило, составляет не менее 200 - 300 Кбайт.

Динамическую память обычно используют при:
1. обработке больших массивов данных;
2. разработке САПР;
3. временном запоминании данных при работе с графическими и звуковыми средствами ЭВМ.

Размещение статических переменных в памяти осуществляется компилятором в процессе компиляции.
**Динамические переменные** - размещаются в памяти непосредственно в процессе работы программы. При динамическом размещении заранее неизвестны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным. Турбо-Паскаль представляет средство управления динамической памятью: указатели.

**Указатель** - это переменная, которая в хранит качестве своего значения адрес байта памяти.
В 8086 адреса задаются совокупностью двух шестнадцатиразрядных слов - сегмента и смещения. Сегмент - участок памяти, имеющий максимальную длину 64 К и начинающийся к физического адреса, кратного 16 (то есть 0, 16, 32, 48 и т.д.). Смещение - указывает, сколько байт от начала сегмента нужно пропустить, чтобы обратиться по нужному адресу. Фрагмент памяти в 16 байт называется параграфом. Сегмент адресует память с точностью до параграфа, а смещение - с точностью до байта.

Каждому сегменту соответствует непрерывная и отдельно адресуемая область памяти. Сегменты могут следовать в памяти один за другим, или с некоторыми интервалами, или, наконец, перекрывать друг друга. Таким образом любой указатель по своей внутренней структуре представляет собой совокупность двух слов (типа Word), трактуемых как сегмент и смещение. Указатель адресует лишь первый байт типа данных.

*Справка: процессор 8086.*
8086 имеет 16 - битовые регистры, всего 14 штук, из них:

- 12 - регистры данных и адресов;
- 1 - указатель команд (регистр адреса команды);
- 1 - регистр состояния (регистр флагов).

Регистры данных и адресов делятся на три группы:
• регистр данных;
• регистр указателей и индексов;
• регистр сегментов.

Процессор 8086 всегда генерирует 20-ти бытовые адреса за счёт добавления 16-ти битового смещения к содержимому регистра, умноженному на 16:
*физический адрес = смещение + 16 * регистр сегмента.*
Вместо умножения на 16 содержимое регистра сегмента используется так, как если бы оно имело четыре дополнительных нулевых бита:
**пусть:**

смещение = 10H
регистр сегмента = 2000H

тогда:

| | | | | | |
|---|---|---|---|---|---|
| | 0000 | 0000 | 0001 | 0000 | (смещение) |
| 0010 | 0000 | 0000 | 0000 | (0000) | (номер блока) |
| 0010 | 0000 | 0000 | 0001 | 0000 | (физический адрес) |

физический адрес = 20010H

У 8086 адрес ячейки задаётся номером блока и смещением, что обусловлено тем, что команды 8086 и её данные должны располагаться в разных частях памяти (в разных сегментах). Если требуется адресоваться к данным, то потребуется адресация блока памяти, с которого начинается сегмент данных (из регистра сегмента данных) и позиция желаемой ячейки в этом сегменте (смещение).

В дополнение к области памяти 1 Мбайт, 8086 может адресоваться к внешним устройствам через 65536 (64 К) портов ввода-вывода. Имеются специальные команды ввода-вывода, позволяющие иметь непосредственный доступ к первым 256 (от 0 до 255) портам. Другие команды позволяют получить косвенный доступ к порту с помощью занесения идентифицирующего его номера (0 - 65535) в определенный регистр данных. Подобно ячейке памяти любой порт может быть 8 или 16- битовым.

**Распределение памяти IBM PC.**
• 16 - старших байт - команды начальной загрузки системы.
• Первые 1024 ячейки - вектора прерываний.

**Типы прерываний.**
Существуют 2 вида прерываний - одни можно игнорировать, другие обязательно обслужить как можно скорее.
8086 может распознать 256 различных прерываний, каждому из них однозначно соответствует код типа (целое число от 0 до 255). Код используется в качестве указателя ячейки в области памяти с младшими адресами (область векторов прерываний).

## 1. Объявление указателей

Как правило, в Турбо-Паскале указатель связывается с некоторым типом данных (**типизированные** указатели).

```
Type
  PerPnt = ^PerRec;{указатель на переменную типа PerRec}
  PerRec = Record
```

```
    Name: String;
    Job: String;
    Next: PerPnt;
  End;
Var
  P1: ^Integer;
  P2: ^Real;
```

***Внимание!*** При объявлении типа PerPnt мы сослались на PerRec, который ещё не был объявлен, это связано с тем, что существует исключение из правил для указателей, которые содержат ссылку на ещё неописанный тип данных. Это исключение сделано не случайно, так как динамическая память позволяет использовать организацию данных в виде списков (каждый элемент имеет в своём составе ссылку на соседний элемент - обратите внимание на поле Next).

В Турбо-Паскале можно объявлять указатель и не связывать его с конкретным типом данных:

```
Var
  PP: Pointer;
```

Указатели такого рода называют **нетипизированными**. В них удобно размещать данные, структура которых меняется по ходу работы.

В Турбо-Паскале можно передавать значения только между указателями, связанными с одним и тем же типом данных.

```
Var
  P1,P2: ^Integer;
  P3: ^Real;
  PP:Pointer;
Begin
  P1:= P2; {- верно}
  P1:= P3; {- неверно}
```

но можно сделать так:

```
  PP:= P3;
  P1:= PP;
```

так как ограничение не распространяются на нетипизированные указатели. Но подобные операции часто путают программиста и чреваты смысловыми ошибками.

## 2. Выделение и освобождение динамической памяти

Вся динамическая память – пространство ячеек, называемое кучей. Физически куча располагается в старших адресах, сразу за программой. Указатель на начало кучи храниться в предопределенной переменной HeapOrg, конец - FreePtr, текущую границу незанятой динамической памяти указывает указатель HeapPtr. Для выделения памяти под любую переменную используется процедура New. Единственным параметром является типизированный указатель:

```
Var
```

```
  I,J: ^Integer;
  R: ^Real;
Begin
  New(I); {под I выделяется область памяти,}
  {адрес первого байта этой области помещается в I}
End.
```

После выполнения этого фрагмента указатель I приобретёт значение, которое перед этим имел указатель кучи HeapPtr, а HeapPtr увеличится на два (т.к. он типа Integer); New(R) - вызовет смещение указателя на 6 байт. После того как указатель приобрёл некоторое значение, то есть стал указывать на конкретный байт памяти, по этому адресу можно разместить значения соответствующего типа:

```
  I^:= 2;
  R^:= 2*Pi;
```

Допустима запись: R^:= Sqr (R^) + I^ - 17;
Но недопустима запись: R:= Sqr (R^) + I^ - 17; так как указателю R нельзя присвоить значение вещественного выражения.

Возврат динамической памяти обратно в кучу осуществляется оператором Dispose:
  Dispose(R);
  Dispose(I); - вернут в кучу, ранее забранные 8 байт.
  Dispose(Ptr) не изменяет значения указателя Ptr, а лишь возвращает в кучу память, связанную с этим указателем. Однако повторное применение процедуры к “свободному” указателю приведет к возникновению ошибки времени исполнения. Чтобы указать, что указатель свободен, нужно использовать зарезервированное слово Nil.

К указателям можно применять операции отношения, в том числе и сравнения с Nil:

```
Const
 P:^Real = Nil;
. . . . . .
Begin
 If P = Nil then
   New (P);
. . . . . .
 Dispose(P);
 P:= Nil;
End.
```

Использование указателей, которым не присвоено значение процедурой New или каким-либо другим способом, никак не контролируется системой и может привести к непредсказуемым результатам. Многократное чередование New и Dispose приводит к “ячеистой” структуре кучи. Дело в том, что все операции с кучей выполняется под управлением особой программы, которая ведёт учёт всех свободных фрагментов в куче. При выполнении оператора New( ) эта программа отыскивает минимальный свободный фрагмент, в котором может разместиться требуемая переменная. Адрес начала найденного фрагмента возвращается в указателе, а сам фрагмент или его часть нужной длины, помечаются как занятая часть кучи.

Можно освободить целый фрагмент кучи следующим образом:

1. Перед началом выделения динамической памяти значения указателя HeapPtr запоминается в переменной-указателе с помощью процедуры Mark.
2. Выполнение программы.
3. Освобождение фрагмента кучи от заполненного адреса до конца динамической памяти с использованием процедуры Release.

```
Var
 P, P1, P2, P3, P4, P5: ^Integer;
Begin
 New(P1);
 New(P2);
 New(P3);
 New(P4);
 New(P5);
 Mark(P);
. . . . . .
 Release (P);
End.
```

В этом примере процедурой Mark(P) в указатель P было помещено текущее значение HeapPtr, однако память под переменную не резервировалась.

Вызов Release уничтожает список свободных фрагментов в куче, созданных Dispose, поэтому совместное применение этих процедур не рекомендуется.

Для работы с нетипизированными указателями используются также процедуры GetMem(P, Size) и FreeMem(P, Size) - резервирование и освобождение памяти.
P - нетипизированный указатель, Size - размер.

За одно обращение к куче процедурой GetMem можно зарезервировать до 65521 байт. Освобождать нужно ровно столько памяти, сколько было зарезервировано, и именно с того адреса, с которого память была зарезервирована, иначе программа не будет работать и завершаться корректно!!!

Использование нетипизированных указателей даёт широкие возможности неявного преобразования типов:

```
Var
  i,j: ^Integer;
  r: ^Real;
Begin
  New (i); { I:= HeapOrg; HeapPtr:= HeapOrg+2 }
  j:= i; { J:= HeapOrg }
  j^:=2;
  Dispose (i); { HeapPtr:= HeapOrg }
  New (r); { R:= HeapOrg; HeapPtr:= HeapOrg+6 }
  r^:= Pi;
  WriteLn (j^);
End.
{Будет выведено: "8578"}
{здесь преобразование не имеет никакого смысла}
```

**Примеры использования указателей.**

Динамическая память - 200..300 Кбайт. Нужно разместить массив 100 * 200 типа Extended. Требуется 100 * 200 * 10 = 200000 байт. Пробуем:

```
Var
  i,j: Integer;
  PtrArr: Array[1..100, 1..200] of ^Extended.
Begin
. . . . . .
  For i:= 1 to 100 do
    For j:= 1 to 200 do
     New (PtrArr [i,j]);
. . . . . .
End.
```

Теперь к любому элементу можно обратиться: PtrArr[i,j]^:=...; Но длина внутреннего представления указателей 4 байта, поэтому потребуется ещё 100*200*4 = 80000 байт, что превышает размер сегмента (65536 байт), доступный для статического размещения данных.

Можно было бы работать с адресной арифметикой (арифметикой над указателями), то есть не создавать массив указателей, а вычислять адрес элемента непосредственно перед обращением к нему. Однако в Турбо-Паскале над указателями не определены никакие операции, кроме присваивания и отношения. Но задачу решить можно:

Seg(x) - возвращает сегментную часть адреса.

Ofs(x) - возвращает смещение.

x - любая переменная, в том числе и та, на которую указывает указатель.

Далее с помощью Ptr(Seg, Ofs: Word): Pointer можно создать значение указателя, совместимое с указателем любого типа.

Таким образом, сначала с помощью GetMem забираем из кучи несколько фрагментов подходящей длины (не более 65521). Удобно по строкам 200 * 100 = 20000 байт. Начало каждого фрагмента запоминается в массиве PtrStr из 100 указателей. Теперь для доступа к любому элементу строки нужно вычислить смещение этого элемента от начала строки и сформировать указатель.

```
Var
  i,j: Integer;
  PtrStr: Array [1..100] of Pointer;
  Pr: ^Extended;
Const
  SizeOfExt = 10;
Begin
  For i:= 1 to 100 do
    GetMem (PtrStr[i], SizeOfExt*200);
. . . . . .
    Pr:= Ptr(Seg(PtrStr[i]^), Ofs(PtrStr[i]^) + (j - 1) * SizeOfExt);
    {Обращение к элементу матрицы [i,j]}
End;
```

далее работаем с Pr^:=. . . и т.д.

Полезно ввести две вспомогательных функции GetExt и PutExt. Каждая из них будет обращаться к функции вычисления адреса AddrE.

```
Program Demo;
Const
  SizeOfExt = 10;
  N = 100;
  M = 200;
Type
  ExtPoint = ^Extended;
Var
  i,j: Integer;
  PtrStr: Array[1..N] of Pointer;
  S: Extended;

  Function AddrE(i,j: Word): ExtPoint;
  Begin
    AddrE:= Ptr(Seg(PtrStr[i]^), Ofs(PtrStr[i]^) + (j - 1) * SizeOfExt);
  End;

  Function GetExt(i,j: Integer): Extended;
  Begin
   GetExt:= AddrE(i,j)^;
  End;

  Procedure PutExt(i,j: Integer; X: Extended);
  Begin
   AddrE(i,j)^:= X;
  End;

  Begin {main}
   Randomize;
   For i:=1 to N do
   Begin
    GetMem (PtrStr[i], M*SizeOfExt);
    For i:= 1 to m do
      PutExt(i, j, Random(255));
   End;
   S:= 0;
  For i:= 1 to N do
   For j:= 1 to M do
    S:= S + GetExt(i,j);
   WriteLn(S/(N*M));
End.
```

Мы предполагали, что каждая строка размещается в куче с начала границы параграфа и смещение каждого указателя PtrStr ровно 0. В действительности при последовательных обращениях к GetMem начало очередного фрагмента следует за концом предыдущего и может не попасть на границу сегмента. В результате при размещении фрагментов максимальной длины (65521 байт) может возникнуть переполнение при вычислении смещения последнего байта.

## 3. Процедуры и функции для работы с динамической памятью

• Функция **Addr** - возвращает результат типа Pointer, в котором содержится адрес аргумента.
• **Addr(X), x** - любой объект программы. Возвращаемый адрес совместим с указателем любого типа. Аналогично операции @.
• Функция **CSeg** - возвращает значения хранящееся в регистре CS (в начале работы программы в CS содержится сегмент начала кода программы), результат CSeg - слово типа Word.
• Процедура **Dispose(x)** - возвращает в кучу фрагмент динамической памяти, зарезервированный за типизированным указателем x.
• Функция **DSeg** - возвращает значение хранящиеся в регистре DS (в начале работы в DS - сегмент начала данных программы), результат - типа Word.
• Процедура **FreeMem** - возвращает в кучу фрагмент динамической памяти, который ранее был зарезервирован за нетипизированным указателем. FreeMem(P, Size), P - нетипизированный указатель. Size - длина фрагмента, подлежащего освобождению.
• Процедура **GetMem(P, Size)** - резервирует память (за одно обращение не более 65521 байт), если нет свободной памяти - ошибка времени исполнения.
• Процедура **Mark(Ptr)** запоминает текущее значение указателя кучи HeapPtr. Ptr - указатель любого типа, в нём будет возвращено HeapPtr. Используется совместно с Release для освобождения части кучи.
• Функция **MaxAvail** - возвращает размер (в байтах) наибольшего непрерывного свободного участка кучи. Результат типа LongInt. За один вызов New или GetMem нельзя зарезервировать значение большее, чем возвращаемое этой функцией.
• Процедура **New(TP)** - резервирует фрагмент кучи для размещения переменной. TP - типизированный указатель (за одно обращение не более 65521байт).
• Функция **MemAvail** - возвращает размер (в байтах) общего свободного пространства кучи. Результат типа Longint.
• Функция **Ofs(X)** - возвращает значение типа Word, содержащее смещение адреса указанного объекта. X - выражение любого типа или имя процедуры.
• Функция **Ptr(Seg, Ofs)** - возвращает значение типа Pointer по заданному сегменту и смещению. Значение, возвращаемое функцией, совместимо с указателем любого типа.
• Процедура **Release(Ptr)** - освобождает участок кучи. Ptr - указатель любого типа, в котором сохранено процедурой Mark значение указателя кучи. Освобожденный участок кучи - от адреса в Ptr до конца. Одновременно уничтожается список свободных фрагментов, созданных по Dispose и FreeMem.
• Функция **Seg(X)** - возвращает значение типа Word, содержащее сегмент адреса указанного объекта.
• Функция **SizeOf(X)** - возвращает длину (в байтах) внутреннего представления указанного объекта. X - имя переменной, функции или типа.

**Проблема потерянных ссылок**

Работа с динамическими переменными через указатели требует большой тщательности и аккуратности при проектировании программ. В частности, следует стремиться освобождать выделенные области сразу же после того, как необходимость в них отпадает, иначе “засорение” памяти ненужными динамическими переменными может привести к быстрому ее исчерпанию.

Кроме того, необходимо учитывать еще одну проблему, связанную с противоречием между стековым принципом размещения статических переменных и произвольным характером создания и уничтожения динамических переменных. Рассмотрим следующий схематический пример программы:

```
Program LostReference;
Type
  PPerson = ^Person;
  Person = Record
. . . .
  End;

Procedure GetPerson;
Var
  P: PPerson;
Begin
  P:= New(PPerson);
End;

Begin
  WriteLn(MemAvail);
  GetPerson;
  Writeln(MemAvail);
End.
```

Вызов New в процедуре GetPerson приводит к отведению памяти для динамической переменной типа Person. Указатель на эту переменную присваивается переменной P. Рассмотрим ситуацию, возникающую после выхода из процедуры GetPerson. По правилам блочности все локальные переменные подпрограммы перестают существовать после ее завершения. В нашем случае исчезает локальная переменная P. Но, с другой стороны, область памяти, отведенная в процессе работы GetPerson, продолжает существовать, так как освободить ее можно только явно, посредством процедуры Dispose. Таким образом, после выхода из GetPerson отсутствует какой бы то ни было доступ к динамической переменной, так как единственная "ниточка", связывавшая ее с программой - указатель P - оказался потерянным при завершении GetPerson. Вывод на печать общего объема свободной памяти до и после работы GetPerson подтверждает потерю определенной области.

При проектировании программ, интенсивно использующих динамическую память, следует с особой внимательностью относиться к данной проблеме, так как Turbo Pascal, как, впрочем, и многие другие языки программирования, не имеет встроенных средств борьбы с засорением памяти неиспользуемыми динамическими переменными. Во всяком случае нужно придерживаться правила, согласно которому при выходе из блока необходимо или освободить все созданные в нем динамические переменные, или сохранить каким-то образом ссылки на них (например, присвоив эти ссылки глобальным переменным).

К описанной проблеме примыкает коллизия другого рода, заключающаяся в ситуации, когда некоторая область памяти освобождена, а в программе остался указатель на эту область. Рассмотрим следующий ***пример:***

```
Var
  P: Integer;

  Procedure X1;
  Var
    i: Integer;
```

```
 Begin
   i:= 12345;
   P:= @i;
   WriteLn(P^); { напечатает 12345 }
 End;

 Procedure X2;
 Var
   j: Integer;
 Begin
   j:= 7777;
   WriteLn(P^); { напечатает 7777, а не 12345 }
 End;

Begin
 X1;
 X2;
End;
```

В этом примере глобальная ссылочная переменная P первоначально (в процедуре X1) устанавливается на локальную переменную i. После завершения процедуры X2 переменная i исчезает, указатель P “повисает”. Вызов процедуры X2 приводит к тому, что на место, локальной переменной i, будет помещена локальная переменная j, и указатель P теперь ссылается на нее, что подтверждает результат второго вызова WriteLn.

Таким образом, смысл указателя P зависит в данном случае не от семантики решаемой задачи, а от системных особенностей ее функционирования, что может привести к неожиданным эффектам. Аналогичные ситуации возможны при повторном использовании областей динамической памяти: если на освобожденную область остался указатель, то он может быть (по ошибке) использован после повторного выделения этой памяти для другой переменной, что опять- таки не будет "замечено" системой, но может сделать поведение программы непредсказуемым.

## Лабораторная работа № 10

## Тема: составление алгоритмов и программ с использованием визуальных компонентов EDIT, LABEL, BUTTON, MEMO, STRINGGRID$.

Все глобальные переменные и типизированные константы размещаются в одной непрерывной области оперативной памяти, которая называется сегментом данных. Длина сегмента определяется архитектурой процессора 8086 и составляет 64 Килобайта (65536 байт), что может вызвать определённые трудности при описании и обработке больших массивов данных. С другой стороны объём стандартной памяти - 640 Килобайт. Выход - использовать динамическую память.

**Динамическая память** - это оперативная память ЭВМ, предоставляемая Турбо-Паскалевой программе при её работе, за вычетом сегмента данных (64 К), стека (обычно 16 К) и собственно тела программы. По умолчанию размер динамической памяти

определяется всей доступной памятью ЭВМ и, как правило, составляет не менее 200 - 300 Кбайт.

Динамическую память обычно используют при:
1. обработке больших массивов данных;
2. разработке САПР;
3. временном запоминании данных при работе с графическими и звуковыми средствами ЭВМ.

Размещение статических переменных в памяти осуществляется компилятором в процессе компиляции.
**Динамические переменные** - размещаются в памяти непосредственно в процессе работы программы. При динамическом размещении заранее неизвестны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным. Турбо-Паскаль представляет средство управления динамической памятью: указатели.

**Указатель** - это переменная, которая в хранит качестве своего значения адрес байта памяти.
В 8086 адреса задаются совокупностью двух шестнадцатиразрядных слов - сегмента и смещения. Сегмент - участок памяти, имеющий максимальную длину 64 К и начинающийся к физического адреса, кратного 16 (то есть 0, 16, 32, 48 и т.д.). Смещение - указывает, сколько байт от начала сегмента нужно пропустить, чтобы обратиться по нужному адресу. Фрагмент памяти в 16 байт называется параграфом. Сегмент адресует память с точностью до параграфа, а смещение - с точностью до байта.

Каждому сегменту соответствует непрерывная и отдельно адресуемая область памяти. Сегменты могут следовать в памяти один за другим, или с некоторыми интервалами, или, наконец, перекрывать друг друга. Таким образом любой указатель по своей внутренней структуре представляет собой совокупность двух слов (типа Word), трактуемых как сегмент и смещение. Указатель адресует лишь первый байт типа данных.

*Справка: процессор 8086.*
8086 имеет 16 - битовые регистры, всего 14 штук, из них:

- 12 - регистры данных и адресов;
- 1 - указатель команд (регистр адреса команды);
- 1 - регистр состояния (регистр флагов).

Регистры данных и адресов делятся на три группы:
- регистр данных;
- регистр указателей и индексов;
- регистр сегментов.

Процессор 8086 всегда генерирует 20-ти бытовые адреса за счёт добавления 16-ти битового смещения к содержимому регистра, умноженному на 16:
*физический адрес = смещение + 16 * регистр сегмента.*
Вместо умножения на 16 содержимое регистра сегмента используется так, как если бы оно имело четыре дополнительных нулевых бита:
**пусть:**

смещение = 10H

регистр сегмента = 2000H
тогда:

| | | | | | |
|---|---|---|---|---|---|
| | 0000 | 0000 | 0001 | 0000 | (смещение) |
| 0010 | 0000 | 0000 | 0000 | (0000) | (номер блока) |
| 0010 | 0000 | 0000 | 0001 | 0000 | (физический адрес) |

физический адрес = 20010H

У 8086 адрес ячейки задаётся номером блока и смещением, что обусловлено тем, что команды 8086 и её данные должны располагаться в разных частях памяти (в разных сегментах). Если требуется адресоваться к данным, то потребуется адресация блока памяти, с которого начинается сегмент данных (из регистра сегмента данных) и позиция желаемой ячейки в этом сегменте (смещение).

В дополнение к области памяти 1 Мбайт, 8086 может адресоваться к внешним устройствам через 65536 (64 К) портов ввода-вывода. Имеются специальные команды ввода-вывода, позволяющие иметь непосредственный доступ к первым 256 (от 0 до 255) портам. Другие команды позволяют получить косвенный доступ к порту с помощью занесения идентифицирующего его номера (0 - 65535) в определенный регистр данных. Подобно ячейке памяти любой порт может быть 8 или 16- битовым.

**Распределение памяти IBM PC.**

• 16 - старших байт - команды начальной загрузки системы.
• Первые 1024 ячейки - вектора прерываний.

**Типы прерываний.**

Существуют 2 вида прерываний - одни можно игнорировать, другие обязательно обслужить как можно скорее.
8086 может распознать 256 различных прерываний, каждому из них однозначно соответствует код типа (целое число от 0 до 255). Код используется в качестве указателя ячейки в области памяти с младшими адресами (область векторов прерываний).

## 1. Объявление указателей

Как правило, в Турбо-Паскале указатель связывается с некоторым типом данных (**типизированные** указатели).

```
Type
  PerPnt = ^PerRec;{указатель на переменную типа PerRec}
  PerRec = Record
    Name: String;
    Job: String;
    Next: PerPnt;
  End;
Var
  P1: ^Integer;
  P2: ^Real;
```

***Внимание!*** При объявлении типа PerPnt мы сослались на PerRec, который ещё не был объявлен, это связано с тем, что существует исключение из правил для указателей, которые содержат ссылку на ещё неописанный тип данных. Это исключение сделано не случайно, так как динамическая память позволяет использовать организацию данных в

виде списков (каждый элемент имеет в своём составе ссылку на соседний элемент - обратите внимание на поле Next).

В Турбо-Паскале можно объявлять указатель и не связывать его с конкретным типом данных:

```
Var
  PP: Pointer;
```

Указатели такого рода называют **нетипизированными**. В них удобно размещать данные, структура которых меняется по ходу работы.

В Турбо-Паскале можно передавать значения только между указателями, связанными с одним и тем же типом данных.

```
Var
  P1,P2: ^Integer;
  P3: ^Real;
  PP:Pointer;
Begin
  P1:= P2; {- верно}
  P1:= P3; {- неверно}
```

но можно сделать так:

```
  PP:= P3;
  P1:= PP;
```

так как ограничение не распространяются на нетипизированные указатели. Но подобные операции часто путают программиста и чреваты смысловыми ошибками.

## 2. Выделение и освобождение динамической памяти

Вся динамическая память – пространство ячеек, называемое кучей. Физически куча располагается в старших адресах, сразу за программой. Указатель на начало кучи хранится в предопределенной переменной HeapOrg, конец - FreePtr, текущую границу незанятой динамической памяти указывает указатель HeapPtr. Для выделения памяти под любую переменную используется процедура New. Единственным параметром является типизированный указатель:

```
Var
  I,J: ^Integer;
  R: ^Real;
Begin
  New(I); {под I выделяется область памяти,}
  {адрес первого байта этой области помещается в I}
End.
```

После выполнения этого фрагмента указатель I приобретёт значение, которое перед этим имел указатель кучи HeapPtr, а HeapPtr увеличится на два (т.к. он типа Integer); New(R) - вызовет смещение указателя на 6 байт. После того как указатель приобрёл некоторое значение, то есть стал указывать на конкретный байт памяти, по этому адресу можно разместить значения соответствующего типа:

```
  I^:= 2;
  R^:= 2*Pi;
```
Допустима запись: R^:= Sqr (R^) + I^ - 17;
Но недопустима запись: R:= Sqr (R^) + I^ - 17; так как указателю R нельзя присвоить значение вещественного выражения.

Возврат динамической памяти обратно в кучу осуществляется оператором Dispose:
```
  Dispose(R);
```
  Dispose(I); - вернут в кучу, ранее забранные 8 байт.
  Dispose(Ptr) не изменяет значения указателя Ptr, а лишь возвращает в кучу память, связанную с этим указателем. Однако повторное применение процедуры к “свободному” указателю приведет к возникновению ошибки времени исполнения. Чтобы указать, что указатель свободен, нужно использовать зарезервированное слово Nil.

К указателям можно применять операции отношения, в том числе и сравнения с Nil:

```
Const
 P:^Real = Nil;
. . . . . .
Begin
 If P = Nil then
   New (P);
. . . . . .
 Dispose(P);
 P:= Nil;
End.
```

Использование указателей, которым не присвоено значение процедурой New или каким-либо другим способом, никак не контролируется системой и может привести к непредсказуемым результатам. Многократное чередование New и Dispose приводит к “ячеистой” структуре кучи. Дело в том, что все операции с кучей выполняется под управлением особой программы, которая ведёт учёт всех свободных фрагментов в куче. При выполнении оператора New( ) эта программа отыскивает минимальный свободный фрагмент, в котором может разместиться требуемая переменная. Адрес начала найденного фрагмента возвращается в указателе, а сам фрагмент или его часть нужной длины, помечаются как занятая часть кучи.

Можно освободить целый фрагмент кучи следующим образом:

1. Перед началом выделения динамической памяти значения указателя HeapPtr запоминается в переменной-указателе с помощью процедуры Mark.
2. Выполнение программы.
3. Освобождение фрагмента кучи от заполненного адреса до конца динамической памяти с использованием процедуры Release.

```
Var
 P, P1, P2, P3, P4, P5: ^Integer;
Begin
 New(P1);
 New(P2);
 New(P3);
 New(P4);
```

```
 New(P5);
 Mark(P);
. . . . . .
 Release (P);
End.
```

В этом примере процедурой Mark(P) в указатель P было помещено текущее значение HeapPtr, однако память под переменную не резервировалась.

Вызов Release уничтожает список свободных фрагментов в куче, созданных Dispose, поэтому совместное применение этих процедур не рекомендуется.

Для работы с нетипизированными указателями используются также процедуры GetMem(P, Size) и FreeMem(P, Size) - резервирование и освобождение памяти.
P - нетипизированный указатель, Size - размер.

За одно обращение к куче процедурой GetMem можно зарезервировать до 65521 байт. Освобождать нужно ровно столько памяти, сколько было зарезервировано, и именно с того адреса, с которого память была зарезервирована, иначе программа не будет работать и завершаться корректно!!!

Использование нетипизированных указателей даёт широкие возможности неявного преобразования типов:

```
Var
  i,j: ^Integer;
  r: ^Real;
Begin
  New (i); { I:= HeapOrg; HeapPtr:= HeapOrg+2 }
  j:= i; { J:= HeapOrg }
  j^:=2;
  Dispose (i); { HeapPtr:= HeapOrg }
  New (r); { R:= HeapOrg; HeapPtr:= HeapOrg+6 }
  r^:= Pi;
  WriteLn (j^);
End.
{Будет выведено: "8578"}
{здесь преобразование не имеет никакого смысла}
```

**Примеры использования указателей.**

Динамическая память - 200..300 Кбайт. Нужно разместить массив 100 * 200 типа Extended. Требуется 100 * 200 * 10 = 200000 байт. Пробуем:

```
Var
  i,j: Integer;
  PtrArr: Array[1..100, 1..200] of ^Extended.
Begin
. . . . . .
  For i:= 1 to 100 do
    For j:= 1 to 200 do
     New (PtrArr [i,j]);
. . . . . .
End.
```

Теперь к любому элементу можно обратиться: PtrArr[i,j]^:=...; Но длина внутреннего представления указателей 4 байта, поэтому потребуется ещё 100*200*4 = 80000 байт, что превышает размер сегмента (65536 байт), доступный для статического размещения данных.

Можно было бы работать с адресной арифметикой (арифметикой над указателями), то есть не создавать массив указателей, а вычислять адрес элемента непосредственно перед обращением к нему. Однако в Турбо-Паскале над указателями не определены никакие операции, кроме присваивания и отношения. Но задачу решить можно:
Seg(x) - возвращает сегментную часть адреса.
Ofs(x) - возвращает смещение.
x - любая переменная, в том числе и та, на которую указывает указатель.

Далее с помощью Ptr(Seg, Ofs: Word): Pointer можно создать значение указателя, совместимое с указателем любого типа.

Таким образом, сначала с помощью GetMem забираем из кучи несколько фрагментов подходящей длины (не более 65521). Удобно по строкам 200 * 100 = 20000 байт. Начало каждого фрагмента запоминается в массиве PtrStr из 100 указателей. Теперь для доступа к любому элементу строки нужно вычислить смещение этого элемента от начала строки и сформировать указатель.

```
Var
  i,j: Integer;
  PtrStr: Array [1..100] of Pointer;
  Pr: ^Extended;
Const
  SizeOfExt = 10;
Begin
  For i:= 1 to 100 do
    GetMem (PtrStr[i], SizeOfExt*200);
. . . . . .
    Pr:= Ptr(Seg(PtrStr[i]^), Ofs(PtrStr[i]^) + (j - 1) * SizeOfExt);
    {Обращение к элементу матрицы [i,j]}
End;
```

далее работаем с Pr^:=. . . и т.д.
Полезно ввести две вспомогательных функции GetExt и PutExt. Каждая из них будет обращаться к функции вычисления адреса AddrE.

```
Program Demo;
Const
  SizeOfExt = 10;
  N = 100;
  M = 200;
Type
  ExtPoint = ^Extended;
Var
  i,j: Integer;
  PtrStr: Array[1..N] of Pointer;
  S: Extended;

  Function AddrE(i,j: Word): ExtPoint;
  Begin
```

```
   AddrE:= Ptr(Seg(PtrStr[i]^), Ofs(PtrStr[i]^) + (j - 1) * SizeOfExt);
 End;

 Function GetExt(i,j: Integer): Extended;
 Begin
  GetExt:= AddrE(i,j)^;
 End;

 Procedure PutExt(i,j: Integer; X: Extended);
 Begin
  AddrE(i,j)^:= X;
 End;

 Begin {main}
  Randomize;
  For i:=1 to N do
  Begin
   GetMem (PtrStr[i], M*SizeOfExt);
   For i:= 1 to m do
     PutExt(i, j, Random(255));
  End;
  S:= 0;
 For i:= 1 to N do
  For j:= 1 to M do
   S:= S + GetExt(i,j);
  WriteLn(S/(N*M));
End.
```

Мы предполагали, что каждая строка размещается в куче с начала границы параграфа и смещение каждого указателя PtrStr ровно 0. В действительности при последовательных обращениях к GetMem начало очередного фрагмента следует за концом предыдущего и может не попасть на границу сегмента. В результате при размещении фрагментов максимальной длины (65521 байт) может возникнуть переполнение при вычислении смещения последнего байта.

## 3. Процедуры и функции для работы с динамической памятью

• Функция **Addr** - возвращает результат типа Pointer, в котором содержится адрес аргумента.
• **Addr(X), x** - любой объект программы. Возвращаемый адрес совместим с указателем любого типа. Аналогично операции @.
• Функция **CSeg** - возвращает значения хранящееся в регистре CS (в начале работы программы в CS содержится сегмент начала кода программы), результат CSeg - слово типа Word.
• Процедура **Dispose(x)** - возвращает в кучу фрагмент динамической памяти, зарезервированный за типизированным указателем x.
• Функция **DSeg** - возвращает значение хранящиеся в регистре DS (в начале работы в DS - сегмент начала данных программы), результат - типа Word.
• Процедура **FreeMem** - возвращает в кучу фрагмент динамической памяти, который ранее был зарезервирован за нетипизированным указателем. FreeMem(P, Size), P -

нетипизированный указатель. Size - длина фрагмента, подлежащего освобождению.
• Процедура **GetMem(P, Size)** - резервирует память (за одно обращение не более 65521 байт), если нет свободной памяти - ошибка времени исполнения.
• Процедура **Mark(Ptr)** запоминает текущее значение указателя кучи HeapPtr. Ptr - указатель любого типа, в нём будет возвращено HeapPtr. Используется совместно с Release для освобождения части кучи.
• Функция **MaxAvail** - возвращает размер (в байтах) наибольшего непрерывного свободного участка кучи. Результат типа LongInt. За один вызов New или GetMem нельзя зарезервировать значение большее, чем возвращаемое этой функцией.
• Процедура **New(TP)** - резервирует фрагмент кучи для размещения переменной. TP - типизированный указатель (за одно обращение не более 65521байт).
• Функция **MemAvail** - возвращает размер (в байтах) общего свободного пространства кучи. Результат типа Longint.
• Функция **Ofs(X)** - возвращает значение типа Word, содержащее смещение адреса указанного объекта. X - выражение любого типа или имя процедуры.
• Функция **Ptr(Seg, Ofs)** - возвращает значение типа Pointer по заданному сегменту и смещению. Значение, возвращаемое функцией, совместимо с указателем любого типа.
• Процедура **Release(Ptr)** - освобождает участок кучи. Ptr - указатель любого типа, в котором сохранено процедурой Mark значение указателя кучи. Освобожденный участок кучи - от адреса в Ptr до конца. Одновременно уничтожается список свободных фрагментов, созданных по Dispose и FreeMem.
• Функция **Seg(X)** - возвращает значение типа Word, содержащее сегмент адреса указанного объекта.
• Функция **SizeOf(X)** - возвращает длину (в байтах) внутреннего представления указанного объекта. X - имя переменной, функции или типа.

**Проблема потерянных ссылок**

Работа с динамическими переменными через указатели требует большой тщательности и аккуратности при проектировании программ. В частности, следует стремиться освобождать выделенные области сразу же после того, как необходимость в них отпадает, иначе “засорение” памяти ненужными динамическими переменными может привести к быстрому ее исчерпанию.

Кроме того, необходимо учитывать еще одну проблему, связанную с противоречием между стековым принципом размещения статических переменных и произвольным характером создания и уничтожения динамических переменных. Рассмотрим следующий схематический пример программы:

```
Program LostReference;
Type
  PPerson = ^Person;
  Person = Record
. . . .
  End;

Procedure GetPerson;
Var
  P: PPerson;
Begin
  P:= New(PPerson);
End;
```

```
Begin
  WriteLn(MemAvail);
  GetPerson;
  Writeln(MemAvail);
End.
```

Вызов New в процедуре GetPerson приводит к отведению памяти для динамической переменной типа Person. Указатель на эту переменную присваивается переменной P. Рассмотрим ситуацию, возникающую после выхода из процедуры GetPerson. По правилам блочности все локальные переменные подпрограммы перестают существовать после ее завершения. В нашем случае исчезает локальная переменная P. Но, с другой стороны, область памяти, отведенная в процессе работы GetPerson, продолжает существовать, так как освободить ее можно только явно, посредством процедуры Dispose. Таким образом, после выхода из GetPerson отсутствует какой бы то ни было доступ к динамической переменной, так как единственная "ниточка", связывавшая ее с программой - указатель P - оказался потерянным при завершении GetPerson. Вывод на печать общего объема свободной памяти до и после работы GetPerson подтверждает потерю определенной области.

При проектировании программ, интенсивно использующих динамическую память, следует с особой внимательностью относиться к данной проблеме, так как Turbo Pascal, как, впрочем, и многие другие языки программирования, не имеет встроенных средств борьбы с засорением памяти неиспользуемыми динамическими переменными. Во всяком случае нужно придерживаться правила, согласно которому при выходе из блока необходимо или освободить все созданные в нем динамические переменные, или сохранить каким-то образом ссылки на них (например, присвоив эти ссылки глобальным переменным).

К описанной проблеме примыкает коллизия другого рода, заключающаяся в ситуации, когда некоторая область памяти освобождена, а в программе остался указатель на эту область. Рассмотрим следующий ***пример:***

```
Var
  P: Integer;

  Procedure X1;
  Var
    i: Integer;
  Begin
    i:= 12345;
    P:= @i;
    WriteLn(P^); { напечатает 12345 }
  End;

  Procedure X2;
  Var
    j: Integer;
  Begin
    j:= 7777;
    WriteLn(P^); { напечатает 7777, а не 12345 }
  End;
```

```
Begin
  X1;
  X2;
End;
```

В этом примере глобальная ссылочная переменная P первоначально (в процедуре X1) устанавливается на локальную переменную i. После завершения процедуры X2 переменная i исчезает, указатель P “повисает”. Вызов процедуры X2 приводит к тому, что на место, локальной переменной i, будет помещена локальная переменная j, и указатель P теперь ссылается на нее, что подтверждает результат второго вызова WriteLn.

Таким образом, смысл указателя P зависит в данном случае не от семантики решаемой задачи, а от системных особенностей ее функционирования, что может привести к неожиданным эффектам. Аналогичные ситуации возможны при повторном использовании областей динамической памяти: если на освобожденную область остался указатель, то он может быть (по ошибке) использован после повторного выделения этой памяти для другой переменной, что опять- таки не будет "замечено" системой, но может сделать поведение программы непредсказуемым.

## Лабораторная работа № 11
## Тема: составление программ с использованием нескольких форм.

Все глобальные переменные и типизированные константы размещаются в одной непрерывной области оперативной памяти, которая называется сегментом данных. Длина сегмента определяется архитектурой процессора 8086 и составляет 64 Килобайта (65536 байт), что может вызвать определённые трудности при описании и обработке больших массивов данных. С другой стороны объём стандартной памяти - 640 Килобайт. Выход - использовать динамическую память.

**Динамическая память** - это оперативная память ЭВМ, предоставляемая Турбо-Паскалевой программе при её работе, за вычетом сегмента данных (64 К), стека (обычно 16 К) и собственно тела программы. По умолчанию размер динамической памяти определяется всей доступной памятью ЭВМ и, как правило, составляет не менее 200 - 300 Кбайт.

Динамическую память обычно используют при:
1. обработке больших массивов данных;
2. разработке САПР;
3. временном запоминании данных при работе с графическими и звуковыми средствами ЭВМ.

Размещение статических переменных в памяти осуществляется компилятором в процессе компиляции.
**Динамические переменные** - размещаются в памяти непосредственно в процессе работы программы. При динамическом размещении заранее неизвестны ни тип, ни количество размещаемых данных, к ним нельзя обращаться по именам, как к статическим переменным. Турбо-Паскаль представляет средство управления динамической памятью: указатели.

**Указатель** - это переменная, которая в хранит качестве своего значения адрес байта

памяти.
В 8086 адреса задаются совокупностью двух шестнадцатиразрядных слов - сегмента и смещения. Сегмент - участок памяти, имеющий максимальную длину 64 К и начинающийся к физического адреса, кратного 16 (то есть 0, 16, 32, 48 и т.д.). Смещение - указывает, сколько байт от начала сегмента нужно пропустить, чтобы обратиться по нужному адресу. Фрагмент памяти в 16 байт называется параграфом. Сегмент адресует память с точностью до параграфа, а смещение - с точностью до байта.

Каждому сегменту соответствует непрерывная и отдельно адресуемая область памяти. Сегменты могут следовать в памяти один за другим, или с некоторыми интервалами, или, наконец, перекрывать друг друга. Таким образом любой указатель по своей внутренней структуре представляет собой совокупность двух слов (типа Word), трактуемых как сегмент и смещение. Указатель адресует лишь первый байт типа данных.

*Справка: процессор 8086.*
8086 имеет 16 - битовые регистры, всего 14 штук, из них:

- 12 - регистры данных и адресов;
- 1 - указатель команд (регистр адреса команды);
- 1 - регистр состояния (регистр флагов).

Регистры данных и адресов делятся на три группы:
- регистр данных;
- регистр указателей и индексов;
- регистр сегментов.

Процессор 8086 всегда генерирует 20-ти бытовые адреса за счёт добавления 16-ти битового смещения к содержимому регистра, умноженному на 16:
*физический адрес = смещение + 16 * регистр сегмента.*
Вместо умножения на 16 содержимое регистра сегмента используется так, как если бы оно имело четыре дополнительных нулевых бита:
**пусть:**

смещение = 10H
регистр сегмента = 2000H

тогда:

| | | | | | |
|---|---|---|---|---|---|
| | 0000 | 0000 | 0001 | 0000 | (смещение) |
| 0010 | 0000 | 0000 | 0000 | (0000) | (номер блока) |
| 0010 | 0000 | 0000 | 0001 | 0000 | (физический адрес) |

физический адрес = 20010H

У 8086 адрес ячейки задаётся номером блока и смещением, что обусловлено тем, что команды 8086 и её данные должны располагаться в разных частях памяти (в разных сегментах). Если требуется адресоваться к данным, то потребуется адресация блока памяти, с которого начинается сегмент данных (из регистра сегмента данных) и позиция желаемой ячейки в этом сегменте (смещение).

В дополнение к области памяти 1 Мбайт, 8086 может адресоваться к внешним устройствам через 65536 (64 К) портов ввода-вывода. Имеются специальные команды ввода-вывода, позволяющие иметь непосредственный доступ к первым 256 (от 0 до 255) портам. Другие команды позволяют получить косвенный доступ к порту с помощью занесения идентифицирующего его номера (0 - 65535) в определенный регистр данных. Подобно ячейке памяти любой порт может быть 8 или 16- битовым.

**Распределение памяти IBM PC.**

• 16 - старших байт - команды начальной загрузки системы.
• Первые 1024 ячейки - вектора прерываний.

**Типы прерываний.**

Существуют 2 вида прерываний - одни можно игнорировать, другие обязательно обслужить как можно скорее.
8086 может распознать 256 различных прерываний, каждому из них однозначно соответствует код типа (целое число от 0 до 255). Код используется в качестве указателя ячейки в области памяти с младшими адресами (область векторов прерываний).

## 1. Объявление указателей

Как правило, в Турбо-Паскале указатель связывается с некоторым типом данных (**типизированные** указатели).

```
Type
  PerPnt = ^PerRec;{указатель на переменную типа PerRec}
  PerRec = Record
    Name: String;
    Job: String;
    Next: PerPnt;
  End;
Var
  P1: ^Integer;
  P2: ^Real;
```

***Внимание!*** При объявлении типа PerPnt мы сослались на PerRec, который ещё не был объявлен, это связано с тем, что существует исключение из правил для указателей, которые содержат ссылку на ещё неописанный тип данных. Это исключение сделано не случайно, так как динамическая память позволяет использовать организацию данных в виде списков (каждый элемент имеет в своём составе ссылку на соседний элемент - обратите внимание на поле Next).

В Турбо-Паскале можно объявлять указатель и не связывать его с конкретным типом данных:

```
Var
  PP: Pointer;
```

Указатели такого рода называют **нетипизированными**. В них удобно размещать данные, структура которых меняется по ходу работы.

В Турбо-Паскале можно передавать значения только между указателями, связанными с одним и тем же типом данных.

```
Var
  P1,P2: ^Integer;
  P3: ^Real;
  PP:Pointer;
Begin
```

```
P1:= P2; {- верно}
P1:= P3; {- неверно}
```

но можно сделать так:

```
PP:= P3;
P1:= PP;
```

так как ограничение не распространяются на нетипизированные указатели. Но подобные операции часто путают программиста и чреваты смысловыми ошибками.

## 2. Выделение и освобождение динамической памяти

Вся динамическая память – пространство ячеек, называемое кучей. Физически куча располагается в старших адресах, сразу за программой. Указатель на начало кучи хранится в предопределенной переменной HeapOrg, конец - FreePtr, текущую границу незанятой динамической памяти указывает указатель HeapPtr. Для выделения памяти под любую переменную используется процедура New. Единственным параметром является типизированный указатель:

```
Var
  I,J: ^Integer;
  R: ^Real;
Begin
  New(I); {под I выделяется область памяти,}
  {адрес первого байта этой области помещается в I}
End.
```

После выполнения этого фрагмента указатель I приобретёт значение, которое перед этим имел указатель кучи HeapPtr, а HeapPtr увеличится на два (т.к. он типа Integer); New(R) - вызовет смещение указателя на 6 байт. После того как указатель приобрёл некоторое значение, то есть стал указывать на конкретный байт памяти, по этому адресу можно разместить значения соответствующего типа:

```
  I^:= 2;
  R^:= 2*Pi;
```

Допустима запись: R^:= Sqr (R^) + I^ - 17;
Но недопустима запись: R:= Sqr (R^) + I^ - 17; так как указателю R нельзя присвоить значение вещественного выражения.

Возврат динамической памяти обратно в кучу осуществляется оператором Dispose:

```
  Dispose(R);
  Dispose(I); - вернут в кучу, ранее забранные 8 байт.
```

Dispose(Ptr) не изменяет значения указателя Ptr, а лишь возвращает в кучу память, связанную с этим указателем. Однако повторное применение процедуры к “свободному” указателю приведет к возникновению ошибки времени исполнения. Чтобы указать, что указатель свободен, нужно использовать зарезервированное слово Nil.

К указателям можно применять операции отношения, в том числе и сравнения с Nil:

```
Const
 P:^Real = Nil;
```

```
. . . . . .
Begin
 If P = Nil then
   New (P);
. . . . . .
 Dispose(P);
 P:= Nil;
End.
```

Использование указателей, которым не присвоено значение процедурой New или каким-либо другим способом, никак не контролируется системой и может привести к непредсказуемым результатам. Многократное чередование New и Dispose приводит к “ячеистой” структуре кучи. Дело в том, что все операции с кучей выполняется под управлением особой программы, которая ведёт учёт всех свободных фрагментов в куче. При выполнении оператора New( ) эта программа отыскивает минимальный свободный фрагмент, в котором может разместиться требуемая переменная. Адрес начала найденного фрагмента возвращается в указателе, а сам фрагмент или его часть нужной длины, помечаются как занятая часть кучи.

Можно освободить целый фрагмент кучи следующим образом:

1. Перед началом выделения динамической памяти значения указателя HeapPtr запоминается в переменной-указателе с помощью процедуры Mark.
2. Выполнение программы.
3. Освобождение фрагмента кучи от заполненного адреса до конца динамической памяти с использованием процедуры Release.

```
Var
 P, P1, P2, P3, P4, P5: ^Integer;
Begin
 New(P1);
 New(P2);
 New(P3);
 New(P4);
 New(P5);
 Mark(P);
. . . . . .
 Release (P);
End.
```

В этом примере процедурой Mark(P) в указатель P было помещено текущее значение HeapPtr, однако память под переменную не резервировалась.

Вызов Release уничтожает список свободных фрагментов в куче, созданных Dispose, поэтому совместное применение этих процедур не рекомендуется.

Для работы с нетипизированными указателями используются также процедуры GetMem(P, Size) и FreeMem(P, Size) - резервирование и освобождение памяти.
P - нетипизированный указатель, Size - размер.

За одно обращение к куче процедурой GetMem можно зарезервировать до 65521 байт. Освобождать нужно ровно столько памяти, сколько было зарезервировано, и именно с

того адреса, с которого память была зарезервирована, иначе программа не будет работать и завершаться корректно!!!

Использование нетипизированных указателей даёт широкие возможности неявного преобразования типов:

```
Var
  i,j: ^Integer;
  r: ^Real;
Begin
  New (i); { I:= HeapOrg; HeapPtr:= HeapOrg+2 }
  j:= i; { J:= HeapOrg }
  j^:=2;
  Dispose (i); { HeapPtr:= HeapOrg }
  New (r); { R:= HeapOrg; HeapPtr:= HeapOrg+6 }
  r^:= Pi;
  WriteLn (j^);
End.
{Будет выведено: "8578"}
{здесь преобразование не имеет никакого смысла}
```

**Примеры использования указателей.**

Динамическая память - 200..300 Кбайт. Нужно разместить массив 100 * 200 типа Extended. Требуется 100 * 200 * 10 = 200000 байт. Пробуем:

```
Var
  i,j: Integer;
  PtrArr: Array[1..100, 1..200] of ^Extended.
Begin
. . . . . .
  For i:= 1 to 100 do
    For j:= 1 to 200 do
     New (PtrArr [i,j]);
. . . . . .
End.
```

Теперь к любому элементу можно обратиться: PtrArr[i,j]^:=...; Но длина внутреннего представления указателей 4 байта, поэтому потребуется ещё 100*200*4 = 80000 байт, что превышает размер сегмента (65536 байт), доступный для статического размещения данных.

Можно было бы работать с адресной арифметикой (арифметикой над указателями), то есть не создавать массив указателей, а вычислять адрес элемента непосредственно перед обращением к нему. Однако в Турбо-Паскале над указателями не определены никакие операции, кроме присваивания и отношения. Но задачу решить можно:

Seg(x) - возвращает сегментную часть адреса.

Ofs(x) - возвращает смещение.

x - любая переменная, в том числе и та, на которую указывает указатель.

Далее с помощью Ptr(Seg, Ofs: Word): Pointer можно создать значение указателя, совместимое с указателем любого типа.

Таким образом, сначала с помощью GetMem забираем из кучи несколько фрагментов подходящей длины (не более 65521). Удобно по строкам 200 * 100 = 20000 байт. Начало

каждого фрагмента запоминается в массиве PtrStr из 100 указателей. Теперь для доступа к любому элементу строки нужно вычислить смещение этого элемента от начала строки и сформировать указатель.

```
Var
  i,j: Integer;
  PtrStr: Array [1..100] of Pointer;
  Pr: ^Extended;
Const
  SizeOfExt = 10;
Begin
  For i:= 1 to 100 do
    GetMem (PtrStr[i], SizeOfExt*200);
. . . . . .
    Pr:= Ptr(Seg(PtrStr[i]^), Ofs(PtrStr[i]^) + (j - 1) * SizeOfExt);
    {Обращение к элементу матрицы [i,j]}
End;
```

далее работаем с Pr^:=. . . и т.д.

Полезно ввести две вспомогательных функции GetExt и PutExt. Каждая из них будет обращаться к функции вычисления адреса AddrE.

```
Program Demo;
Const
  SizeOfExt = 10;
  N = 100;
  M = 200;
Type
  ExtPoint = ^Extended;
Var
  i,j: Integer;
  PtrStr: Array[1..N] of Pointer;
  S: Extended;

  Function AddrE(i,j: Word): ExtPoint;
  Begin
    AddrE:= Ptr(Seg(PtrStr[i]^), Ofs(PtrStr[i]^) + (j - 1) * SizeOfExt);
  End;

  Function GetExt(i,j: Integer): Extended;
  Begin
   GetExt:= AddrE(i,j)^;
  End;

  Procedure PutExt(i,j: Integer; X: Extended);
  Begin
   AddrE(i,j)^:= X;
  End;

  Begin {main}
   Randomize;
   For i:=1 to N do
   Begin
    GetMem (PtrStr[i], M*SizeOfExt);
```

```
    For i:= 1 to m do
      PutExt(i, j, Random(255));
   End;
   S:= 0;
  For i:= 1 to N do
   For j:= 1 to M do
    S:= S + GetExt(i,j);
   WriteLn(S/(N*M));
End.
```

Мы предполагали, что каждая строка размещается в куче с начала границы параграфа и смещение каждого указателя PtrStr ровно 0. В действительности при последовательных обращениях к GetMem начало очередного фрагмента следует за концом предыдущего и может не попасть на границу сегмента. В результате при размещении фрагментов максимальной длины (65521 байт) может возникнуть переполнение при вычислении смещения последнего байта.

### 3. Процедуры и функции для работы с динамической памятью

• Функция **Addr** - возвращает результат типа Pointer, в котором содержится адрес аргумента.
• **Addr(X), x** - любой объект программы. Возвращаемый адрес совместим с указателем любого типа. Аналогично операции @.
• Функция **CSeg** - возвращает значения хранящееся в регистре CS (в начале работы программы в CS содержится сегмент начала кода программы), результат CSeg - слово типа Word.
• Процедура **Dispose(x)** - возвращает в кучу фрагмент динамической памяти, зарезервированный за типизированным указателем x.
• Функция **DSeg** - возвращает значение хранящиеся в регистре DS (в начале работы в DS - сегмент начала данных программы), результат - типа Word.
• Процедура **FreeMem** - возвращает в кучу фрагмент динамической памяти, который ранее был зарезервирован за нетипизированным указателем. FreeMem(P, Size), P - нетипизированный указатель. Size - длина фрагмента, подлежащего освобождению.
• Процедура **GetMem(P, Size)** - резервирует память (за одно обращение не более 65521 байт), если нет свободной памяти - ошибка времени исполнения.
• Процедура **Mark(Ptr)** запоминает текущее значение указателя кучи HeapPtr. Ptr - указатель любого типа, в нём будет возвращено HeapPtr. Используется совместно с Release для освобождения части кучи.
• Функция **MaxAvail** - возвращает размер (в байтах) наибольшего непрерывного свободного участка кучи. Результат типа LongInt. За один вызов New или GetMem нельзя зарезервировать значение большее, чем возвращаемое этой функцией.
• Процедура **New(TP)** - резервирует фрагмент кучи для размещения переменной. TP - типизированный указатель (за одно обращение не более 65521байт).
• Функция **MemAvail** - возвращает размер (в байтах) общего свободного пространства кучи. Результат типа Longint.
• Функция **Ofs(X)** - возвращает значение типа Word, содержащее смещение адреса указанного объекта. X - выражение любого типа или имя процедуры.
• Функция **Ptr(Seg, Ofs)** - возвращает значение типа Pointer по заданному сегменту и смещению. Значение, возвращаемое функцией, совместимо с указателем любого типа.
• Процедура **Release(Ptr)** - освобождает участок кучи. Ptr - указатель любого типа, в

котором сохранено процедурой Mark значение указателя кучи. Освобожденный участок кучи - от адреса в Ptr до конца. Одновременно уничтожается список свободных фрагментов, созданных по Dispose и FreeMem.
• Функция **Seg(X)** - возвращает значение типа Word, содержащее сегмент адреса указанного объекта.
• Функция **SizeOf(X)** - возвращает длину (в байтах) внутреннего представления указанного объекта. X - имя переменной, функции или типа.

**Проблема потерянных ссылок**

Работа с динамическими переменными через указатели требует большой тщательности и аккуратности при проектировании программ. В частности, следует стремиться освобождать выделенные области сразу же после того, как необходимость в них отпадает, иначе “засорение” памяти ненужными динамическими переменными может привести к быстрому ее исчерпанию.

Кроме того, необходимо учитывать еще одну проблему, связанную с противоречием между стековым принципом размещения статических переменных и произвольным характером создания и уничтожения динамических переменных. Рассмотрим следующий схематический пример программы:

```
Program LostReference;
Type
  PPerson = ^Person;
  Person = Record
. . . .
  End;

Procedure GetPerson;
Var
  P: PPerson;
Begin
  P:= New(PPerson);
End;

Begin
  WriteLn(MemAvail);
  GetPerson;
  Writeln(MemAvail);
End.
```

Вызов New в процедуре GetPerson приводит к отведению памяти для динамической переменной типа Person. Указатель на эту переменную присваивается переменной P. Рассмотрим ситуацию, возникающую после выхода из процедуры GetPerson. По правилам блочности все локальные переменные подпрограммы перестают существовать после ее завершения. В нашем случае исчезает локальная переменная P. Но, с другой стороны, область памяти, отведенная в процессе работы GetPerson, продолжает существовать, так как освободить ее можно только явно, посредством процедуры Dispose. Таким образом, после выхода из GetPerson отсутствует какой бы то ни было доступ к динамической переменной, так как единственная "ниточка", связывавшая ее с программой - указатель P - оказался потерянным при завершении GetPerson. Вывод на печать общего объема свободной памяти до и после работы GetPerson подтверждает потерю определенной области.

При проектировании программ, интенсивно использующих динамическую память, следует с особой внимательностью относиться к данной проблеме, так как Turbo Pascal, как, впрочем, и многие другие языки программирования, не имеет встроенных средств борьбы с засорением памяти неиспользуемыми динамическими переменными. Во всяком случае нужно придерживаться правила, согласно которому при выходе из блока необходимо или освободить все созданные в нем динамические переменные, или сохранить каким-то образом ссылки на них (например, присвоив эти ссылки глобальным переменным).

К описанной проблеме примыкает коллизия другого рода, заключающаяся в ситуации, когда некоторая область памяти освобождена, а в программе остался указатель на эту область. Рассмотрим следующий ***пример:***

```
Var
  P: Integer;

  Procedure X1;
  Var
    i: Integer;
  Begin
    i:= 12345;
    P:= @i;
    WriteLn(P^); { напечатает 12345 }
  End;

  Procedure X2;
  Var
    j: Integer;
  Begin
    j:= 7777;
    WriteLn(P^); { напечатает 7777, а не 12345 }
  End;

Begin
  X1;
  X2;
End;
```

В этом примере глобальная ссылочная переменная P первоначально (в процедуре X1) устанавливается на локальную переменную i. После завершения процедуры X2 переменная i исчезает, указатель P “повисает”. Вызов процедуры X2 приводит к тому, что на место, локальной переменной i, будет помещена локальная переменная j, и указатель P теперь ссылается на нее, что подтверждает результат второго вызова WriteLn.

Таким образом, смысл указателя P зависит в данном случае не от семантики решаемой задачи, а от системных особенностей ее функционирования, что может привести к неожиданным эффектам. Аналогичные ситуации возможны при повторном использовании областей динамической памяти: если на освобожденную область остался указатель, то он может быть (по ошибке) использован после повторного выделения этой памяти для другой переменной, что опять- таки не будет "замечено" системой, но может сделать поведение программы непредсказуемым.

# Лабораторная работа №14,15

**Составление программ с использованием MAINMENU, POPURMENU.**

Цель работы:

1.Изучить характеристики компонентов MainMenu, PopupMenu инаучиться их использовать при разработке программ.

2.Изучить характеристики стандартных диалогов: для открытия и сохранения файлов OpenDialog, OpenPictureDialog, SaveDialog, SavePictureDialog;. для выбора шрифтов FontDialog; для выбора цвета ColorDialog, для печати и установки принтера PrintDialog, PrinterSetupDialog инаучиться их использовать при разработке программ.

3.Изучить характеристики информационного окна программы «О программе» AboutBox, содержащего краткую справочную информацию о названии программного продукта, его версии, дате выпуска, авторах, фотографий авторов, контактный телефон,e-mailи другие данные инаучиться его использовать при разработке приложений.

4.Выполнить практическую работу по разработке приложений.

Выполнить самостоятельную работупо разработке
приложений.

5.Защитить лабораторную работу, продемонстрировать

программы всех практических заданий, ответить на дополнительные вопросы по данной теме.

Порядок работы:

Большинство сложных программ в настоящее время обладают системой меню, которая предназначена для выбора различных путей выполнения приложения.

Меню представляет собой набор команд, при выборе каждой из которых выполняются определенные программистом действия. В Delphi есть два компонента, используемые для создания меню:MainMenu - главное меню иРорирМепи - вспомогательное меню.

MainMenu. Для пользователя размещение на форме этого элемента не играет никакой роли, потому что он является невизуальным компонентом. При выполнении программы будет видно только меню, которое сгенерируетMainMenu.

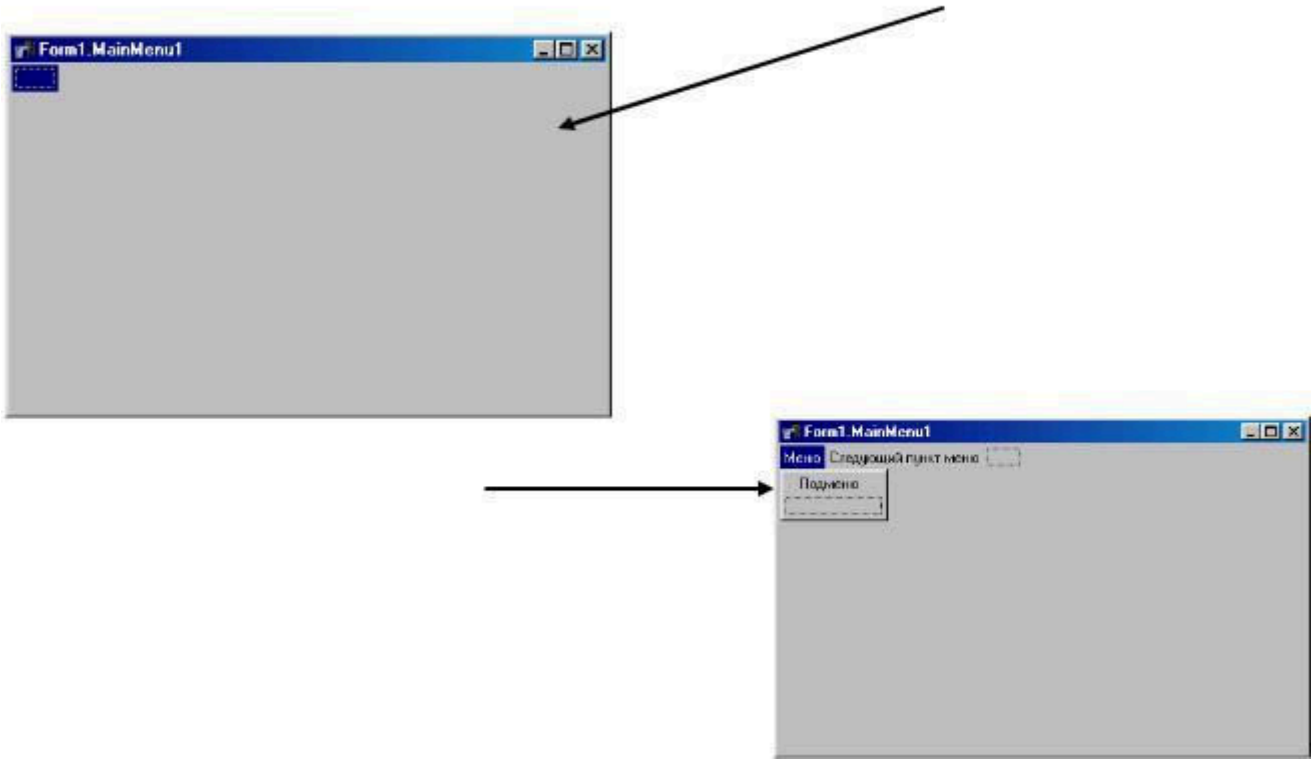
Обычно на форме достаточно одного главного меню и, соответственно, элемента MainMenu. В этом случае свойству Menu формы, на которой расположеноMainMenu, автоматически присваивается имя данного компонента, т.е. значение его свойства Name. Однако на одной форме возможно размещение нескольких компонентовMainMenu с различными разделами.

Тогда свойству Menu формы присваивается имя одного из них. А при выполнении программы это свойство можно изменять, преобразуя соответствующим образом разделы главного меню приложения.

Наиболее важным свойством MainMenu является свойствоItems. Его заполнение производится с помощью окнаMenu Designer (конструктора меню), которое можно вызвать тремя способами.

Во-первых,щелкнув мышью два раза по компонентуMainMenu.

---




2

Во-вторых,нажав на кнопку, расположенную в правой части свойства Items в инспекторе объектов.

Или нажать на элементе правую кнопку мыши и в контекстном меню выбрать команду Menu Designer. Результат любого из трех действий представлен на рис.

В этом окне проектируются меню. Для создания разделов в конструкторе меню необходимо выделить рамку, состоящую из точек, которая обозначает место расположения нового раздела. Затем в Object Inspector в свойствеCaption задать заголовок раздела (например, 'Меню'). После чего щелкнуть мышью на этом разделе - автоматически

появляется подменю этого раздела (снизу) и следующий раздел (справа) см. рис.

Следует отметить тот факт, что если был выбран пункт подменю и проделаны описанные выше действия, то создастся только следующий пункт подменю (ниже выбранного).

Используя мышь, есть возможность менять расположение разделов (порядок их следования).

При работе в Menu Designer можно использовать его локальное меню.

Его пункты выполняют следующие действия:

Insert - вставляет новый пункт меню;

Delete - удаляет активный пункт меню;

Create SubMenu - создает подменю для 'активного пункта меню;

Select Menu - открывает окно с расположенным в нем списком всех созданных в форме меню. Выбранное в этом окне меню открывается для редактирования;

Save As Template - сохраняет меню в качестве шаблона;

Insert From Template — добавляет меню из окна шаблона;

Delete Templates - удаляет шаблон из окна шаблона меню;

Insert From Resource - открывает диалоговое окно добавления меню из ресурсов.

Вторым вариантом создания нового раздела является использование команды Insert контекстного меню. В этом случае новый раздел вставится перед тем, рамка которого была выделена. Следует отметить, что перед такой вставкой предварительно должен быть выделенкакой-либораздел меню.

Каждый элемент свойства Items, которым является любой раздел, - это объект класса TMenuItem со своими свойствами, методами и событиями. Основные свойства класса TMenuItem описаны в табл.

Таблица Свойства класса TMenuItem

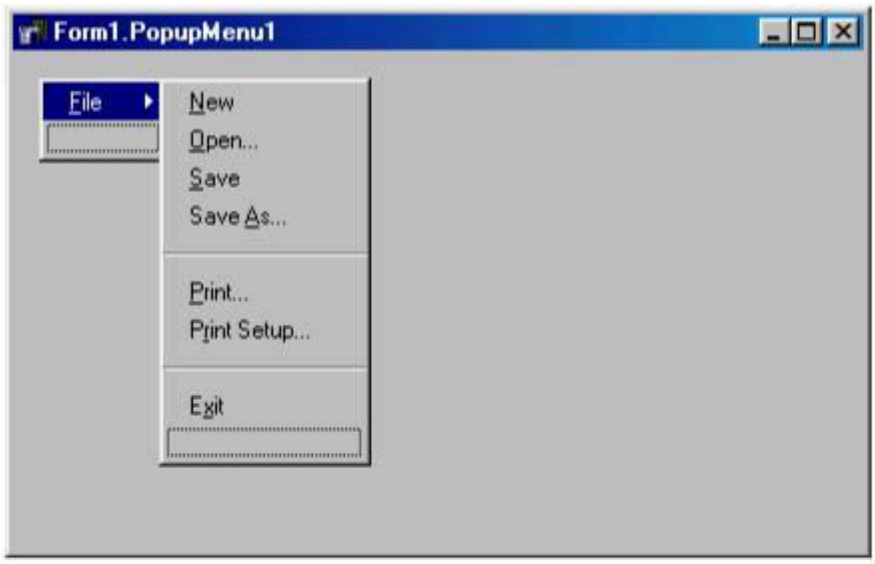
| Название | Описание |
|---|---|
| Bitmap | Указывает на изображение, которое будет размещено слева от подменю разделов (если есть изображение и не установлено свойство ImageIndex) |
| | 3 |
| Break | Предоставляет возможность разбивать подменю на столбцы (mbNone - отсутствие разбиения, mbBarBreak - вводится новый столбец, отделенный полосой, mbBreak - столбец, отделенный пробелами). |
| Caption | Содержит текст меню |
| Checked | Размещает рядом с элементом маркер флажка (при True) |
| Command | Используется при разработке программ, обращающихся |

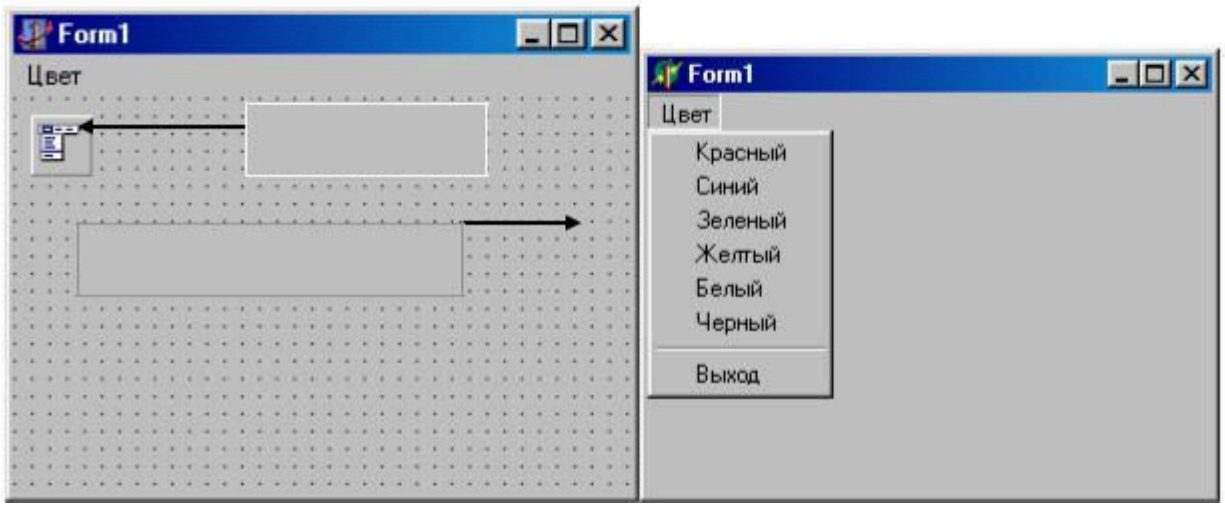
| | |
|---|---|
| | непосредственно к API-командам.Только для чтения |
| Count | Содержит количество элементов, содержащихся в свойстве Items. Используется только для чтения |
| Default | Определяет опцию по умолчанию (значение True). Такая опция выделяется цветом и выбирается двойным щелчком на родительской опции. В подменю может быть только одна умалчиваемая опция |
| Grouplndex | Задает номер группы для зависимых опций, определяемых в Radioltem |
| Imagelndex | Связывает изображение из компонента TListImage с опцией и присваивает значению свойства его индекс. Изображение появляется слева от опции, если свойство не равно -1 |
| Items | Позволяет обращаться к опции подчиненного меню по ее номеру |
| Menu Index | Вычисляет номер опции в родительском свойстве Items |
| Name | Предоставляет возможность задавать имя объекта, соответствующего разделу меню |
| Radioltem | Делает помеченным этот элемент из всех, у которых Grouplndex одинаков. Размещает рядом с элементом кружок (при True) |
| Shortcut | Предлагает установить код клавиши быстрого реагирования |

РорирМепи. Эти компоненты используются для создания локального (контекстного) меню. Локальное меню привязывается к конкретным компонентам. В отличие от главного меню, которое постоянно находится на экране, локальное меню выводится по мере необходимости. Для того чтобы вспомогательное меню компонента появилось на экране, необходимо установить фокус на этот компонент и нажать правую кнопку мыши. В локальное меню включаются команды, которые требуются при работе с компонентом в первую очередь. Как правило, такое меню являются одноуровневым, хотя это не обязательно.

Большинство оконных компонентов (рамки, редакторы текста и т.д.) содержат свойство РорирМепи, которое по умолчанию не заполнено. Посредством этого свойства и производится связывание компонента с локальным меню. В свойстве записывается имя соответствующего вспомогательного меню. Для создания локального меню необходимо поместить компонент РорирМепи на форму и, два раза кликнув по нему мышью, вызвать конструктор меню. Теперь можно создавать меню. Пункты локального меню будут добавляться только по вертикали. При разработке вспомогательного меню вMenu Designer также можно использовать его локальное меню.

Пусть необходимо, чтобы на форме осталось две кнопки, для одной из которых требуется создать локальное меню.


Form1.PopupMenu1
File
New
Open...
Save
Save As...
Print...
Print Setup...
Exit


Form1
Цвет
Form1
Цвет
Красный
Синий
Зеленый
Желтый
Белый
Черный
Выход

Для этого нужно поместить компонент РорирМепи на форму и разместить в нем какойнибудь пункт (например, стандартный пункт менюFile). Для размещения пунктаFile следует выполнить команду всплывающего менюInsert From Template редактора меню. В открывшемся окнеInsert Template выбираетсяFileMenu и нажимается кнопкаОК. Для привязки созданного локального меню к кнопке в ее свойствоPopupMenu заносится имя соответствующего компонентаРорирМепи. Теперь при установке курсора мыши на кнопкуОК и нажатии правой кнопки мыши появится всплывающее меню.

Основным событием раздела меню является событие OnClick, возникающее после щелчка пользователя на разделе (т.е. выборе пункта меню). В обработчик этого события помещаются все действия, которые необходимо выполнять при выборе соответствующего пункта меню. Чтобы написать программный код в обработчик OnClick, необходимо либо вMenu Designer, либо в редакторе форм два раза щелкнуть мышью по соответствующему пункту меню и ввести код.

Практическое задание № 1.

В качестве примера приложения с использованием меню будет рассмотрена программа,

реализующая простейшие возможности изменения цветового фона формы.

MainMenu1

Список пунктов меню

Проектируемая форма Работающее приложение

1.Свойства пустой формы: Color = clBtnFace

2.Создаем меню пользователя- размещаем компоненту MainMenu на форме, задаем пункт менюЦвет и подпункты менюКрасный Синий Зеленый Желтый Белый

Черный



3.Функциональность приложения: по нажатию на пункт меню (выбор цвета) меняется цвет формы. Первоначально после старта программы – серый цвет.

Обработчики событий для программы 2.
1. Выбор красного цвета 2. Выбор синего цвета
procedure TForm1.N2Click(Sender: TObject);procedure TForm1.N3Click(Sender: TObject);

```
begin
Form1.Color:=clRed;
end;
```

3. Выбор зеленого цвета

```
procedure TForm1.N4Click(Sender: TObject);
begin
Form1.Color:=clGreen;
end;
```

```
begin
Form1.Color:=clBlue;
end;
```

4. Выбор желтого цвета

```
procedure TForm1.N5Click(Sender: TObject);
begin
Form1.Color:=clYellow;
end;
```

5. Выбор белого цвета

```
procedure TForm1.N6Click(Sender: TObject);
begin
Form1.Color:=clWhite;
end;
```

6. Выбор черного цвета

```
procedure TForm1.N7Click(Sender: TObject);
begin
Form1.Color:=clBlack;
end;
```

7. Закрыть форму

```
procedure TForm1.N9Click(Sender: TObject);
begin
Close;
end;
```

Выполните компиляцию программы. После ее запуска на экран будет выведена форма приложения. Проверьте ее выполнение.

Самостоятельная работа № 1:

Реализуйте эту же функциональность приложения через локальное меню (РорирМепи).

Практическое задание № 2.

Написать программу «Текстовый блокнот», которая позволяет открывать текстовые файлы (*.txt), редактировать их и сохранять изменения. Программа должна иметь дружелюбный пользовательский интерфейс, а также главное меню программы.

Выполнение.

Создайте новый проект и сохраните его в отдельной папке. Самостоятельно назовите проект в соответствии с правилами создания имен переменных.

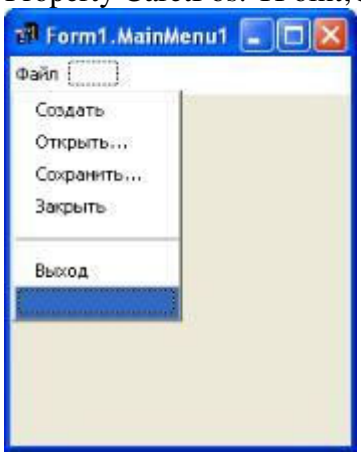
Чтобы создать простейший текстовый редактор нам понадобиться компонент

«Memo».

«TMemo» - многострочное редактируемое текстовое поле. Компоненты класса «TMemo» предназначены для ввода, редактирования и/или отображения достаточно длинного многострочного текста. Текст храниться в свойстве «Lines» класса «TStrings» и представляет собой пронумерованный набор строк (нумерация начинается с нуля). С помощью свойств и методов этого класса («Count», «Add», «Delete», «Clear» и т.д.) можно динамически формировать содержимое компонента.

Таблица 1. Основные свойства TMemo

| Свойство | Описание |
| --- | --- |
| Property CaretPos: TPoint; | Содержит координаты мигающего текстового курсора относительно границ клиентской области компонента |
| Property Lines: TStrings; | Содержит строки текста |
| Property ScrollBars: TScrollStyle; | Определяет наличие полос прокрутки |
| Property Text: String; | Отображает содержимое свойства «Lines» в виде одной длинной строки |






Контрольный вопрос: что означает слово «property»?

Самостоятельная работа №2: в справке найдите информацию о классе «TStrings».

Законспектируйте себе методы: «Add», «LoadFromFile», «SaveToFile», «Clear», «Delete», «AddString», «Insert», «IndexOf».

Для создания пользовательского меню нам понадобиться компонент «MainMenu». Этот компонент определяет главное меню программы. На форму можно помещать сколько угодно компонентов этого класса, но отображаться в строке меню в верхней части формы будет только тот из них, который указан в свойстве «Menu» формы.

После установки компонента на форму можно создать пункты меню. Для этого следует либо дважды щелкнуть мышкой на компоненте, либо вызвать контекстное меню и выбрать команду «Menu Designer», либо выбрать свойство «Items» в окне инспектора объектов. Создание пунктов меню происходит путем выбора свободной области и набора названия пункта. Обработка выбора конкретного пункта происходит при описании события «OnClick» соответствующего пункта.

Положите на форму редактор «Memo» и распределите его по всей форме. На него положите компонент «MainMenu». Вызовите «Menu Designer» и создайте меню как на рисунке. Для получения разделителя между пунктами «Закрыть» и «Выход» используйте символ «минус». После создания меню закройте

«Menu Designer»

Сохраните проект и проверьте на ошибки. Запустите приложение и проверьте работоспособность главного меню.

Контрольный вопрос: что означает свойство «Menu» у объекта «Form1»?

Контрольный вопрос: как создать вложенное меню (подменю)?

Для выполнения операций открытия и сохранения файлов нам понадобятся диалоговые окна. На палитре компонент найдите два компонента

«OpenDialog» и «SaveDialog». Поместите их на форму.

Настроить диалоговые окна очень просто. Выберите объект «OpenDialog1» и в инспекторе объектов определите свойства:

| | |
|---|---|
| DefaultExt | Указывает на расширение файла по-умолчанию |
| Filter | Составляется список фильтров. В редакторе фильтров в левой части напишите «Текстовые файлы», а в правой части напишите расширение «*.txt» |
| Title | Заголовок диалогового окна |

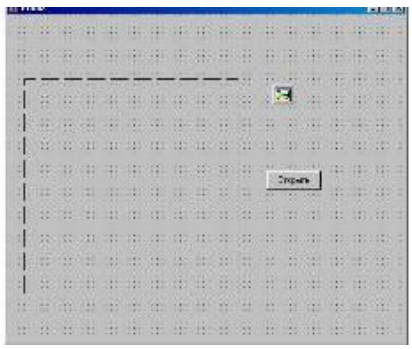
Самостоятельная работа: выполните самостоятельно заполнения свойств объекта

«SaveDialog1».

Следующий этап – описание событий каждого пункта меню. На главной форме вашего приложения выберите пункт «Файл – Открыть…». Откроется текстовый редактор с шаблоном процедуры. Опишите это событие, как показано ниже.

```
procedure TForm1.N3Click(Sender: TObject);begin

if OpenDialog1.Execute then //Если выбран файл и нажата кнопка "Открыть"

Memo1.Lines.LoadFromFile(OpenDialog1.FileName); //Загружаем файл в Memo

end;
```

Самостоятельная работа № 3: воспользуйтесь справкой и найдите методы для сохранения текста из «Memo».





Самостоятельная работа № 4: опишите самостоятельно остальные пункты меню, в которых следует реализовать такие команды, как Редактирование шрифтов, Настройка цвета.

В работе обязательно использование:

-использование стандартных диалогов для выбора шрифтов и его характеристик (FontDialog);

-использование стандартного диалога выбора цвета (ColorDialog);

-использование стандартного диалога печати и установки принтера

(PrintDialog, PrinterSetupDialog).

Практическое задание № 3.

Написать программу «Графический редактор», которая позволяет открывать графические файлы (*.bmp, *.jpeg), и сохранять. Программа должна иметь дружелюбный пользовательский интерфейс, а также главное меню программы.

Использовать диалоги для открытия и сохранения изображений (OpenPictureDialog, SavePictureDialog), изучить и научиться применять основные свойства и методы этих компонентов.

«Загрузка изображения из графического файла с использованием стандартного диалога открытия графических изображений»

Компонент Image позволяет поместить графическое изображение в любое место на форме. Для включения этого объекта в состав приложения необходимо выбрать его на страницеAdditional палитры компонентов и поместить в нужное место формы. Изображение можно загрузить во время дизайна в редакторе свойстваPicture (Инспектор Объектов). Файл изображения должен иметь форматBMP, WMF, ICO.

Для работы с изображениями в формате JPEG применяется специальный классTJPEGIMAGE. Чтобы использовать этот класс, необходимо в разделеUses подключить модульJPEG.

Свойство Picture определяет изображение, которое появится в поле компонента Image. При проектировании следует помнить, что изображение, помещенное на форму во время дизайна, включается в файл проекта DPR и затем присоединяется к исполняемому файлу. Поэтому такойEXE-файлможет получиться достаточно большим. В качестве альтернативы можно рассмотреть загрузку картинки во время выполнения программы.

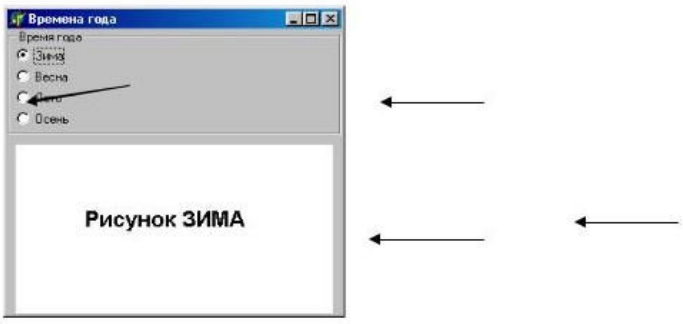
Свойство Center (тип Boolean) установить вTrue, центр изображения будет совмещаться с центромImage. СвойствоStretch установить вTrue, изображение будет сжиматься или растягиваться, чтобы заполнить весь объектImage.

Загрузка изображения из файла осуществляется с использованием метода LoadFromFile,

который имеет следующий синтаксис: LoadFromFile (const FileName: String);

Пример работы с графическим образом:

На форму помещается компонент Image1, кнопка Button1 с заголовком Открыть и диалог OpenPictureDialog1. Программа по нажатию кнопки вызывает диалог, затем открывает выбранный файл и помещает его содержимое в поле компонента Image1.




8

Код приложения «Графический образ»

procedure TForm2.Button1Click(Sender: TObject); begin

if OpenPictureDialog1.Executethen

// Если диалог отработал Image1.Picture.LoadFromFile(OpenPictureDialog1.FileName); end;

end.

Для очистки рисунка можно использовать обработку:

Image1.Picture:=nil;

Практическое задание № 4.

Написать программу, которая открывает указанные пользователем графические файлы, соответствующие временам года (Зима, Весна, Лето, Осень). Выбор файлов должен производиться с помощью стандартных диалоговых окон. Программа должна содержать элемент управления переключатели, для выбора времени года. Рисунки для просмотра находятся в папке Рисунки, для удобства выбора, заранее сгруппируйте их по темам.

Св-воItems:TString RadioGroup1

| | |
|---|---|
| Image1 | Св-воPicture меняем при выполнении программы, а не в Инспекторе объектов |

procedure TForm1.RadioGroup1Click(Sender: TObject); begin

case RadioGroup1.ItemIndexof //Выбрать из возможных вариантов

0:Image1.Picture.LoadFromFile('ris\zima.jpeg'); //Загрузка из файла графической

//информации

1:Image1.Picture.LoadFromFile('ris\vesna.jpeg');

2:Image1.Picture.LoadFromFile('ris\leto.jpeg');

3:Image1.Picture.LoadFromFile('ris\osen.jpeg');

end;

end;

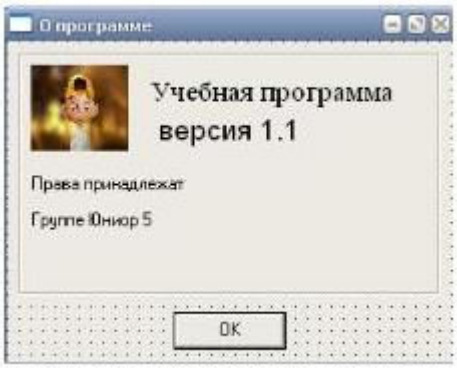
end.

Самостоятельная работа № 5

Измените обработчик события для загрузки изображения из выбранного файла в стандартном диалоге OpenPictureDialog1.

Самостоятельная работа № 6

Каждому студенту создать собственный вариант использования стандартного диалога для работы с изображениями.


9

Практическое задание № 5.

В большинстве программ существует возможность открывать специальное информационное окно программы «О программе», содержащее краткую справочную информацию о названии программного продукта, его версии, дате выпуска, авторах, фотографий авторов, контактный телефон, e-mailи другие данные.

Задание: Создать собственный вариант использования информационного окна.

В объект Image вставить фотографию разработчика.

Вызов информационного окна на экран обычно осуществляется в модальном режиме, т.е. с использованием метода ShowModal.

# Лабораторная работа № 16,17

# Тема: Логическое проектирование базы данных, нормализация отношений, операции над отношениями;

**Методические указания**

На этапе логического проектирования разрабатывается логическая модель базы данных. Так как в настоящее время наибольшее распространение получила реляционная модель, в этой работе рассматривается проектирование реляционной базы данных.

## Реляционная база данных

Реляционная модель состоит из трех частей:

-структурной части.

-целостной части.

-манипуляционной части.

Структурная часть описывает, какие объекты рассматриваются реляционной моделью. Считается, что единственной структурой данных, используемой в реляционной модели, являются взаимосвязанные отношения. Отношение можно представить в форме таблицы особого вида.Целостная часть описывает ограничения, которые должны выполняться для любых отношений в любых реляционных базах данных.Манипуляционная часть описывает способы манипулирования данными.

Вданной работе рассматривается структурная часть реляционной модели. Фундаментальными понятиями реляционной модели данных являются: отноше-

ние, атрибут, кортеж, первичный и внешний ключи.

Отношение это структурная единица реляционной модели, содержащая информацию об одном объекте предметной области. Имя отношения уникально в базе данных.Отношение, состоит из двух частей: заголовка и тела.Заголовок отношения, содержит фиксированное количество атрибутов.Атрибут определяет одно из свойств объекта. Имена атрибутов должны быть уникальны в пределах отношения.

Запись вида Имя отношения (список атрибутов)называют схемой отношения.

Тело отношения содержит множествокортежей отношения. Каждый кортеж представляет экземпляр объекта.

Первичный ключ – атрибут (группа атрибутов) значения которых уникальны в данном отношении, что обеспечивает уникальность каждого кортежа. Первичный ключ не может содержать нулевоезначение(null-значение).Внешний ключ – атрибут (группа атрибутов) вводимые в отношение, для установки связи. Назначение внешнего ключа, обеспечение связи с другими отношениями. Реляционная модель поддерживает три типа связи между отношениями: 1:1, 1:M, M:1. Ключи отношения могут бытьпростыми – состоящими из одного атрибута илисоставными – содержащими несколько атрибутов, а

такжеестественными – определяемыми из состава атрибутов отношения илиискусственными – вводимыми в отношение разработчиками базы данных.

Рассмотрим отношение Сотрудники, состоящее из атрибутов НомерСотрудника, Фамилия, Зарплата, НомерОтдела.

Пусть в данный момент отношение содержит три кортежа: (1,Иванов, 1000, 1)

(2, Петров, 2000, 2) (3, Сидоров, 3000, 1)

такое отношение естественным образом представляется в виде таблицы:

ФБУ ФИНАНСЫ И КРЕДИТ1

| НомерСотрудника | Фамилия | Зарплата | НомерОтдела |
|---|---|---|---|
| 1 | Иванов | 1000 | 1 |
| 2 | Петров | 2000 | 2 |
| 3 | Сидоров | 3000 | 1 |

Первичный ключ отношения – НомерСотрудника, внешний ключ –НомерОтдела (обеспечивает связь с отношениемОтделы(НомерОтдела, Наименование). Схема отношения имеет вид:Сотрудники(НомерСотрудника, Фамилия, Зарплата, Но-

мерОтдела).

Реляционной базой данных называется набор отношений.

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

Термины, которыми оперирует реляционная модель данных, имеют соответствующие "табличные" синонимы:

| Реляционный термин | Соответствующий "табличный" термин |
|---|---|
| База данных | Набор таблиц |
| Схема базы данных | Набор заголовков таблиц |
| Отношение | Таблица |
| Заголовок отношения | Заголовок таблицы |
| Тело отношения | Тело таблицы |
| Атрибут отношения | Наименование столбца таблицы |

Кортеж отношения  Строка таблицы

Хотя любое отношение можно изобразить в виде таблицы, нужно четко понимать, что отношения не являются таблицами. Это близкие, но не совпадающие понятия. Отношение обладает следующими свойствами:

− в отношении нет одинаковых кортежей.

− кортежи не упорядочены.

− атрибуты не упорядочены.

− все значения атрибутов атомарны, т.е. неделимы.

Вэтих свойствах в основном и состоят различия между отношениями и таблица-

ми.

# Особенности проектирования реляционной базы данных

Проектирование реляционной базы данных проходит в том же порядке, что и проектирование БД других моделей, но имеет свои особенности. Проблема логического проектирования реляционной базы данных состоит в обоснованном принятии решений о том:

-из каких отношений должна состоять база данных;

-какие атрибуты должны быть у этих отношений.

Проектирование схемы БД должно решать задачи минимизации дублирования данных и упрощения процедур их обработки и обновления. При неправильно спроектированной схеме БД могут возникнуть аномалии модификации данных. Для решения подобных проблем проводится нормализация отношений.



Нормализация отношения это пошаговый процесс декомпозиции(разбиения) исходного отношения на более простые отношения. Каждый этап данного процесса приводит схему базы данных к определенной нормальной форме. Каждая следующая нормальная форма обладает “лучшими свойствами”, чем предыдущая. Окончательная цель нормализации сводится к получению такой базы данных, в которой каждый факт появляется лишь в одном месте, то есть, исключена избыточность данных.

В теории реляционных баз данных принято выделять следующую последовательность нормальных форм:

-первая нормальная форма (1НФ);

-вторая нормальная форма (2НФ);

-третья нормальная форма (3НФ);

-нормальная форма Бойса-Кодда(НФБК);

-четвертая нормальная форма (4НФ);

-нормальные формы более высоких порядков.

Каждой нормальной форме соответствует некоторый набор ограничений. Отношение находится в определенной нормальной форме, если оно удовлетворяет набору ограничений этой формы.

Процесс нормализации основан на понятии функциональной зависимости атрибутов. Атрибут Y функционально зависит от атрибута X (обозначается X®Y, читается как “X функционально (или однозначно) определяет Y”), если в любой момент времени каждому значению атрибута X соответствует единственное значение атрибута Y.Неключевым атрибутом называется любой атрибут отношения, не входящий в состав первичного ключа. Если неключевой атрибут зависит от составного первичного ключа в целом и не зависит от его части, то говорят ополной функциональной зависимости атрибута от составного первичного ключа. Два или более атрибутавзаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

Первая нормальная форма

Отношение находится в 1НФ, если значения атрибутов атомарны и все неключевые атрибуты функционально зависят от первичного ключа.

Вторая нормальная форма

Отношение находится во2НФ, если выполняются ограничения1НФ, и каждый неключевой атрибут находится в полной функциональной зависимости от составного первичного ключа.

Для того чтобы привести отношение ко2НФ, нужно:

1.исключить из исходного отношения неключевые атрибуты, которые не находятся в полной функциональной зависимости от составного первичного ключа;

2.создать новое отношение (отношения) включив в него часть составного первичного ключа и атрибуты, функционально зависящие от этой части.

Отношение с простым первичным ключом автоматически находится во 2НФ.

Третья нормальная форма

Отношение находится в 3НФ, если выполняются ограничения 2НФ, и все неключевые атрибуты взаимно независимы и полностью зависят от первичного ключа.

Для того чтобы привести отношение к 3НФ, нужно:

1.исключить из исходного отношения атрибуты, которые функционально зависят от неключевого атрибута;

2.создать новое отношение (отношения) включив в него функционально зависимые неключевые атрибуты.

Отношение с одним неключевым атрибутом автоматически находится в 3НФ.

ФБУ ФИНАНСЫ И КРЕДИТ3

Третья нормальная форма базы данных в большинстве случаев считается достаточной для достижения целей нормализации. Приведением отношений к третьей нормальной форме процесс проектирования реляционной базы данных обычно заканчивается

# Пример проектирования реляционной базы данных (метод нормализации)

Рассмотрим в качестве предметной области некоторую организацию, выполняющую некоторые проекты. Модель предметной области опишем следующим неформальным текстом:

1.Сотрудники организации выполняют проекты.

2.Проекты состоят из нескольких заданий.

3.Каждый сотрудник может участвовать в одном или нескольких проектах, или временно не участвовать ни в каких проектах.

4.Над каждым проектом может работать несколько сотрудников, или временно проект может быть приостановлен, тогда над ним не работает ни один сотрудник.

5.Над каждым заданием в проекте работает только один сотрудник.

6.Каждый сотрудник числится в одном отделе.

7.Каждый сотрудник имеет телефон, находящийся в отделе.

Входе дополнительного уточнения того, какие данные необходимо учитывать, выяснилось следующее:

1.О каждом сотруднике необходимо хранить табельный номер и фамилию. Табельный номер является уникальным для каждого сотрудника.

2.Каждый отдел имеет уникальный номер.

3.Каждый проект имеет номер и наименование. Номер проекта является уникальным.

4.Каждая работа из проекта имеет номер, уникальный в пределах проекта. Работы в разных проектах могут иметь одинаковые номера.

Входе логического проектирования на первом шаге предлагается хранить данные

водном отношении, имеющем следующие схему:

СотрудникиОтделыПроекты (НомерСотрудника, Фамилия,

НомерОтдела, Телефон, НомерПректа, Проект, НомерЗадания), где НомерСотрудника - табельный номер сотрудника Фамилияфамилия сотрудника

НомерОтдела - номер отдела, в котором числится сотрудник Телефон - телефон сотрудника НомерПроекта - номер проекта, над которым работает сотрудник

Проект - наименование проекта, над которым работает сотрудник НомерЗадания - номер задания, над которым работает сотрудник

Так как каждый сотрудник в каждом проекте выполняет ровно одно задание, то в качестве первичного ключа отношения необходимо взять пару атрибутов{Номер-

Сотрудника, НомерПроекта}.

В текущий момент состояние предметной области отражается следующими фактами:

-Сотрудник Иванов, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 1 и во втором проекте "Климат" задание 1.

-Сотрудник Петров, работающий в 1 отделе, выполняет в первом проекте "Космос" задание 2.

-Сотрудник Сидоров, работающий во 2 отделе, выполняет в первом проекте "Космос" задание 3 и во втором проекте "Климат" задание 2.

Это состояние отражается в таблице (полужирным курсивом выделены ключевые атрибуты, составной первичный ключ):



| Номер Сотрудника | Фамилия | Номер Отдела | Телефон | Номер Проекта | Проект | Номер Задания |
|---|---|---|---|---|---|---|
| 1 | Иванов | 1 | 11-22-33 | 1 | Космос | 1 |
| 1 | Иванов | 1 | 11-22-33 | 2 | Климат | 1 |
| 2 | Петров | 1 | 11-22-33 | 1 | Космос | 2 |
| 3 | Сидоров | 2 | 33-22-11 | 1 | Космос | 3 |
| 3 | Сидоров | 2 | 33-22-11 | 2 | Климат | 2 |

Первая нормальная форма Отношение СотрудникиОтделыПроектынаходится в 1НФ.

Все атрибуты имеют атомарные значения.

Определен составной первичный ключ, {НомерСотрудника,НомерПроекта}. Все неключевые атрибуты функционально зависят от первичного ключа.

{НомерСотрудника,НомерПроекта} Фамилия {НомерСотрудника,НомерПроекта} НомерОтдела {НомерСотрудника,НомерПроекта} Телефон {НомерСотрудника,НомерПроекта} Проект {НомерСотрудника,НомерПроекта} НомерЗадания

Одного взгляда на таблицу отношения СотрудникиОтделыПроекты достаточно, чтобы увидеть, что данные хранятся в ней сбольшой избыточностью. Во многих строках повторяются фамилии сотрудников, номера телефонов, наименования проектов. Кроме того, в данном отношении хранятся вместе независимые друг от друга данные - и данные о сотрудниках, и об отделах, и о проектах, и о работах по проектам. Пока никаких действий с отношением не производится, это не страшно. Но как только состояние предметной области изменяется, то, при попытке соответствующим образом изменить состояние базы данных, возникает большое количество проблем.

Исторически эти проблемы получили названиеаномалий. Понятие аномалии в данном случае рассматривается, как неадекватности модели данных предметной области, (что говорит на самом деле о том, что логическая модель данных попросту неверна!)

Т.к. аномалии проявляют себя при выполнении операций, изменяющих состояние базы данных, то различают следующие виды аномалий:

-Аномалии добавления

-Аномалии обновления

-Аномалии удаления

Аномалии добавления. В отношение СотрудникиОтделыПроектынельзя до-

бавить данные о сотруднике, который пока не участвует ни в одном проекте. Действительно, если, например, во втором отделе появляется новый сотрудник, скажем, Пушников, и он пока не участвует ни в одном проекте, то мы должны вставить в отношение кортеж (4, Пушников, 2, 33-22-11,null, null, null). Это сделать невозможно, т.к. атрибутНомерПроекта (номер проекта) входит в состав первичного ключа, и, следовательно, не может содержатьnull-значений.

Аномалии обновления. Фамилии сотрудников повторяются во многих кортежах отношения. Поэтому если сотрудник меняет фамилию, то такие изменения необходимоодновременно выполнить во всех местах, где эта фамилия встречается, иначе отношение станет некорректным (например, один и тот же сотрудник в разных кор-

тежах будет иметь разные фамилии). Таким образом, обновление базы данных одним действием реализовать невозможно.

Аномалии удаления. При удалении некоторых данных может произойти потеря другой информации. Например, если закрыть проект "Космос" и удалить все строки, в которых он встречается, то будут потеряны все данные о сотруднике Петрове.

Для устранения указанных аномалий продолжим процесс нормализации.

Вторая нормальная форма

В отношении СотрудникиОтделыПроекты можно привести следующие примеры функциональных зависимостей неключевых атрибутов от части составного первичного ключа:

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника:

НомерСотрудника Фамилия НомерСотрудникаНомерОтдела НомерСотрудникаТелефон

Зависимость наименования проекта от номера проекта:

НомерПроекта$\rightarrow$Проект

Замечание. Приведенные функциональные зависимостине выведены из внешнего вида отношения, приведенного в таблице. Эти зависимости отражают взаимосвязи, обнаруженные между объектами предметной области и являются дополнительными ограничениями, определяемыми предметной областью. Таким образом, функциональная зависимость -семантическое понятие. Она возникает, когда по значениям одних данных в предметной области можно определить значения других данных. Например, зная табельный номер сотрудника, можно определить его фамилию, по номеру отдела можно определить телефона. Функциональная зависимостьзадает дополнительные ограничения на данные, которые могут храниться в отношениях.

Отношение СотрудникиОтделыПроекты не соответствует требованиям2НФ, т.к. есть атрибуты, зависящие от части составного первичного ключа.

Зависимость атрибутов, характеризующих сотрудника от табельного номера сотрудника является зависимостью от части составного первичного ключа. Зависимость наименования проекта от номера проекта является зависимостью от части составного первичного ключа.

Для того чтобы устранить зависимость атрибутов от части составного первичного ключа, нужно произвести декомпозицию (разбиение) отношения на несколько от-

ношений. При этом те атрибуты, которые зависят от части составного ключа,

выносятся в отдельное отношение.

Отношение СотрудникиОтделыПроекты декомпозируем на три отношения-

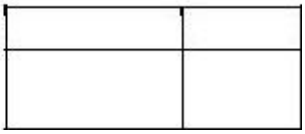
СотрудникиОтделы, Проекты, Задания.

Отношение СотрудникиОтделы (НомерСотрудника, Фамилия, НомерОтдела,

Телефон)

| Номер Сотрудника | Фамилия | Номер Отдела | Телефон |
|---|---|---|---|
| 1 | Иванов | 1 | 11-22-33 |
| 2 | Петров | 1 | 11-22-33 |
| 3 | Сидоров | 2 | 33-22-11 |





Отношение Проекты(НомерПректа, Проект)

| Номер Проекта | Проект |
|---|---|
| 1 | Космос |
| 2 | Климат |

Отношение Задания(НомерСотрудника, НомерПроекта, НомерЗадания):

| НомерСотрудника | НомерПроекта | НомерЗадания |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 2 | 1 |
| 2 | 1 | 2 |
| 3 | 1 | 3 |
| 3 | 2 | 2 |

Отношения, полученные в результате декомпозиции, находятся во 2НФ. Действительно, отношения СотрудникиОтделы иПроекты имеют простые ключи, следовательно, автоматически находятся во2НФ, отношениеЗадания имеет составной первичный ключ, но единственный неключевой атрибут НомерЗадания функционально полно зависит от первичного ключа{НомерСотрудника, НомерПроекта}. Часть аномалий обновления устранена.

Третья нормальная форма

Отношение СотрудникиОтделы не находится в 3НФ, т.к. имеется функциональная зависимость неключевых атрибутов(зависимость номера телефона от номера отдела):

НомерОтдела $\rightarrow$Телефон Для того чтобы, устранить функциональную зависимость неключевых атрибутов,

нужно произвести декомпозицию отношения на несколько отношений. При этом те неключевые атрибуты, которые являются зависимыми, выносятся в отдельное от-

ношение.

Отношение СотрудникиОтделы декомпозируем на два отношения –Сотрудни-

ки и Отделы.

Отношение Сотрудники (НомерСотрудника, Фамилия,НомерОтдела):

НомерСотрудникаФамилияНомерОтдела

1 Иванов 1
2 Петров 1
3 Сидоров 2

Отношение Отделы (НомерОтдела, Телефон):

НомерОтдела Телефон

111-22-33
233-22-11

Обратим внимание на то, что атрибут НомерОтдела,не являвшийся ключевым в

отношении СотрудникиОтделы, становится первичным ключомв отношении От-

делы. Именно за счет этого устраняется избыточность, связанная с многократным хранением одних и тех же номеров телефонов. АтрибутНомерОтдела в отношенииСотрудники становится внешним ключ, для обеспечения связи между отношениями

Таким образом, все обнаруженные аномалии устранены. Реляционная модель со-

стоит из четырех отношений: Сотрудники, Отделы, Проекты, Задания. Каждое отношение соответствует третьей нормальной форме. База данных является адекватной описанной модели предметной области.

Схема базы данных имеет следующее описание:

Отделы (НомерОтдела, Телефон):

Сотрудники (НомерСотрудника, Фамилия,НомерОтдела):Проекты (НомерПректа, Проект)

Задания (НомерСотрудника, НомерПроекта, НомерЗадания):

Схема БД может иметь также графическое представление.

# Создание схемы базы данных в Microsoft Visio

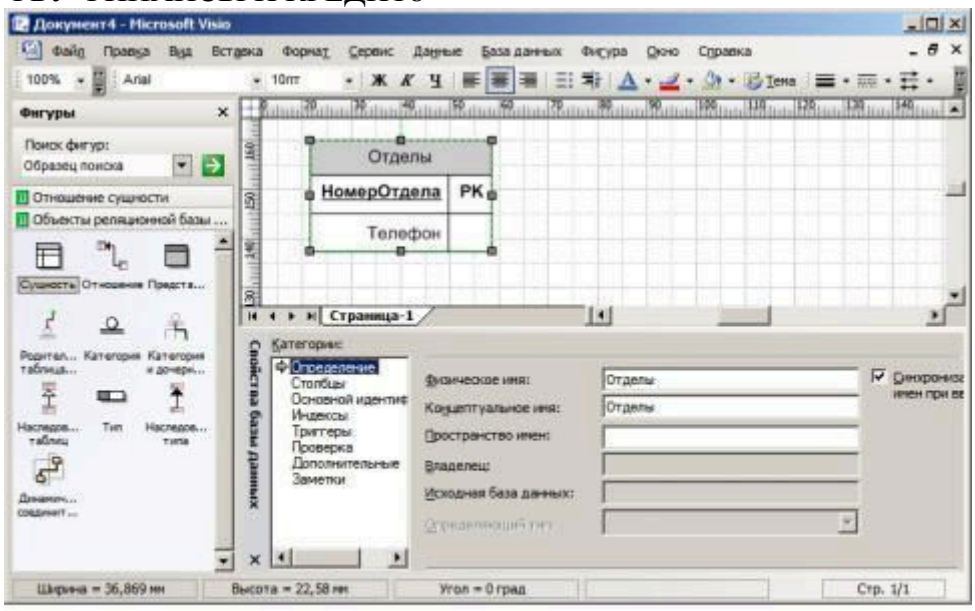
1.Загрузить Microsoft Visio. Выбрать категорию “Database” (Базы данных). Выбрать шаблон “Database model diagram” (Схема модели базы данных).

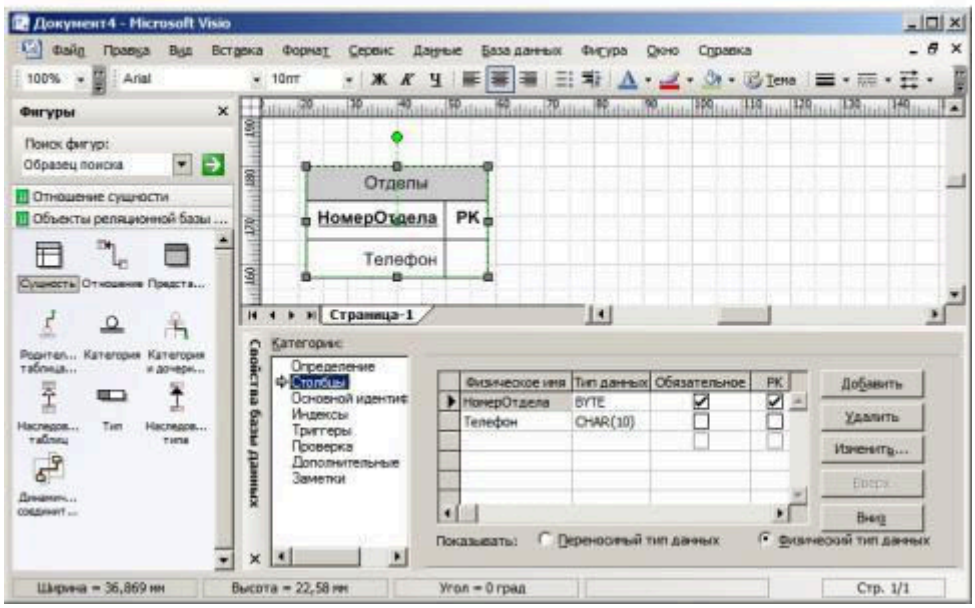
2.Использовать фигуру “Entity” (Сущность) для добавления таблицы.

3.Открыть окно “Database properties” (Свойства базы данных) двойным щелчком по фигуре “Entity” или используя контекстное меню.

4.Для каждой таблицы в окне “Database properties” ввести “Физическое имя:” – имя отношения.

ФБУ ФИНАНСЫ И КРЕДИТ8

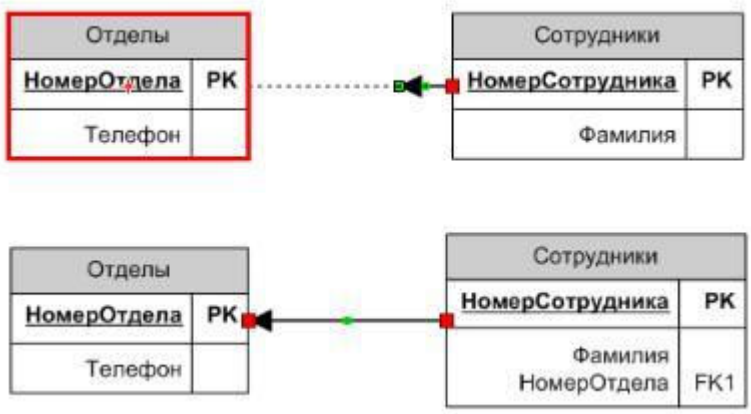


5.Для каждого атрибута ввести имя, выбрать тип данных (BYTE - числовой, CHAR(10) - символьный, DATETIME – дата/время, CURRENCY –денежный).

Если атрибут является первичным ключом установить флажокPK (Primary Key, первичный ключ). Внешние ключи не описывать, они добавляются автоматически при установке связи между таблицами.

ФБУ ФИНАНСЫ И КРЕДИТ9




6.Использовать фигуру “Relationship” (Отношение) для изображения связи между таблицами. Начало линии связи установить на подчиненную (дочернюю) сущность, конец линии (стрелка), на главную (родительскую) сущность. Связь считается установленной в том случае, если сущность выделена красным прямоугольником.

Обратите внимание на сущность “Сотрудники”, в неё добавился внешний ключ

(FK1, Forent key) “НомерОтдела”.

Получение реляционной схемы из модели “объект-связь”Шаг 1. Каждый объект превращается в отношение. Имя объекта становится

именем отношения.

Шаг 2. Каждое свойство объекта становится атрибутом отношения. Уточняется имя атрибута.

Шаг 3. Идентификатор объекта превращаются в первичный ключ отношения. Если у объекта идентификатор не определен, первичный ключ отношения определяется из состава атрибутов или вводится искусственный первичный ключ.

Шаг 4. Связи "один к одному" становятся внешними ключами. Для этого первичный ключ одного отношения участвующего в связи вводится в другое отношение в качестве внешнего ключа

Шаг 5. Связи "один ко многим" становятся внешними ключами. Для этого первичный ключ отношения участвующего в связи со стороны“один” вводится в отношение участвующее в связи со стороны “многие” в качестве внешнего ключа.

Шаг 6. Связи "многие ко многим" становятся новым отношением, состоящим из первичных ключей отношений участвующих в связи. Таким образом, связь "многие ко многим" преобразуется в связи "один ко многим”.

Шаг 7. Выполняются шаги по нормализации полученных отношений и приведение их к третьей нормальной форме.

Преобразование диаграммы “объект/связь” разработанной в лабораторной работе №1 в реляционную модель данных.

Схемы отношений:

Покупатель(КодПокупателя, Наименование, Адрес, Банк);Склад(НомерСклада, Наименование);

Накладная(НомерНакладной, Дата, Сумма, КодПокупателя, НомерСклада);

Товар(НомерТовара, Наименование, ЕдиницаИзмерения, ТекущаяЦена);

СписокТоваров(НомерНакладной, НомерТовара, Количество, Цена);

оварНаСкладе(НомерСклада, НомерТовара, Количество).

Схема базы данных:

## Самостоятельная работа

Разработать в представленных ниже задачах схему БД в третьей нормальной форме. Вариант задания указывается преподавателем. Отчет о выполнении задания подготовить в MS Word. Для построения схемы базы данных в графическом виде исполь-

зовать MS Visio.

### Технология выполнения

1.Описать модель предметной области неформальным текстом.

2.Уточнить набор данных.

3.Создать универсальное отношение. Записать схему отношения. Описать содержание атрибутов.

4.Определить первичный ключ отношения.

5.Описать текущее состояние предметной области. Представить в виде таблицы.

6.Проверить соответствие отношения требованиям1НФ. Если необходимо провести нормализацию отношения. Записать схему отношения и содержание таблицы.

7.Проверить соответствие отношения требованиям2НФ. Если необходимо провести нормализацию отношения. Записать схему базы данных и содержание таблиц.

8.Проверить соответствие отношения требованиям3НФ. Если необходимо провести нормализацию отношений. Записать схему базы данных в текстовом виде. Представить схему базы данных и графической виде и содержание таблиц.

ФБУ ФИНАНСЫ И КРЕДИТ

## Вариант 1

Построить базу данных, описывающую результаты сессии. Информация должна содержать номер семестра, сведения о студенте (ФИО, группа, специальность), сведения о сдаваемом предмете(название, семестр), дату сдачи экзамена, оценку и ФИО экзаменатора.

## Вариант 2

Построить базу данных, описывающую работу библиотеки с читателем. Информация должна содержать сведения о читателе(ФИО, адрес, телефон), информацию о выданной книге (название, автор, издательство) и дату выдачи книги.

## Вариант 3

Построить базу данных, описывающую работу отдела кадров по ведению личных дел сотрудников. Информация должна содержать сведения о сотрудниках(табельный номер, ФИО, отдел (место работы), должность, оклад), отделах (наименование, ФИО начальника, ФИО сотрудников, количество сотрудников).

## Вариант 4

Построить базу данных, описывающую работу с заказами некоторой оптовой базы. Информация должна содержать сведения о заказчике (Название фирмы, адрес, телефон), сведения о заказываемом товаре (Наименование, фирма изготовитель, год выпуска, стоимость единицы продукции), а также количество заказанного товара.

## Вариант 5

Построить базу данных, описывающую формирование фонда сети магазинов некоторой фирмы. Информация должна содержать сведения о магазине(название, адрес, телефон), сведения о поставщике (наименование, адрес, телефон) сведения о товаре (наименование, количество) и дату поставки.

## Вариант 6

Построить базу данных, описывающую работу с клиентами фирмы по техническому обслуживанию торгового оборудования. Информация должна собираться о мастерах, выполняющих ремонтные работы (ФИО, квалификация, телефон), о магазинах, подающих заявки на ремонт оборудования(наименование оборудования, магазин, адрес, телефон) и о выполнении заказа с указанием даты выполнения и оплате.

### Вариант 7

Построить базу данных, описывающую работу с вкладчиками в отделении банка. База данных должна содержать сведения о сотрудниках (табельный номер, ФИО, должность), клиентах (ФИО, номер паспорта, наименование вклада, дата вклада, сумма вклада, ФИО сотрудника), вкладах (наименование, минимальный срок, минимальная сумма, процентная ставка).

### Вариант 8

Построить базу данных, описывающую работу с заказами в авторемонтной мастерской. База данных должна содержать сведения о клиенте(ФИО, адрес), тип работы, оплату и информацию об исполнителе (ФИО, квалификация).

### Вариант 9

Построить базу данных, описывающую поиск книг в каталоге библиотеки. Информация должна содержать сведения о жанрах(наименование, описание), издательствах (наименование, город), книгах (название, автор, издательство, год издания, жанр).

## Контрольные вопросы

1.Каковы задачи, решаемые на этапе логического проектирования?

2.Каковы базовые свойства реляционной модели данных?

ФБУ ФИНАНСЫ И КРЕДИТ12

3.В чем состоят требования структурной части реляционной модели данных?

4.В чем состоят требования манипуляционной части реляционной модели дан-

ных?

5.Дайте определение отношения реляционной модели данных.

6.Дайте определение схемы отношения, схемы базы данных.

7.Что такое первичный ключ отношения?

8.Что такое внешний ключ отношения?

9.Перечислите фундаментальные свойства отношений.

10.Какие требования должны удовлетворяться в процессе логического проектирования базы данных?

11.Какие неудобства влекут за собой аномалии обновления?

12.Какие неудобства влекут за собой аномалии удаления?

13.Какие неудобства влекут за собой аномалии добавления?

14.В чем состоит процесс нормализации отношений?

15.Каковы общие свойства нормальных форм?

16.Дайте определение функциональной зависимости.

17.Дайте определение функционально полной зависимости.

18.Что такое взаимно независимые атрибуты?

19.Какие условия должны выполняться, чтобы отношение находилось в первой нормальной форме?

20.Каковы негативные последствия влечет нахождение отношения лишь в первой нормальной форме?

21.Какие условия должны выполняться, чтобы отношение находилось во второй нормальной форме?

22.Какие условия должны выполняться, чтобы отношение находилось в третьей нормальной форме?

23.Что понимается под полной декомпозицией отношения?

24.Какова процедура получения реляционной схемы РБД из ER-диаграммы?

## Лабораторная работа № 18,19

## Инфологическое проектирование, составление ER–диаграммы и схемы отношений

Цель работы:

Научиться проектировать структуру (инфологическую модель) реляционной базы данных.

Результат:

К защите работы должна быть представлена концептуальная (инфологическая) модель данных, спроектированная двумя методами (сущность-связь, методом функциональных зависимостей), выполненная в MS Word, и модель БД, представленная в ErWin.

Общая информация

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

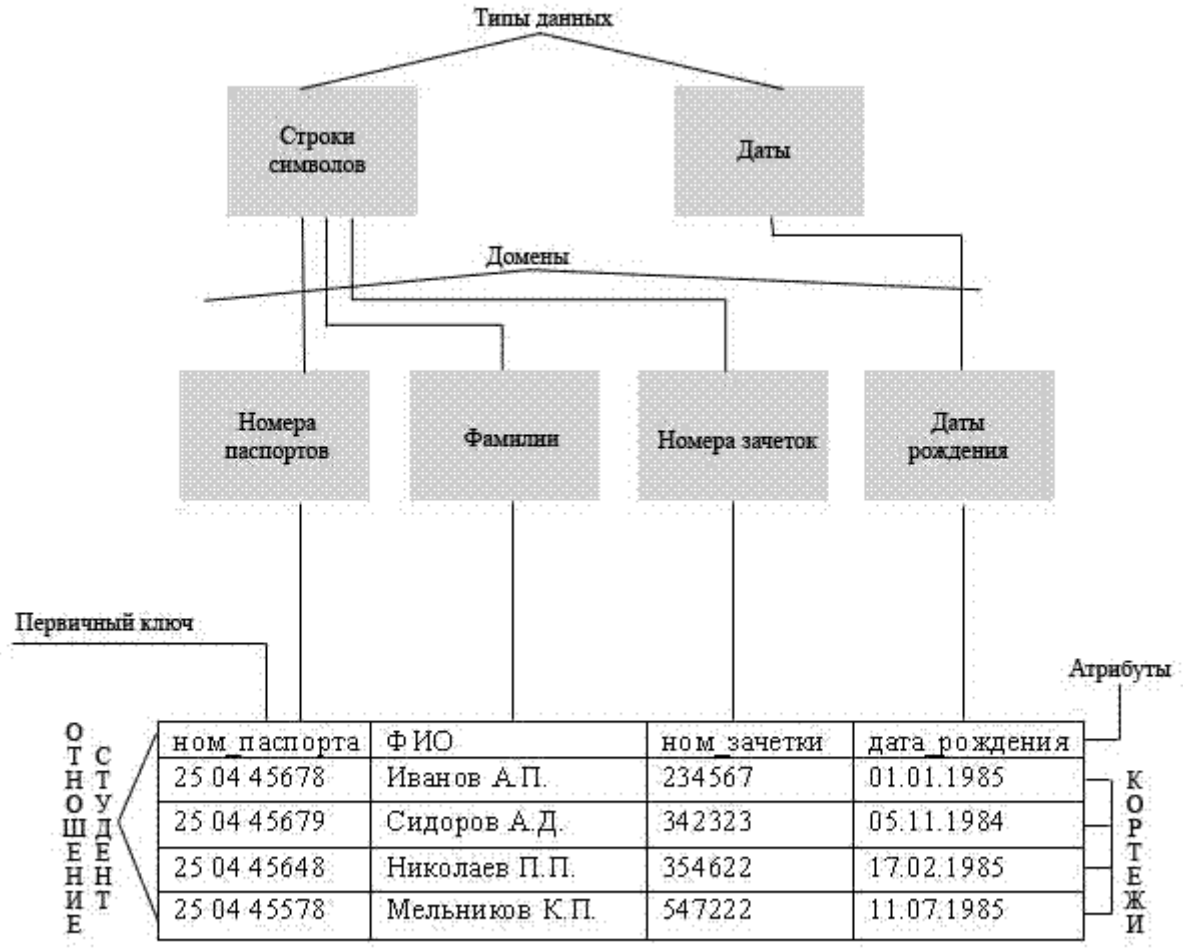



рис. 1 Основные понятия реляционных баз данных

Для начала покажем смысл этих понятий на примере отношения СТУДЕНТ, содержащего информацию о студентах (рис. 1).

Понятие **тип данных** в реляционной модели данных полностью адекватно понятию типа данных в языках программирования. Обычно в современных реляционных БД допускается хранение символьных, числовых данных, битовых строк, специализированных числовых данных (таких как "деньги"), а также специальных временных данных (дата, время, временной интервал). Достаточно активно развивается подход к расширению возможностей реляционных систем абстрактными типами данных (соответствующими возможностями обладают, например, системы семейства Ingres/Postgres). В нашем примере мы имеем дело с данными двух типов: строки символов и даты.

Понятие **домена** более специфично для баз данных, хотя и имеет некоторые аналогии с подтипами в некоторых языках программирования. В самом общем виде домен определяется заданием некоторого базового типа данных, к которому относятся элементы домена, и произвольного логического выражения, применяемого к элементу типа данных.

Следует отметить также семантическую нагрузку понятия домена: данные считаются сравнимыми только в том случае, когда они относятся к одному домену. В нашем примере значения доменов "Номера паспортов" и "Фамилии" относятся к типу строк символов, но не являются сравнимыми. Заметим, что в большинстве реляционных СУБД задание домена не является обязательным.

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

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

**Кортеж**, соответствующий данной схеме отношения, - это множество пар {имя атрибута, значение}, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения. "Значение" является допустимым значением домена данного атрибута (или типа данных, если понятие домена не поддерживается).

**Отношение** - это множество кортежей, соответствующих одной схеме отношения.

Обычным житейским представлением отношения является таблица, заголовком которой является схема отношения, а строками - кортежи отношения-экземпляра; в этом случае имена атрибутов именуют столбцы этой таблицы. Поэтому иногда говорят "столбец таблицы", имея в виду "атрибут отношения".

Остановимся теперь на некоторых важных свойствах отношений, которые следуют из приведенных ранее определений:

1. **Отсутствие кортежей-дубликатов** следует из определения отношения как множества кортежей. В классической теории множеств по определению каждое множество состоит из различных элементов. Из этого свойства вытекает наличие у каждого отношения так называемого первичного ключа - набора атрибутов, значения которых однозначно определяют кортеж отношения. Для каждого отношения по крайней мере полный набор его атрибутов обладает этим свойством. Однако при формальном определении первичного ключа требуется обеспечение его "минимальности", т. е. в набор атрибутов первичного ключа не должны входить такие атрибуты, которые можно отбросить без ущерба для основного свойства - однозначно определять кортеж. Понятие первичного ключа является исключительно важным в связи с понятием целостности баз данных.

2. **Отсутствие упорядоченности кортежей**. Свойство отсутствия упорядоченности кортежей отношения также является следствием определения отношения-экземпляра как множества кортежей. Отсутствие требования к поддержанию порядка на множестве кортежей отношения дает дополнительную гибкость СУБД при хранении баз данных во внешней памяти и при выполнении запросов к базе данных. Это не противоречит тому, что при формулировании запроса к БД, например, на языке SQL можно потребовать сортировки результирующей таблицы в соответствии со значениями некоторых столбцов. Такой результат, вообще говоря, не отношение, а некоторый упорядоченный список кортежей.

3. **Отсутствие упорядоченности атрибутов.** Атрибуты отношений не упорядочены, поскольку по определению схема отношения есть множество пар {имя атрибута, имя домена}. Для ссылки на значение атрибута в кортеже отношения всегда используется имя атрибута. Это свойство теоретически позволяет, например, модифицировать схемы существующих отношений не только путем добавления новых атрибутов, но и путем удаления существующих атрибутов. Однако в большинстве существующих систем такая возможность не допускается, и хотя упорядоченность набора атрибутов отношения явно не требуется, часто в качестве неявного порядка атрибутов используется их порядок в линейной форме определения схемы отношения.

- Задать вопрос

**4. Атомарность значений атрибутов**. Значения всех атрибутов являются атомарными. Это следует из определения домена как потенциального множества значений простого типа данных, т. е. среди значений домена не могут содержаться множества значений (отношения).

**Элементы модели "сущность-связь"**

В реальном проектировании структуры базы данных применяются метод - так называемое, семантическое моделирование. Семантическое моделирование представляет собой моделирование структуры данных, опираясь на смысл этих данных. В качестве инструмента семантического моделирования используются различные варианты диаграмм сущность-связь (ER - Entity-Relationship).

Первый вариант модели сущность-связь был предложен в 1976 г. Питером Пин-Шэн Ченом В дальнейшем многими авторами были разработаны свои варианты подобных моделей (нотация Мартина, нотация IDEF1X, нотация Баркера и др.). Кроме того, различные программные средства, реализующие одну и ту же нотацию, могут отличаться своими возможностями. По сути, все варианты диаграмм сущность-связь исходят из одной идеи - рисунок всегда нагляднее текстового описания. Все такие диаграммы используют графическое изображение сущностей предметной области, их свойств (атрибутов), и взаимосвязей между сущностями.

Мы опишем работу с упрощенными по отношению к нотации Баркера ER-диаграммами.

Основные понятия ER-диаграмм

**Сущность** - это класс однотипных объектов, информация о которых должна быть учтена в модели.


рис. 2 Сущность

Каждая сущность должна иметь наименование, выраженное существительным в единственном числе.

Примерами сущностей могут быть такие классы объектов как "Поставщик", "Сотрудник", "Накладная".

Каждая сущность в модели изображается в виде прямоугольника с наименованием (рис 2.).

**Экземпляр сущности** - это конкретный представитель данной сущности.

Например, представителем сущности "Студент" может быть "Студент Иванов".

Экземпляры сущностей должны быть различимы, т. е. сущности должны иметь некоторые свойства, уникальные для каждого экземпляра этой сущности.

**Атрибут сущности** - это именованная характеристика, являющаяся некоторым свойством сущности.

Наименование атрибута должно быть выражено существительным в единственном числе (возможно, с характеризующими прилагательными).

Примерами атрибутов сущности "Студент" могут быть такие атрибуты как "Номер паспорта", "Фамилия", "Имя", "Отчество", "Номер зачетки", "Дата рождения"

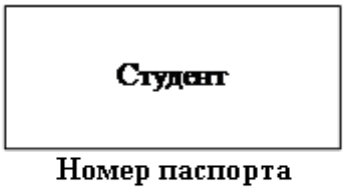



рис. 3 Сущность с указанным ключевым атрибутом

**Ключ сущности** - это неизбыточный набор атрибутов, значения которых в совокупности являются уникальными для каждого экземпляра сущности. Неизбыточность заключается в том, что удалением любого атрибута из ключа нарушается его уникальность.

Сущность может иметь несколько различных ключей.

Расположение ключевых атрибутов на диаграмме показано на рис. 3.

**Связь** - это некоторая ассоциация между двумя сущностями. Одна сущность может быть связана с другой сущностью или сама с собою.

Связи позволяют по одной сущности находить другие сущности, связанные с нею.

Например, связи между сущностями могут выражаться следующими фразами - "ГРУППА может содержать несколько СТУДЕНТОВ".

Графически связь изображается линией, соединяющей две сущности.

Каждая связь может иметь один из следующих типов связи:

Связь типа **один-к-одному** означает, что один экземпляр первой сущности (левой) связан с одним экземпляром второй сущности (правой). Связь один-к-одному чаще всего свидетельствует о том, что на самом деле мы имеем всего одну сущность, неправильно разделенную на две.

Связь типа **один-ко-многим** означает, что один экземпляр первой сущности (левой) связан с несколькими экземплярами второй сущности (правой). Это наиболее часто используемый тип связи. Левая сущность (со стороны "один") называется родительской, правая (со стороны "много") - дочерней.

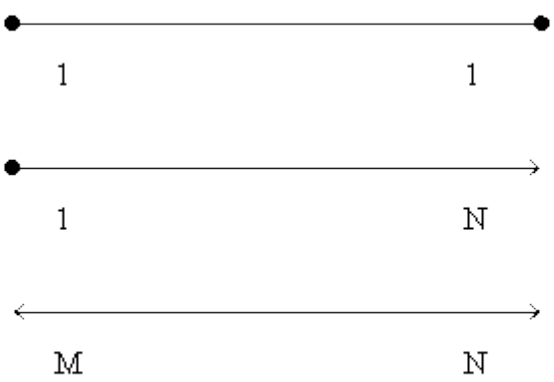



Рис. 4 Типы связей

Связь типа **много-ко-многим** означает, что каждый экземпляр первой сущности может быть связан с несколькими экземплярами второй сущности, и каждый экземпляр второй сущности может быть связан с несколькими экземплярами первой сущности. Тип связи много-ко-многим является временным типом связи, допустимым на ранних этапах разработки модели. В дальнейшем этот тип связи должен быть заменен двумя связями типа

один-ко-многим путем создания промежуточной сущности.

На рис.4 приведено, как должны отображаться на диаграммах все три типа связей.

Каждая связь, по отношению к сущности, может иметь один из двух классов принадлежности.

Класс принадлежности необязательный (Н) означает, что экземпляр одной сущности может быть связан с одним или несколькими экземплярами другой сущности, а может быть, и не связан ни с одним экземпляром.

Класс принадлежности обязательный (О) означает, что экземпляр одной сущности обязан быть связан не менее чем с одним экземпляром другой сущности.

Связь может иметь разный класс принадлежности с разных концов.

Проектирование концептуальной модели состоит в построении реляционных отношений и их первичных ключей на основе анализа ER – диаграмм и применении следующих правил, учитывающих степень связи и класс принадлежности объектов:



**ПРАВИЛО 1**. Если степень связи 1:1 и классы принадлежности О-О, то требуется только одно отношение, ключом которого может быть ключ любого объекта.

**ПРАВИЛО 2**. Если степень связи 1:1 и классы принадлежности О-Н, то необходимо построение двух отношений. Каждому объекту соответствует одно отношение, при этом ключ объекта является ключом соответствующего отношения. Кроме того, ключ второго объекта добавляется в качестве атрибута в первое отношение.

**ПРАВИЛО 3**. Если степень связи 1:1 и классы принадлежности Н-Н, то необходимо построение трех отношений: по одному на каждый объект, причем ключи отношений совпадают с ключами объектов, и связующего отношения, ключом которого будет комбинация ключей объектов.

**ПРАВИЛО 4**. Если степень связи 1:N и классы принадлежности *-О, то достаточным является использование двух отношений, соответствующих объектам, причем ключами будут ключи объектов, но дополнительно ключ первого объекта должен быть атрибутом второго отношения.

**ПРАВИЛО 5**. Если степень связи 1:N и классы принадлежности *-Н, то необходимо построение трех отношений: по одному на каждый объект, причем ключи отношений совпадают с ключами объектов, и связующего отношения, ключом которого будет комбинация ключей объектов.

**ПРАВИЛО 6**. Если степень связи M:N, то необходимо построение трех отношений: по одному на каждый объект, причем ключи отношений совпадают с ключами объектов, и связующего отношения, ключом которого будет комбинация ключей объектов.

После этого все неключевые атрибуты распределяются по отношениям.

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

· первая нормальная форма (1НФ[1NF]);

· вторая нормальная форма (2НФ[2NF]);

· третья нормальная форма (3НФ[3NF]);

· нормальная форма Бойса-Кодда (НФБК[BCNF]);

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

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

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

**Транзитивная функциональная зависимость.** Функциональная зависимость R. X -> R. Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R. X -> R. Z и R. Z -> R. Y и отсутствует функциональная зависимость R. Z --> R. X. (При отсутствии последнего требования мы имели бы "неинтересные" транзитивные зависимости в любом отношении, обладающем несколькими ключами.)

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

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

Отношение находится в первой нормальной форме (1НФ) тогда и только тогда, когда ни один из ее кортежей не содержит в любом своем поле более одного значения и ни одно из ее ключевых полей не пусто.

Отношение находится во второй нормальной форме (2НФ), если она удовлетворяет определению 1НФ и все ее поля, не входящие в первичный ключ, связаны полной функциональной зависимостью с первичным ключом.

Отношение находится в третьей нормальной форме (3НФ), если она удовлетворяет определению 2НФ и не одно из ее неключевых полей не зависит функционально от любого другого неключевого поля.

Отношение находится в нормальной форме Бойса-Кодда (НФБК), если и только если любая функциональная зависимость между его полями сводится к полной функциональной зависимости от возможного ключа.

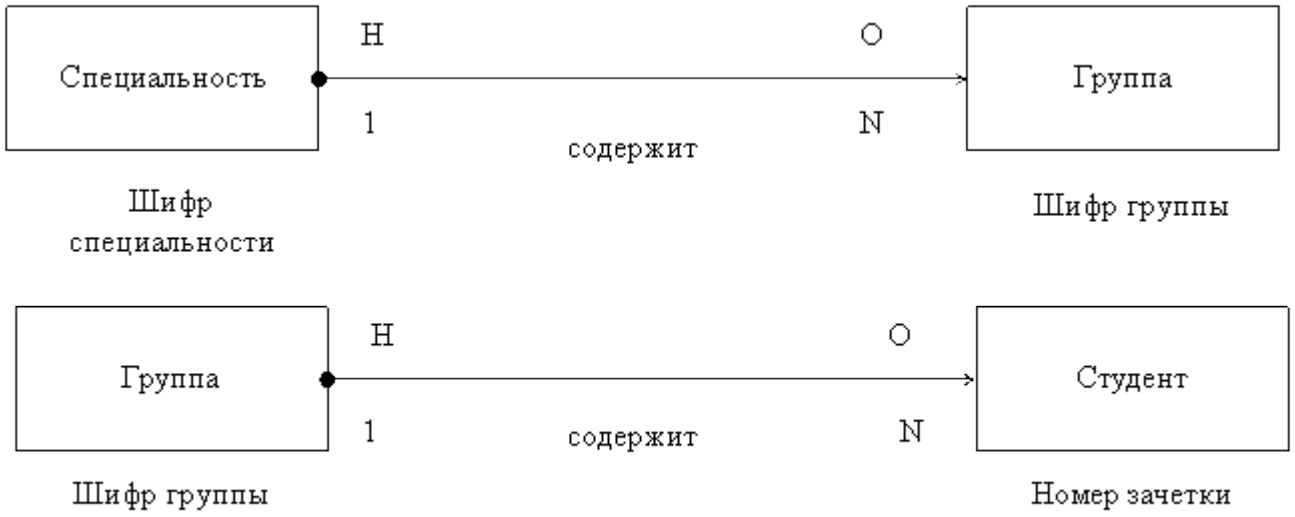
Пример выполнения работы

Для предметной области, описанной в задании, необходимо определить, какие сущности можно из нее выделить. Обычно это существительные из описания предметной области.

· Специальность

· Группа

· Студент




Затем определить пары сущностей, между которыми возможно установить связь.

Далее согласно правилам преобразования ER-диаграмм в отношения записываем набор получившихся отношений. Мы получаем по два отношения от каждой ER-диаграммы.

1) по 4-му правилу:

· Специальность (Шифр специальности)

· Группа (Шифр группы, Шифр специальности)

2) по 4-му правилу:

· Группа (Шифр группы)

· Студент (Номер зачетки, Шифр группы)

Ключевые поля подчеркиваются.

Дублирующие отношения исключаются, оставляем те, в которых получилось больше атрибутов.

Исключая дублирующие отношения и добавляя неключевые атрибуты, получаем:

· Специальность (Шифр специальности, наименование специальности)

· Группа (Шифр группы, Год поступления, Номер, Шифр специальности)

· Студент (Номер зачетки, Шифр группы, Ф. И.О., домашний адрес, дата рождения)

Получить полный текст

- Задать вопрос

Приведем пример проектирования инфологической модели методом функциональных зависимостей.

Пусть необходимо хранить информацию о врачах, пациентах и визитах пациентов к врачам.

Сведем все необходимые атрибуты в одно универсальное отношение и выделим первичный ключ.

ПОСЕЩЕНИЕ

| |
|---|
| Табельный номер врача<br>Номер медицинской карты<br>Дата посещения<br>Время посещения |
| ФИО пациента<br>Адрес пациента<br>**Дата рождения пациента**<br>Пол пациента<br>ФИО врача<br>Категория врача<br>Цель посещения<br>Статус (первичный, вторичный)<br>Диагноз |

1) 1NF требует атомарности атрибутов и отсутствия повторяющихся групп атрибутов.

Повторяющихся групп в полученном универсальном отношении нет.

Адрес пациента будем считать атомарным.

«ФИО пациента» требует разбиения на три поля: «Фамилия», «Имя», «Отчество», т. к. фамилия и инициалы пациентов могут совпадать.

Будем считать, что ФИО врачей не совпадают в одной больнице.

Получаем:

ПОСЕЩЕНИЕ

| |
|---|
| Табельный номер врача<br>Номер медицинской карты<br>Дата посещения<br>Время посещения |
| **Фамилия пациента**<br>**Имя пациента**<br>**Отчество пациента**<br>Адрес пациента<br>Дата рождения пациента<br>Пол пациента<br>ФИО врача<br>Категория врача<br>Цель посещения<br>Статус<br>Диагноз |

2) 2NF требует неизбыточности первичного ключа и независимости неключевых атрибутов от части составного первичного ключа.

Наш первичный ключ является неизбыточным.

Запишем функциональные зависимости:

{Таб. номер врача, Номер мед. карты, Дата посещения, Время посещения} -> цель посещения

{Таб. номер врача, Номер мед. карты, Дата посещения, Время посещения} -> диагноз

{Таб. номер врача, Номер мед. карты, Дата посещения, Время посещения} -> статус

НО:

Табельный номер врача -> ФИО врача

Табельный номер врача -> Категория врача

Номер медицинской карты -> Фамилия пациента

Номер медицинской карты -> Имя пациента

Номер медицинской карты -> Отчество пациента

Номер медицинской карты -> Адрес пациента

Номер медицинской карты -> Пол пациента

Номер медицинской карты -> Дата рождения пациента

Поэтому выносим атрибуты с одинаковой функциональной зависимостью в отдельные сущности «Врач» и «Пациент»:

ВРАЧ ПОСЕЩЕНИЕ ПАЦИЕНТ

| | | |
|---|---|---|
| Табельный номер | Дата посещения<br>**Время посещения**<br>Табельный номер (FK)<br>Номер мед. карты (FK) | Номер мед. карты |
| ФИО врача<br>Категория врача | Цель посещения<br>Статус<br>Диагноз | Фамилия пациента<br>Имя пациента<br>Отчество пациента<br>Адрес пациента<br>Пол пациента |

3) 3NF требует взаимной независимости не ключевых атрибутов.

Все не ключевые атрибуты полученных отношений независимы друг от друга.

Учитывая возможную повторяемость значений атрибутов «Цель посещения и «Категория врача» можно создать дополнительные сущности, играющие роль справочников. Из соображений разумной достаточности в данном примере этого делать не будем.

Следующий этап выполнения лабораторной работы – это построение ER-диаграммы в пакете ERwin.

Реализация моделирования в ERwin базируется на теории реляционных баз данных и на методологии IDEF1X. Методология IDEF1X была разработана для ВВС США и теперь используется, в частности, в правительственных, аэрокосмических и финансовых учреждениях, а также в большом числе частных компаний.

Методология IDEF1X определяет стандарты терминологии, используемой при информационном моделировании, и графического изображения типовых элементов на диаграммах. Возможны две точки зрения на информационную модель и, соответственно, два уровня модели. Первый - логический (точка зрения пользователя) - описывает данные, задействованные в бизнесе предприятия. Второй - физический - определяет представление информации в БД. ERwin объединяет их в единую диаграмму, имеющую несколько уровней представления.

Процесс построения информационной модели состоит из следующих шагов:

· определение сущностей;

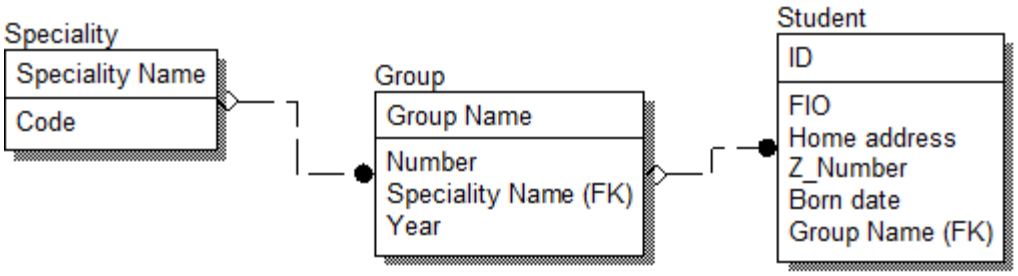
· определение зависимостей между сущностями;

· задание первичных и альтернативных ключей;

· определение атрибутов сущностей;

ERwin создает визуальное представление (модель данных) для решаемой задачи. Это представление может использоваться для детального анализа, уточнения и распространения как части документации, необходимой в цикле разработки. Однако ERwin далеко не только инструмент для рисования. ERwin автоматически создает базу данных (таблицы, индексы, хранимые процедуры, триггеры для обеспечения ссылочной целостности и другие объекты, необходимые для управления данными).

Ниже приведен пример диаграммы (логическая модель), построенной в ERWin.




В верхней части прямоугольников, изображающих сущности, находятся ключевые атрибуты, в нижних – неключевые. FK – внешний ключ. Для атрибутов следует установить тип данных.

Между сущностями устанавливается неидентифицирующая связь.

# Лабораторная работа № 20

## Тема: Создание и заполнение таблиц учебной БД

1. Для создания новой базы:

■ загрузите Access, в появившемся окне выберите пункт **Новая база данных;**

■ в окне «Файл новой базы данных» задайте имя вашей базы (по умолчанию Accessпредлагает вам имя базыdbl, смените его на «Автоцентр» ) и выберите папку**,** где ваша база данных будет находиться (рекомендуется предварительно создать личную папку)

■ щелкните по кнопке <Создать>.

2. Создадим таблицу «***Консультанты***»:

■ в окне базы данных выберите вкладку *Таблицы,* а затем щелкните по кнопке <Создать> (рис. 9);

■ в окне «Новая таблица» выберите пункт **Конструктор** и щелкните по кнопке <ОК>

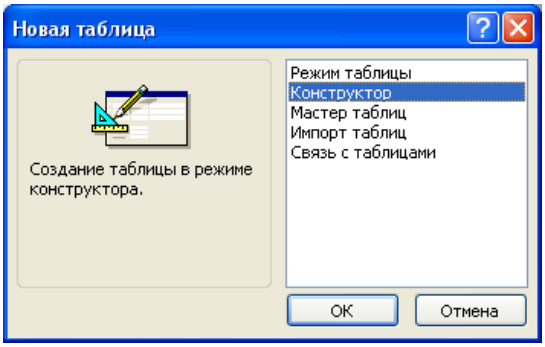



Рис.9 Создание новой таблицы

В результате проделанных операций открывается окно таблицы в режиме конструктора, в котором следует определить поля таблицы.

■ введите в строку столбца «Имя поля» имя первого поля «Табельный номер консультанта»*;*

■ в строке столбца «Тип данных» щелкните по кнопке списка и выберите тип данных *Числовой.*

■ в столбце «Описание» можно оставить пояснительную информацию разного рода, в данном случае пояснений пока не требуется;

■ в нижней части окна расположены две закладки «Общие» и «Подстановка» для указания свойств полей. В свойстве «Размер поля» выберите тип «Целое»

(Для справки: «целый тип» допускает хранение целых чисел из диапазона -32768..32767, тип «байт» используют, если в поле будут заноситься целые значения из диапазона 0..255, «длинное целое» допускает диапазон -2147483648..2147483647, если в поле будут

храниться вещественные дробные значения, следует выбрать «одинарный с плавающей точкой», «двойной с плавающей точкой» или «действительный» формат )

■ поле «Табельный номер консультанта» является ключевым для таблицы «Консультанты» (т.е. однозначно идентифицирует каждую запись), поэтому нажмите на кнопку с изображением ключа на панели инструментов 

■ Переходим к определению свойств следующего поля таблицы, полю «ФИО»

■ задаем имя поля, тип данных- текстовый, столбец «описание» оставим пустым, размер поля- сменим стандартное «50» на «20», уменьшив таким образом максимально допустимое количество символов;

■ Следующее поле, «Стаж работы». Тип данных- текстовый, размер поля- 13, в столбце «Описание» напишите: «поле использует подстановку из фиксированных значений». Пусть при заполнении этого поля нам предлагается список выбора из трех значений: 1 год, 2 года, более 3-х лет, для этого: в столбце «тип данных» выбираем «Мастер подстановок», отмечаем галочкой пункт «будет введен фиксированный набор значений ».

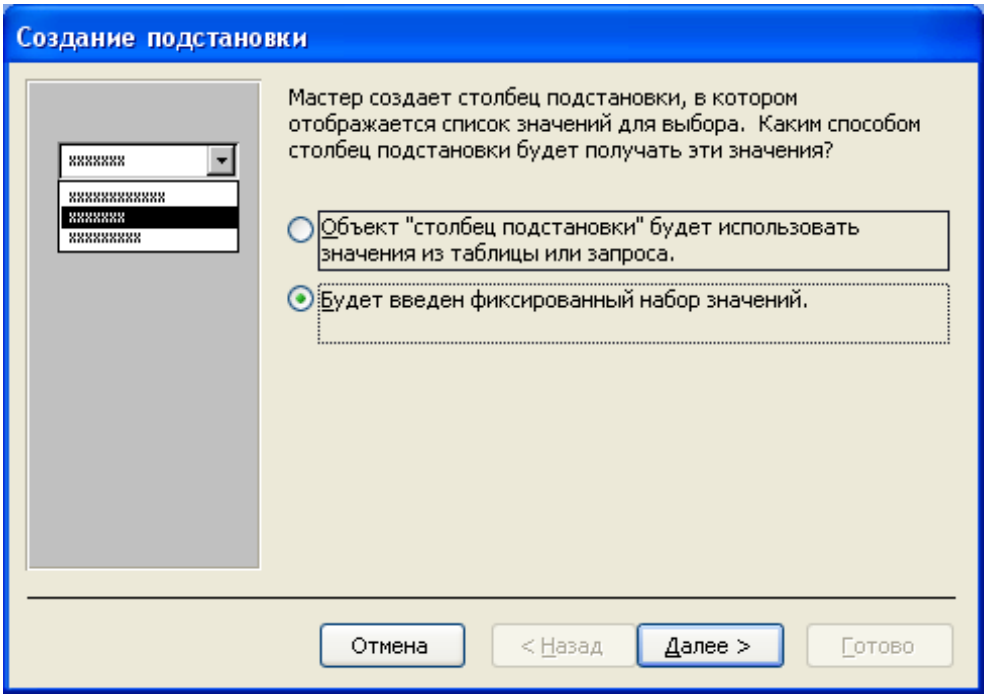



Рис. 10 Создание подстановки

«Далее», в пункте «Число столбцов» указываем «1», и по вертикали перечисляем все возможные значения списка выбора (рис. 11):

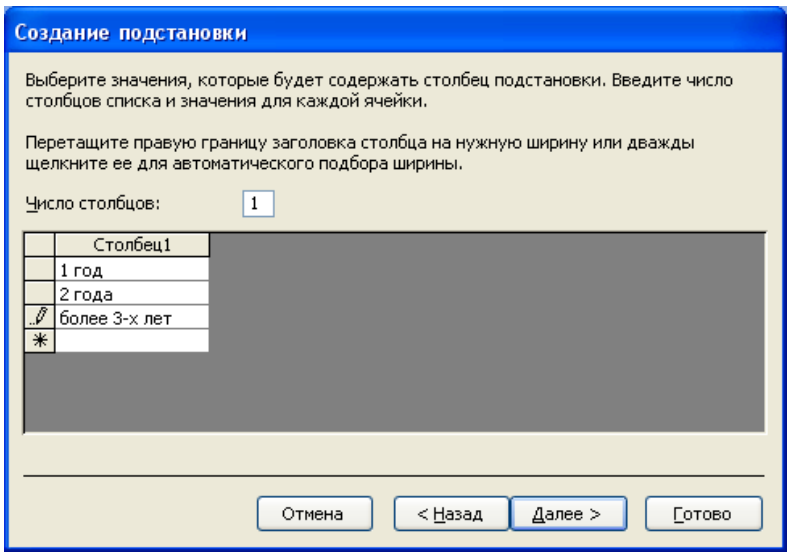



Рис.11 Создание фиксированной подстановки

«Далее», «Готово». Обратите внимание, тип столбца остался по прежнему текстовым (подставляются ведь текстовые значения), о том, что поле использует подстановку говорит лишь наша пояснительная запись (ну и это отражено на вкладке «Подстановка», просмотрите ее содержимое)

■ Переходим к следующему полю «Адрес». Тип- текстовый, в пояснениях не нуждается, размер поля- 30.

■ Следующее поле- «Телефон». Тип- текстовый (в данном случае использовать числовой тип нерационально- мы ведь не будем использовать номер телефона в математических расчетах ) Размер поля- 14

■ Поле «Дата рождения». Тип- дата/время. Формат поля- краткий формат даты (выберите из списка). Зададим так же маску ввода: щелкните на кнопку  в строке «маска ввода», выберите шаблон, соответствующий краткому формату даты (если Access попросит сначала сохранить таблицу- сохраните под именем «*Консультанты*»)

Установим ограничения на возраст сотрудников- они должны быть старше 16 лет. Поэтому в строке «Условие на значение» запишем: < #01.01.1990#. В противном случае на экран должно быть выведено сообщение: «Проверьте дату рождения» (записываем эту фразу в строку «Сообщение об ошибке»)

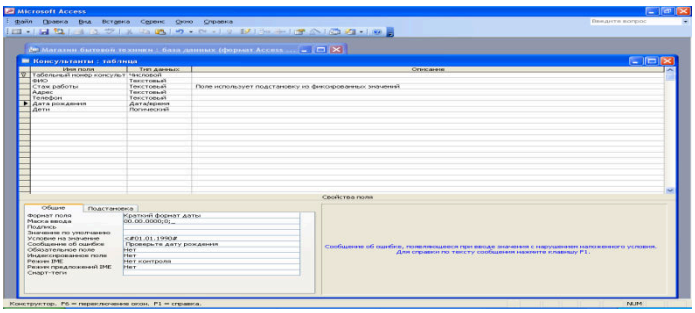


Рис.12 Задание свойств полей

■ И последнее поле- «Дети». Тип- логический.

В итоге должно получиться:

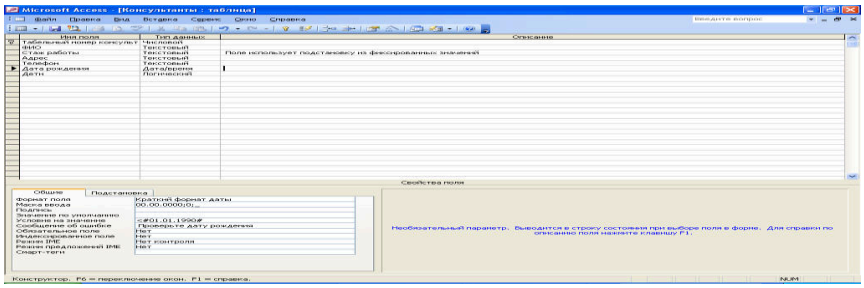


Рис.13 Структура таблицы «Консультанты»

Для сохранения таблицы:

■ выберите пункт меню Файл, Сохранить (или просто попытайтесь закрыть окно конструктора)

■ в диалоговом окне «Сохранение» введите имя таблицы *Консультанты*(если это не было проделано ранее)*;*

■ щелкните по кнопке <ОК>.

Заполним эту таблицу данными.

■ Дважды щелкните по названию таблицы (или по пункту меню «Открыть»)

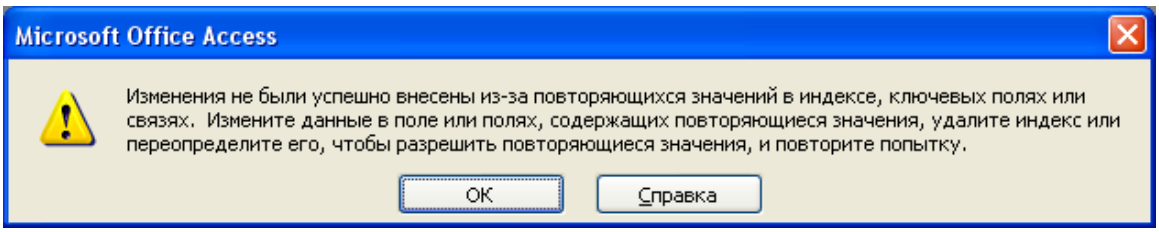
■ Заполните таблицу данными:

Таблица 5. Консультанты

| Таб. номер | ФИО | Стаж работы | Адрес | Телефон | Дата рождения | Дети |
|---|---|---|---|---|---|---|
| 1 | Иванов М.П. | более 3-х лет | Декабристов 5,21 | 267-13-27 | 12.12.1975 | Да |
| 2 | Петров И.О. | 1 год | Парковый 12, 3 | 222-56-17 | 11.10.1980 | Нет |
| 3 | Сидоров Н.К. | более 3-х лет | Сибирская 2, 97 | 265-76-99 | 03.09.1970 | Да |

Посмотрите что будет, если в качестве даты рождения указать, например, 12.12.2000

**Внимание!**Если при заполнении этой и других таблиц вы видите такое сообщение:




То:

1. Вспомните, какое поле в таблице у вас объявлено ключевым;

2. Посмотрите, не пытаетесь ли вы занести в этот столбец одинаковые значения (значения ключевого столбца должны быть уникальны)
3. Если пытаетесь- в пределах текущей (!) строки исправьте ключевое значение на нужное (или как минимум- на отличное от остальных)

■ Для удаления лишних строк (если таковые вдруг появились) или вставки новой строки:

- Щелкаем правой клавишей мыши по полю слева от удаляемой строки. Строка должна выделиться, появляется контекстное меню. Выбираем пункт «Удалить запись» :

■ Откорректируем ширину столбцов. Для этого нажмите клавишу *Shift*, не отпуская ее щелкните мышкой по названиям всех столбцов (они должны выделиться), зайдите в пункт меню «Формат»/ «Ширина столбца»/ «По ширине данных»

■ Проведите сортировку данных по дате рождения, для этого встаньте мышкой в любую строку поля «Дата рождения», на панели инструментов и щелкните по кнопке  . Сами проведите сортировку по табельному номеру.

■ Отфильтруем данные. Например, оставим сведения только тех сотрудников, стаж работы которых более 3-х лет. Для этого встаем мышкой на выражение «более 3-х лет» в любой их строк, на панели инструментов нажимаем кнопку , для снятия фильтра- кнопка

■ Закройте таблицу, предварительно ее сохранив.

3. Создадим таблицу «***Покупатели***».

■ (аналогично через пункты «Создать»/ «Конструктор») В нее войдут поля «ФИО покупателя» - текстовое ключевое поле, размер- 20; «Адрес»- текстовое поле, размер 30; «Телефон»- текстовое, размер 14. Пояснения и какие-либо дополнительные параметры задавать не будем. Сохраните структуру таблицы, задав ей имя «*Покупатели*»:

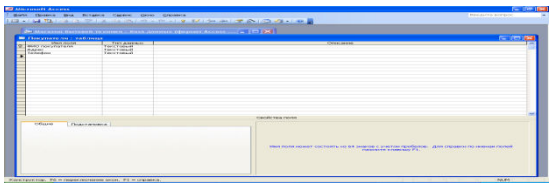


Рис.14 Структура таблицы «Покупатели»

Заполнять эту таблицу данными пока не будем.

4. Создадим таблицу «***Поставщики***»:

■ в нее войдут поля: «Номер поставщика»- тип числовой, размер поля- целое, это ключевое поле; «Название поставщика»- тип текстовый, размер- 20, «Адрес»- текстовый, размер-30, «Телефон»- текстовый, размер 14. Сохраните таблицу.

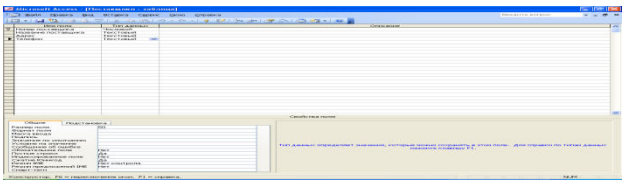


Рис. 15 Структура таблицы «Поставщики»

■ Откроем таблицу для заполнения -двойным щелчком по названию или через пункт «Вид»/ «Режим таблицы» (если вы находитесь в режиме конструктора)

■ Заносим в нее информацию о двух поставщиках:

Таблица 6. Поставщики

| **Поставщики** | | | |
|---|---|---|---|
| **Номер поставщика** | **Название поставщика** | **Адрес** | **Телефон** |
| 1 | Автоленд и Ко | Москва, Садовая 13 | 8-937-56-7777 |
| 2 | Renault в России | Москва, Кутузова 1 | 8-937-45-3333 |

Сохраняем, выходим из таблицы.

5. Создаем таблицу «***Автомобили***»:

■ Поле «Марка автомобиля»- текстовое, размер- 15 символов, ключевое поле.

■ Поле «Технические характеристики». Тип- поле Memo. В столбце «Описание» напишем: документация по авто, максимум на 65535 знаков.

■ Поле «Изображение». Тип- поле объекта Ole. В него будем вставлять картинки.

Все, сохраняем и закрываем конструктор таблицы «Автомобили»:

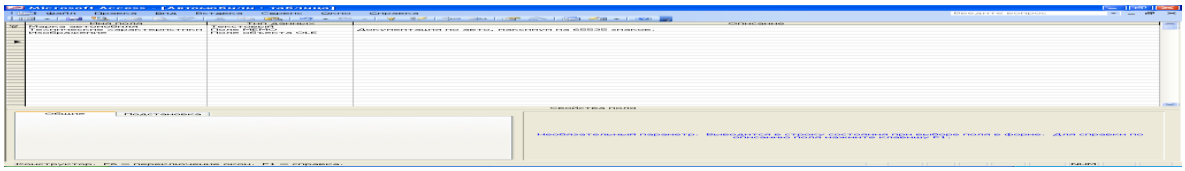


Рис.16 Структура таблицы «Автомобили»

■ Открываем таблицу для заполнения (двойным щелчком, или кнопкой «Открыть»)

■ Заполняем столбцы «Марка автомобиля» (Clio, Kangoo, Laguna,Logan,Megane,Modus,Symbol), «Технические характеристики» (здесь укажем только ссылки на официальный сайт Рено);

■ Столбец «Изображение». Щелкаем правой клавишей мыши в первой строке поля «Изображение», выбираем пункт «Добавить объект»:

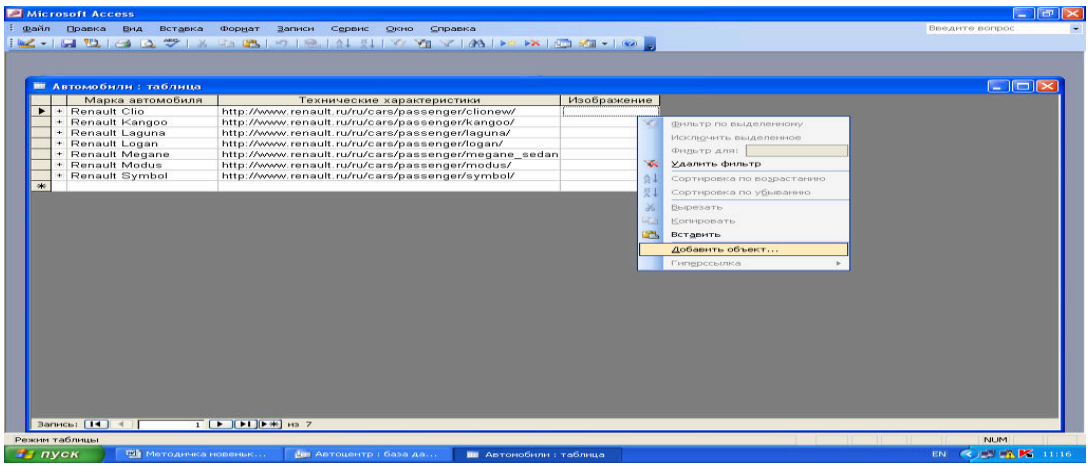



Рис.17 Добавление *Ole*- объектов

Тип объекта- рисунок MSWord. ОК.

■ Откроется лист Word. Находясь курсором в рамке, выберите в меню пункт Вставка/ Рисунок/ Из файла. Спросите у преподавателя, в какой папке находятся изображения машин, откройте эту папку, выберите нужный рисунок (Clio), «Вставить».

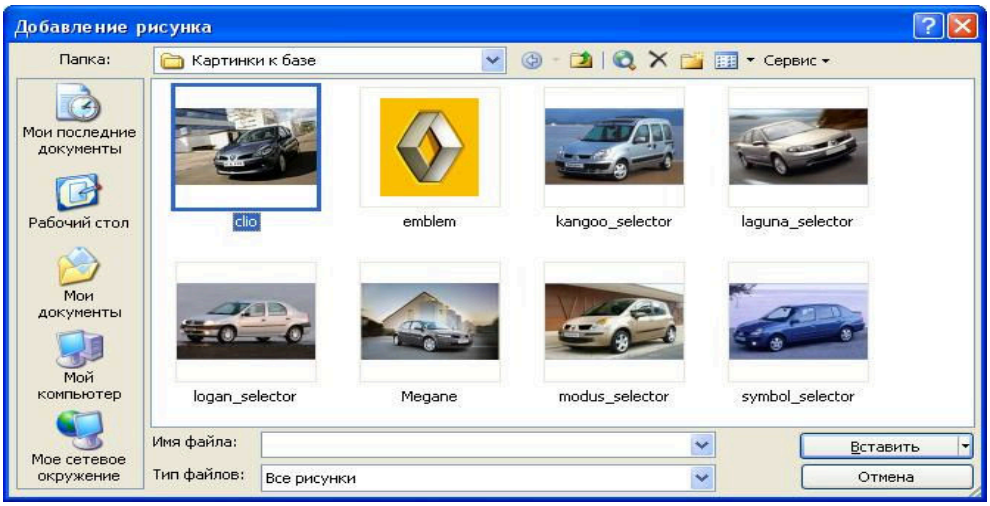



Рис.18 Изображения моделей

Подгоните изображение за узлы сетки под размер рамки, затем щелкните «Закрыть рисунок».

■ Аналогично вставляем картинку для следующей машины. Повторяем для всех семи марок.

■ Откорректируйте ширину столбцов;

■ Закрываем таблицу «*Автомобили*», сохраняя все изменения.

6. Создадим таблицу «***Поставки***»:

■ Поле «Номер поставки», тип- числовой, размер- целый, это ключевое поле таблицы.

■ Поле «Марка автомобиля», тип- текстовый, размер- 20. Виды автомобилей будем выбирать из списка (закупаем только те марки, которые упомянуты в таблице «Автомобили») Таблица «Автомобили» может изменяться, столбец подстановки будет так же динамично ему соответствовать. Создадим подстановку: тип «текстовый» меняем на «мастер подстановок», выбираем «объект будет использовать значения из таблицы или запроса», далее выбираем таблицу «Автомобили», далее, в столбец «выбранные поля» переносим поле «марка автомобиля» (кнопкой  ):

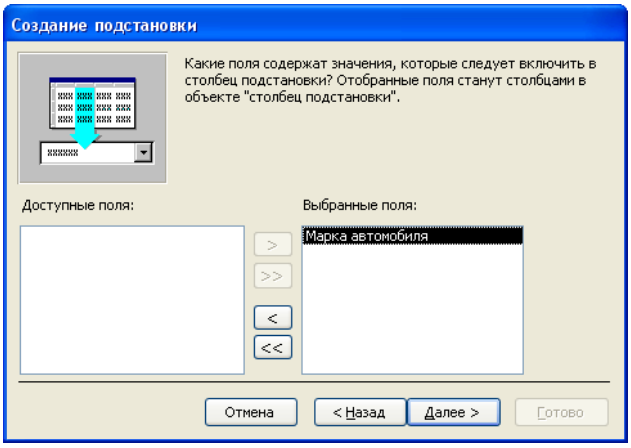



Рис.18 Создание подстановки из таблицы

«Далее», «Далее» (можно выбрать сортировку по марке), «Далее», «Готово».

В столбце «Описание» упомянем про используемую подстановку.

■ Поле «Количество»- числовое, целое.

■ Поле «Закупочная цена»- тип денежный, формат Евро.

■ Поле «Дата поставки»- тип дата/время, формат – краткий формат даты, установите маску ввода; в строке «Значение по умолчанию» напишите *Date()*– по умолчанию будет ставиться текущая дата:

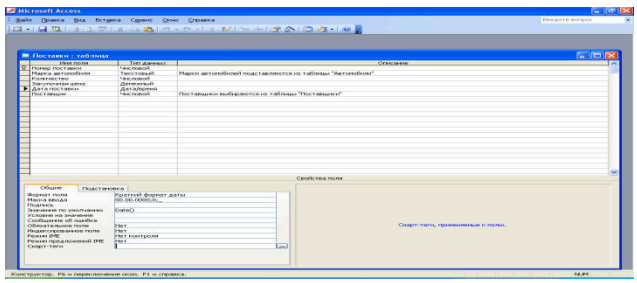


Рис. 19 Свойства поля «Дата поставки»

■ Поле «Номер поставщика»- числовое, целое. Создадим для него подстановку: тип меняем на «мастер подстановок», выбираем «объект будет использовать значения из таблицы или запроса», далее выбираем таблицу «Поставщики», далее, в столбец «выбранные поля» переносим оба доступных поля (номер поставщика и название поставщика), далее, далее, убираем галочку в строке «Скрыть ключевой столбец»

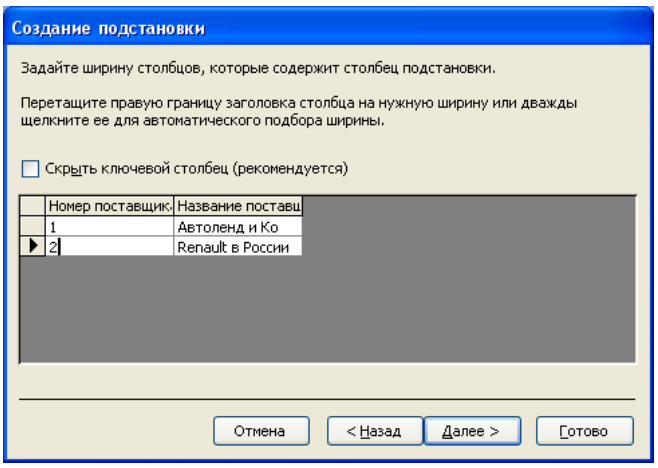



Рис.20 Создание подстановки для поля «Номер поставщика»

Далее, в окне «Доступные поля» выбрать «Номер поставщика», далее, готово. При появлении новых поставщиков их названия автоматически появятся в списке выбора.

■ Закройте таблицу, назвав ее «*Поставки*» (если не сделали этого ранее)

Пока не будем заполнять эту таблицу данными.

7. Создадим таблицу «***Продажи***» с полями:

■ «Номер продажи»- тип числовой, размер целый, это ключевое поле.

■ «Марка автомобиля»- тип текстовый, затем меняем его на мастер подстановок, выбираем «объект будет использовать значения из таблицы или запроса», далее выбираем таблицу «Автомобили», далее, в столбец «выбранные поля» переносим поле «Марка автомобиля» «Далее», «Далее» , «Далее», «Готово». Делаем соответствующую запись в столбце «Описание»

■ Поле «Цвет»- текстовое, размер- 15, зададим список из фиксированных значений: «мастер подстановок»/ «будет введен фиксированный набор значений »/ «Далее», в пункте «Число столбцов» указываем «1», и по вертикали перечисляем все возможные значения списка выбора: черный, серебристый, белый, красный, песочный, синий, бордо и т.д. (укажите 5-7 вариантов любых цветов), «Далее», «Готово».

■ Поле «Цена продажи» - тип денежный, формат Евро.

■ Поле «ФИО покупателя»- текстовое, размер- 20, создаем подстановку из таблицы «Покупатели»: мастер подстановок, выбираем «объект будет использовать значения из таблицы или запроса», далее выбираем таблицу «Покупатели», далее, в столбец «выбранные поля» переносим поле «ФИО покупателя» «Далее», «Далее» , «Далее», «Готово». Делаем соответствующую запись в столбце «Описание».

■ Поле «Номер консультанта». Числовое, целое. Создаем подстановку из таблицы «Консультанты»: мастер подстановок, выбираем «объект будет использовать значения из таблицы или запроса», далее выбираем таблицу «Консультанты», далее, в столбец «выбранные поля» переносим поля «Табельный номер консультанта» и «ФИО», «Далее», «Далее», убираем галочку у пункта «Скрыть ключевой столбец», «Далее», доступные

поля- «таб. номер консультанта», «Далее», «Готово». Делаем соответствующую запись в столбце «Описание».

■ Дату продажи разобьем на три отдельных поля: «Число» (числовой тип), «Месяц»- текстовый тип и «Год»- числовой тип. Создайте для каждого из этих полей подстановки из фиксированных значений (как мы это сделали для поля «Цвет»):

Подстановочный столбец для поля «Число»: 1 2 3 4 и т.д. до 31; для «Месяц»: января, февраля, марта и т.д. для поля «Год»: 2006, 2007, 2008 (Значения эти заносите по вертикали!)

■ Сохраняем таблицу «*Продажи*»:

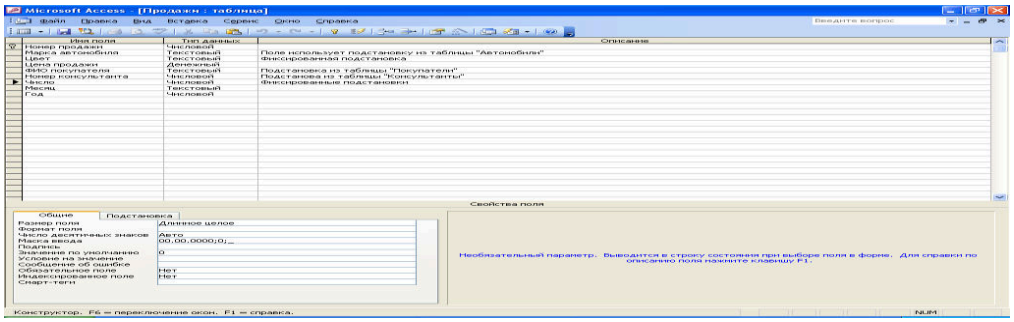


Рис.21 Структура таблицы «Продажи»

■ Выходим из конструктора таблицы. Заполнять ее данными пока не будем.

В результате проделанных операций должны получить шесть таблиц: *Автомобили, Консультанты, Покупатели, Поставки, Поставщики, Продажи*.

## Лабораторная работа № 21

## Тема: Организация связи таблиц

Вызываем окно схемы данных:

На панели инструментов щелкаем по кнопке  .

■ Из за ряда используемых подстановок таблицы уже связаны между собой. Разместите таблицы на схеме, как вам удобно (перетаскивая их за заголовки), проследите, что бы были видны все их поля (если не видны- двойной стрелкой «удлините» таблицы снизу). ■ Для добавления таблиц на схему используйте кнопку  с панели инструментов. У нас на схеме присутствуют все 6 таблиц базы, добавлять их нет необходимости. Добавим в качестве тренировки таблицу «Автомобили» (она появится на схеме с именем «Автомобили_1»), тут же удалим ее (встав на нее мышкой, и нажав Delete)

2. Изменим типы связей между таблицами. Для этого щелкаем правой клавишей мыши на схеме данных **по линии связи**, появится небольшое окно с двумя пунктами : «Изменить связь» и «Удалить»:

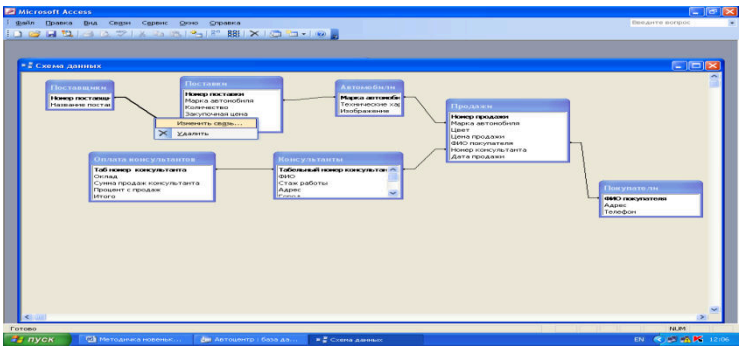
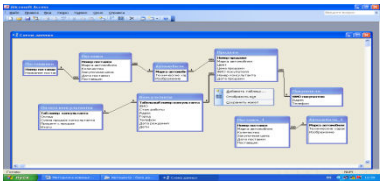


Рис.22 Изменение типа связи

если же вы видите следующее:



то, вероятно, вы не попали мышкой по линии связи. Щелкните еще раз правой клавишей по связи.

■ Выделяем пункт «Изменить связь».

■ Ставим галочки у пунктов «Обеспечение целостности данных», «Каскадное обновление связанных полей», «Каскадное удаление связанных полей». ОК. (см. рис 23)

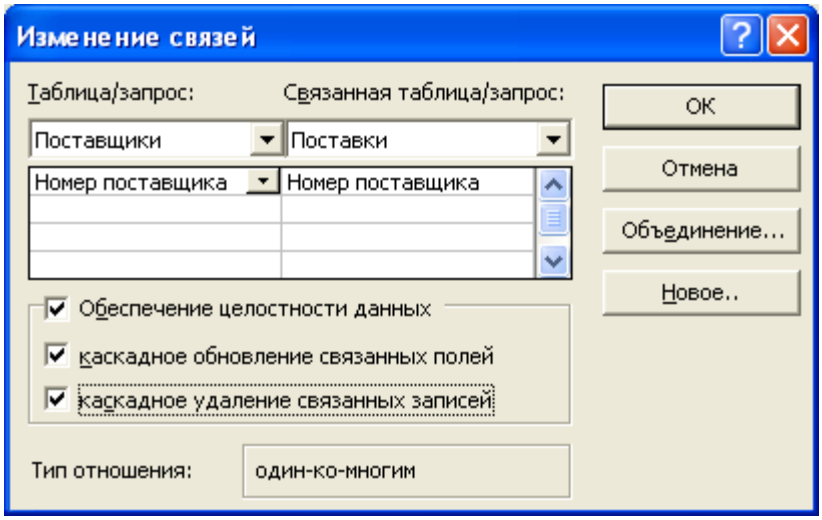



Рис. 23 Изменение типа связи

Эти меры уменьшат нам время на ввод и корректировку данных, помогут «на лету» выявлять противоречивости и прочие возможные ошибки в данных.

3. Проделываем аналогичные операции для всех связей схемы, в результате должны получить:

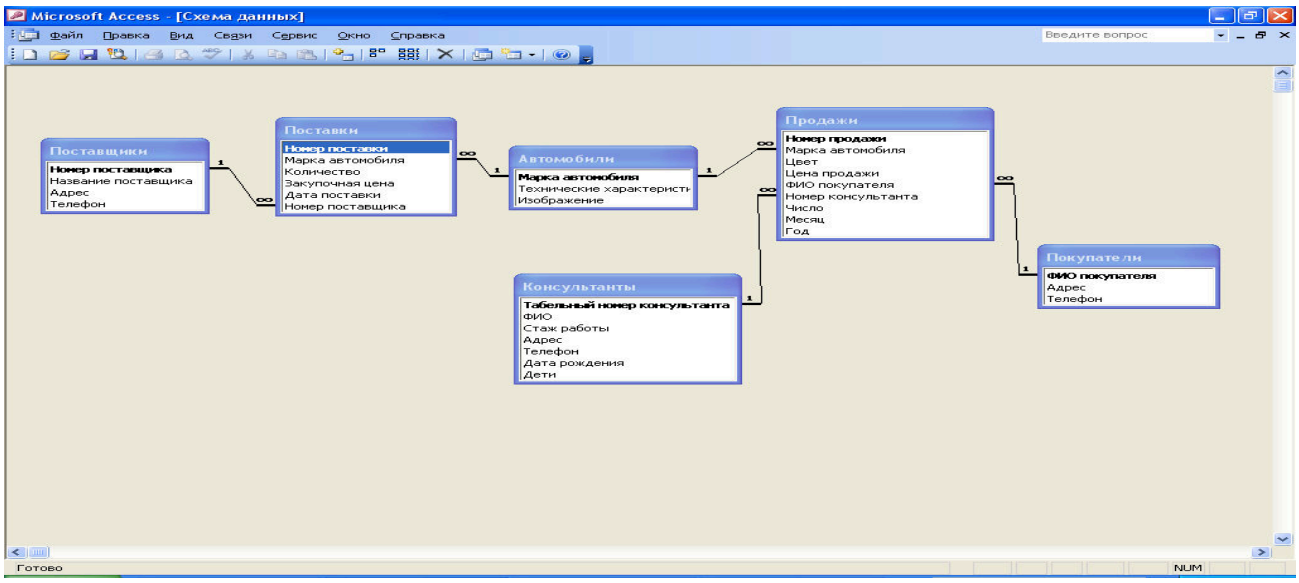



Рис.24 Схема данных

■ Обратите внимание- при связывании ключевого поля с неключевым Access автоматически создает связь типа «один-ко-многим». Это полностью соответствует инфологической модели данных, построенной в работе 1.

**Важно!** Если ваша схема выглядит иным образом (не считая расположения таблиц- оно произвольно) или возникают ошибки при установке связей- попробуйте разобраться в причине сами. (Возможные ошибки: не верно указаны ключевые поля в таблицах, связь устанавливаете по полям разных типов или разных форматов (размеров) и пр. -читайте тексты ошибок!) Проверьте эти моменты в конструкторах таблиц (быстро попасть туда можно щелкнув правой к.м. по заголовку таблицы и выбрав пункт «Конструктор таблицы»). Не помогает? Зовите преподавателя, двигаться дальше не имеет смысла.

**4.** Как установить связи между таблицами, если при их создании не использовались никакие взаимные подстановки, т.е. при открытии схемы данных связей между ними нет?

- Нажмите на кнопку на панели инструментов в окне схемы, добавьте таблицы «Автомобили» и «Поставки». Они появятся с именами «Автомобили_1» и «Поставки_1» и никак между собой не связаны. Левой клавишей мыши щелкните на поле «Марка автомобиля» в таблице «Поставки_1». Не отпуская (!) левую клавишу мыши протаскиваем курсор к полю «Марка автомобиля» в таблице «Автомобили_1». (Курсор при попадании в область таблиц приобретает вид небольшого прямоугольника) Отпускаем л.к.м., открывается окно «Изменение связи». Проставляем три галочки у пунктов «Обеспечение целостности данных», «Каскадное обновление связанных полей», «Каскадное удаление связанных полей». «Создать»:

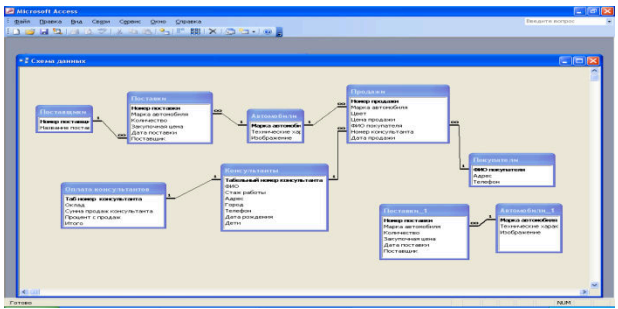


Рис.25 Установка связей между таблицами

■ Получилось? удаляем сначала линию связи (п.к.м. по линии связи / удалить), затем обе эти таблицы (Delete).

■ Закрываем схему данных, сохранив изменения.

Итак, в мы создали шесть таблиц, определили перечень полей в них, задали для каждого поля тип, указали свойства. На схеме данных задали связи между таблицами. В качестве примера даже заполнили 3 таблицы начальными данными (*Автомобили, Поставщики* и *Консультанты*). Дальше вкладкой «Таблицы» пользоваться практически не будем. Для занесения данных в базу, их просмотра, изменения и прочих операций гораздо удобнее пользоваться формами.

# Лабораторная работа № 22

# Вывод результата в компонент DBGRID

Цель работы:

1. Изучить начальные этапы создания приложения для работы с базами данных в среде Delphi:

- ознакомить с компонентами доступа к БД: **TTable, TDataSource;**
- ознакомить с компонентами управления БД: **TDBGrid, TDBNavigator.**

2. Усвоить ввод и редактирование текста.

Методика создания приложения для работы с базой данных ничем не отличается от методики создания обычной программы: к форме добавляются необходимые компоненты, устанавливаются значения свойств компонентов, разрабатываются необходимые процедуры обработки событий.

Приложение работы с базой данных должно содержать компоненты, обеспечивающие доступ к данным, возможность просмотра и редактирования содержимого полей. Компоненты доступа к данным находятся на вкладке **Data Access** и **BDE** палитры компонентов, а компоненты отображения данных — на вкладке **Data Controls.**

Основные компоненты доступа и управления к базам данных:

Компоненты доступа к базам данных:

**TTable** - обеспечивает взаимодействие с таблицей БД, т.е. компонент **TTable** указывает, откуда брать данные и какие поля будут составлять набор данных. Компонент **TTable** имеет следующие основные свойства:

- **DatabaseName** – база данных
- **TableName** – имя таблицы
- **Active** – активация таблицы (значение **True** активирует ее)

Свойство **DatabaseName** определяет базу данных, в которой находится таблица. Это свойство может содержать:

1. псевдоним (псевдоним)
2. путь для локальных БД

3. путь и имя файла базы данных для **Local InterBase**
4. локальный псевдоним, определенный через компонент **`TDatabase`**.

Свойство **`TableName`** определяет имя таблицы базы данных.

**TDataSource**  - определяет связь между базой данных и компонентами управления данными, то есть компонент **TDataSource** является промежуточным звеном между компонентом **Table1**, соединенным с реальной БД и визуальными компонентами **DBGrid1** и **DBNavigator1**, с помощью которых пользователь взаимодействует с таблицей.

В большинстве случаев, все, что нужно сделать с **DataSource** - это указать в свойстве **DataSet** соответствующий **TTable**. Затем, у визуального компонента вроде **DBGrid** или **DBNavigator** в свойстве **DataSource** указывается **TDataSource**, который используется в настоящее время.

Компоненты управления данными с палитры **Data Contorls**:

**TDBGrid**  - отображает содержимое таблицы БД в виде сетки, в котором столбцы соответствуют полям, а строки записям таблицы.

Компонент имеет следующие свойства:

- **Data Source** – содержит ссылку на компонент типа **TDataSource**, служащий источником данных;
- **Editor Mode** – если содержит **true**, пользователь может редактировать ячейку после нажатия клавиши **F2** или **Enter**. Игнорируется, если свойство **Option** включает значение **goEditing** или **goAlwaysShowEditor**;
- **Option** – определяет вид и поведение компонента;

- **dgEditing** – разрешает изменение набора данных;
- **dgAlwaysShowEditor** – автоматически переводит столбец в режим редактирования при его выделение;
- **dgTitles** – показывает заголовки столбцов;
- **dgIndicator** – показывает индикатор текущей строки в самом левом фиксированном столбце;
- **dgColumnResize** – разрешает пользователю вручную изменять ширину столбцов;
- **dgColLines** – показывает разделяющие вертикальные линии;
- **dgRowLines** – показывает разделяющие горизонтальные линии;
- **dgTabs** – разрешает переход от столбца к столбцу с помощью клавиши **Tab**;
- **dgRowSelect** – разрешает выделение цветом всей выбранной строки;
- **dgAlwaysShowSelection** – выделение текущей строки сохраняется, если компонент теряет фокус ввода;
- **dgConfirmDelete** – удаление строки должно подтверждаться;
- **dgCancelOnExit** – если пользователь вставил пустую строку и покинул ее, она не помещается в набор данных;
- **dgMultiSelect** – разрешает множественный выбор строк.

**TDBNavigator**  - осуществляет перемещение и редактирование записей (вид и назначение кнопок указаны в пункте **DBNavigator**). С помощью свойства **DataSource** компонент связывается с нужным источником данных **TDataSource** – это все, что необходимо для его нормальной работы. Свойство **ConfirmDelete** управляет отображением диалогового окна с просьбой подтвердить удаление записи (значение **True** этого свойства выводит окно).

**Создание приложения:**

1. Запустить Delphi.
2. В свойстве **Caption** изменить имя формы на *Студенты*.
3. Установить на форму компоненту **TTable**.
4. Определить следующие свойства компоненты **Table1**.

- определить псевдоним - выбрать в свойстве **DatabaseName** инспектора объектов псевдоним «Student».
- задать имя таблицы - выбрать в свойстве **TableName** таблицу Student.
- активизировать таблицу - установить в свойстве **Active** значение **true** (это будет возможно после выполнения пункта 4).

5. Разместить на форму компоненту **DataSource** с закладки **Data Access** и в свойстве **DataSet** инспектора объектов выбрать компоненту **Table1**.
6. Расположить на форме компоненту **DBGrid** с закладки **Data Controls** и в свойстве **DataSource** выбрать **DataSource1**.

Если вы внесли несколько записей, то форма **Студенты** примет вид:

**Редактор полей:**

Для управления отображением данных таблицы используют специальный редактор полей - **Editor Field**.

Для вызова **Editor Field** следует:

- Дважды щелкнуть по **Table1**.
- Для открывшегося окна вызвать контекстное меню и выбрать пункт **Add All Field**, если необходимо добавить все поля таблицы.
- **Add Field** для выбора отдельного поля.

Редактор полей имеет следующие свойства:

- **DisplayLabel** – задает имя полю*;*
- **DisplayWidth –** определяет количество символов, которое будет выводится в поле;

Определим свойства для полей таблицы *Student.db*.

1. Выбрать в окне редактора полей таблицы поле **SFio** и в свойстве **DisplayLabel** инспектора объектов изменить **SFio** на **ФИО**. Выбрать свойство **DisplayWidth** и заменить размер на 35.

2. Так же поменять свойства других полей таблицы.
3. Для полей логического типа в свойстве **DisplayValues** можно написать варианты для значений **True** и **False**. В поле **SSpec** в этом свойстве написать «**Математика;Физика**» (без пробела, разделяя «;»). Получиться как показано на рисунке.
4. Если возникнет необходимость можно скрыть любое поле, выбрав его и в свойстве **Visible** инспектора объектов установив значение **false**.

После выполненных действий сетка **DBGrid1** будет выглядеть так:

**Ввод данных:**

Компоненты для организации доступа к таблицам БД позволяют выполнять всевозможные операции с наборами данных: добавлять или удалять записи, перемещаться по ним. При этом следует иметь в виду, что в любой момент времени доступна для выполнения конкретных действий только одна запись, называемая *текущей.* В этой лабораторной работе рассматриваются наиболее часто используемые методы компоненты **Table**.

Основные методы для организации доступа компоненты **Table**:

- **Append** – добавить новую запись в конец таблицы.
- **Delete** – удалить текущую строку.
- **Edit** – перейти в режим редактирования. После этого можно изменять значения полей.
- **Insert** – вставить новую строку в таблицу.
- **Post** – принять все изменения.
- **Refresh** – обновить информацию о данных.
- **UpdateRecord** – обновить текущую запись.

1. Откроем созданное приложение.
2. Разместим на форме три компоненты **SpeedButton** из палитры **Additional**. Одна из кнопок будет добавлять запись, другая – изменять данные в записи, третья – удалять. Назовем их соответственно.

3. Создадим новую форму, которая будет вызываться нажатием кнопки *«Добавить»*. На форме расположены 4 компоненты **Edit**, компонент **DateTimePicker** с закладки **Win32**, компонент **CheckBox** и компонент **RadioGroup.**

4. Текст процедуры для события **OnClick** кнопки *«Добавить»* на форме **Студенты**:

```
procedure TForm1.SpeedButton1Click(Sender: TObject);

begin

Form2.ShowModal; //открывает форму «Добавление записи»

end;
```

5. Текст процедуры для события **OnClick** кнопки *«ОК»* на форме **Добавление записи**:

```
procedure TForm2.Button1Click(Sender: TObject);

begin

Form1.Table1.Insert;

Form1.Table1.FieldByName('SFio').Text:=Edit1.Text;

Form1.Table1.FieldByName('SOsn').Text:=Edit2.Text;

Form1.Table1.FieldByName('SNom').Text:=Edit3.Text;

Form1.Table1.FieldByName('SKurs').Text:=Edit4.Text;

Form1.Table1.FieldByName('SData').AsDateTime:=DateTimePicker1.Date;

if CheckBox1.Checked then

Form1.Table1.FieldByName('SStip').Text:='ðà'
```

*else*

*Form1.Table1.FieldByName('SStip').Text:='нет';*

*//при нажатии на флажок полю SStip (Стипендия) передается*

*//значение True, в противном случае вводится передается*

*//значение False*

*case RadioGroup1.ItemIndex of*

*0: Form1.Table1.FieldByName('SSpec').Text:='Математика';*

*1: Form1.Table1.FieldByName('SSpec').Text:='Физика';*

*end;*

*if form1.Table1.Modified*

*then form1.Table1.Post;*

*close;*

***Комметарий:*** в строке *Form1.Table1.Insert* вызывается метод, который допускает вставку новой строки в таблицу, которая находится на форме «**Студенты**». Без вызова этого метода дальнейшая работа по вставке записи в таблицу невозможна. Запись *Form1.Table1.FieldByName('SFio').Text:=Edit1.Text* означает, что текст, который находится в **Edit1** по нажатии кнопки будет перенесен в таблицу на форме «**Студенты**» в новую запись в текстовое поле **ФИО**. Остальные записи в процедуре работают аналогичным образом. Запись *if form1.Table1.Modified then form1.Table1.Post* сохраняет изменения в таблице. *Close* – закрывает форму «**Добавление записи**».

6. По нажатии кнопки **Cancel** осуществляется выход. То же и на форме «Редактирование записи».
7. Текст процедуры для события **OnClick** при нажатии клавиши *«Удалить»* на форме **Студенты**:

*procedure TForm1.SpeedButton3Click(Sender: TObject);*

*begin*

*Table1.Delete //удаляет текущую запись в таблице*

*end;*

**<u>Редактирование данных:</u>**

Компоненты, отражающие информацию, делятся на две категории – те, которые не связаны с таблицами БД, и компоненты, связанные с таблицами и обменивающиеся с ними информацией. В первую категорию входят обычные компоненты Delphi. Компоненты второй категории расположены на странице **Data Controls**. Почти каждая из них имеет аналог среди обычных компонент; основные отличия заключаются в том, что они могут работать с данными, хранящимися в БД. К этой группе относится компонента **DBEdit**, которая используется для ввода текстовой однострочной информации.

Чтобы компонент **DBEdit** видел данные из поля таблицы, следует указать в свойствах:

- DataSource – источник данных;
- DataField – поле для редактирования.

***Этот компонент с заданными уже свойствами может появиться автоматически при перетаскивании имени поля из окна редактора полей***.

1. Создадим новую форму: **Редактирование записи**.

2. В случае перетаскивания поля логического типа, на форме автоматически устанавливается компонента **DBCheckBox**, что не всегда удобно.
    1. Установим на форму компоненту **DBRadioGroup** с закладки **Data Control**.
    2. В свойстве **DataSource** укажем **DataSource1**.
    3. В свойстве **DataField** укажем **SSpec**.
3. Текст процедуры для события **OnClick** при нажатии клавиши *«Изменить»* на форму **Студенты**:

```
begin

Form3.ShowModal //вызов Form3

end;
```

4. Текст процедуры для события **OnClick** при нажатии клавиши *«Сохранить»* на форму **Редактирование**:

```
begin

if form1.Table1.Modified

then form1.Table1.Post; //все изменения в таблице сохраняются

close;

end;
```

5. Пользователь имеет возможность редактировать записи в таблице напрямую. Чтобы это предотвратить используется свойство компоненты **DBGrid** ***dgEditing***. Нужно выделить **DBGrid1** и в свойстве ***Options►*** ***dgEditing*** инспектора объектов поставить **false**.

## Лабораторная работа № 23 .

## Тема: Выполнение сортировки и модификации данных в таблицах.

Цель: изучение информационной технологии редактирования и модификации таблиц в СУБД MS Access.

Задание 2.1. Произвести модификацию таблицы«Сотрудники фирмы».

Порядок выполнения работы:

1.Запустите программу СУБД Microsoft Access и откройте свою базу данных.

2.Создайте копию таблицы «Сотрудники фирмы», назвав ее «Сотрудники фирмы_новый».

3.В копии произведите редактирование данных:

−удалите восьмую запись. Для этого выделите запись нажатием

на кнопку слева от записи и воспользуйтесь командой Удалить запись контекстного меню, вызываемого правой кнопкой мыши. При удалении программа попросит подтверждение на удаление (рис. 2.1). Дайте подтверждение удаления кнопкойДА. Если все сделано правильно, то восьмой записи после этой операции не будет.

Рис. 2.1. Подтверждение удаления записи в таблице БД

−в третьей записи измените фамилию на Арбенин;

−введите новую запись в Режиме таблицы с фамилией

Рокотов;

−переместите первую запись в конец таблицы (выделите первую запись и воспользуйтесь командой Правая кнопка мыши /Вырезать, далее выделите очередную свободную строку записи и воспользуйтесь командойПравая кнопка мыши /Вставить; если вы

выполнили все правильно, то записи с номером 1 после этой операции не будет);

−скопируйте запись с фамилией Рокотов на вторую и измените в

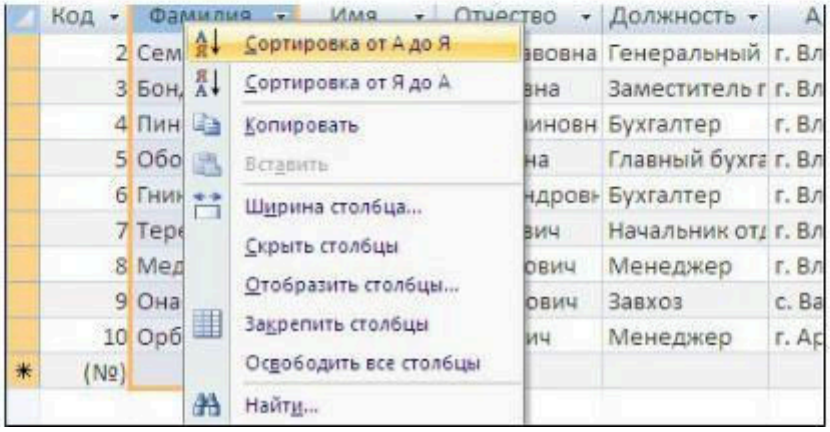
ней имя;

−проверьте правильность изменений БД: должны быть записи с номерами со 2 по 7 и с 9 по 13.

3.Проведите сортировку данных по полю Фамилия в порядке

убывания (выделите соответствующее поле Фамилия нажатием на его название и выберите командуПравая кнопка мыши /Сортировка от Я до

| Код | Фамилия | Имя | Отчество | Должность | А |
|---|---|---|---|---|---|
| 2 | Семидьянова | Евгения | Вячеславовна | Генеральный | г. Вл |
| 3 | Бондаренко | Анна | Сергеевна | Заместитель г | г. Вл |
| 4 | Пинегина | Кристина | Вениаминовн | Бухгалтер | г. Вл |
| 5 | Обова | Анфиса | Павловна | Главный бухга | г. Вл |
| 6 | Гниненко | Ольга | Александровн | Бухгалтер | г. Вл |
| 7 | Терещенко | Вадим | Сергеевич | Начальник отд | г. Вл |
| 8 | Медведь | Данила | Робертович | Менеджер | г. Вл |
| 9 | Онаев | Иван | Педросович | Завхоз | с. Ва |
| 10 | Орбачев | Александр | Иванович | Менеджер | г. Ар |
| * | (№) | | | | |




А) (рис. 2.3).

Аналогично проведите сортировку данных по полю Дата найма в порядке возрастания.

Рис. 2.2. Примерный вид таблицы «Сотрудники фирмы_новый» после редактирования

Рис. 2.3. Сортировка полей таблицы БД

5.Проведите поиск всех записей с фамилией Рокотов, для этого установите курсор или выделите необходимое полеФамилия и выберите командуПравая кнопка мыши/Найти (рис. 2.4).

6.Измените имя поля «Номер паспорта» на «Паспортные данные» в режиме «Таблицы», для этого установите указатель на имя поля и выполните двойной щелчок мыши.

7.Удалите поле Паспортные данные,используя команду Правая кнопка мыши/Удалить столбец.Не забудьте предварительно выделить поле и в

процессе работы дать подтверждение на удаление.

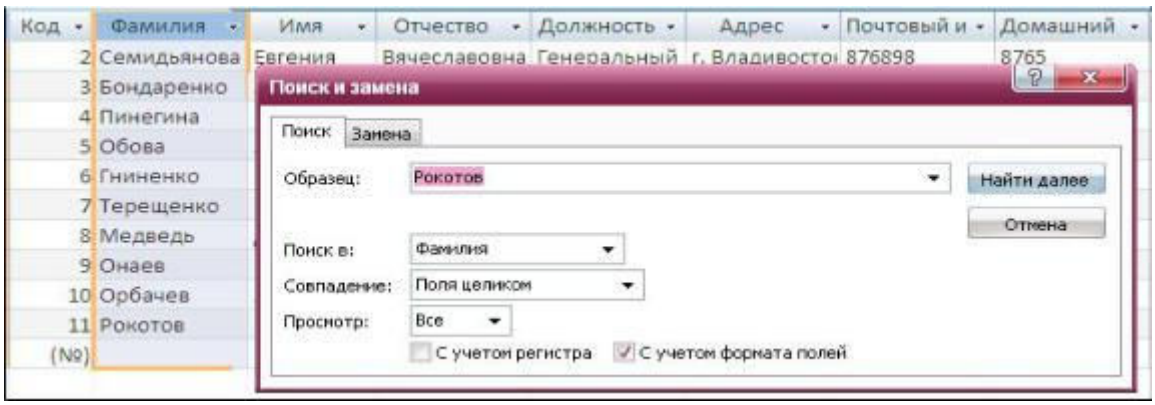



Рис. 2.4. Поиск записей

7. Добавьте в таблицу «Сотрудники фирмы» перед полемПримечание новые поля:Ставка, Премия, Зарплата. Для этого выделите поле

Примечание и выберите командуВставка/Столбец. Присвойте созданным полям соответствующие имена.

8. Перейдите в режимКонструктор и проверьте, а при необходимости измените типы данных созданных полей (созданные поля должны иметь

числовой или денежныйтип данных). Вернитесь в Режим таблицы.

9.Заполните поле Ставка числовыми данными. Для корректной дальнейшей работы наберите несколько ставок со значениями в интервале 2000... 3000 р.

Примечание. Для удобства работы некоторые поля можно скрыть командойПравая кнопка мыши / Скрыть столбцы, для вызова скрытых столбцов воспользуйтесь командойПравая кнопка мыши /Отобразить столбцы.

10.Сохраните изменения в таблице.

Задание 2.2. Произвести расчеты значений Премии и Зарплаты в таблице «Сотрудники фирмы». Премия составляет 27 % от Ставки, а Зарплата рассчитывается как сумма полейПремия иСтавка.

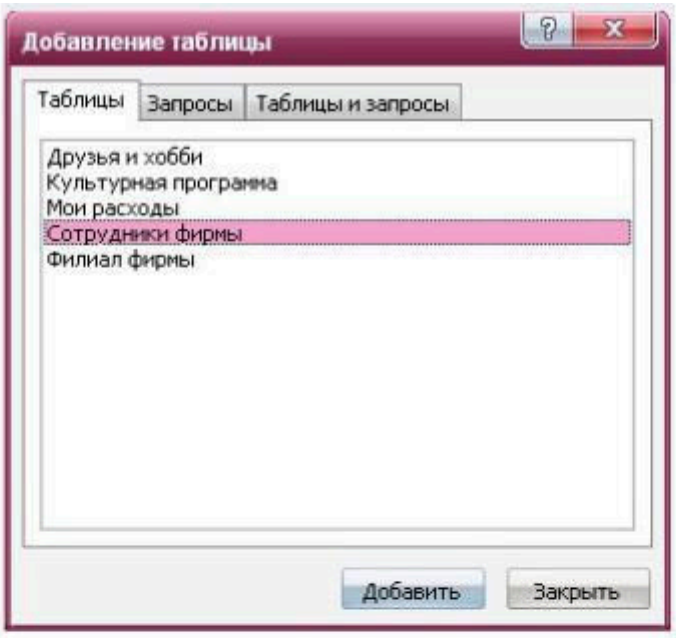
Порядок выполнения работы:

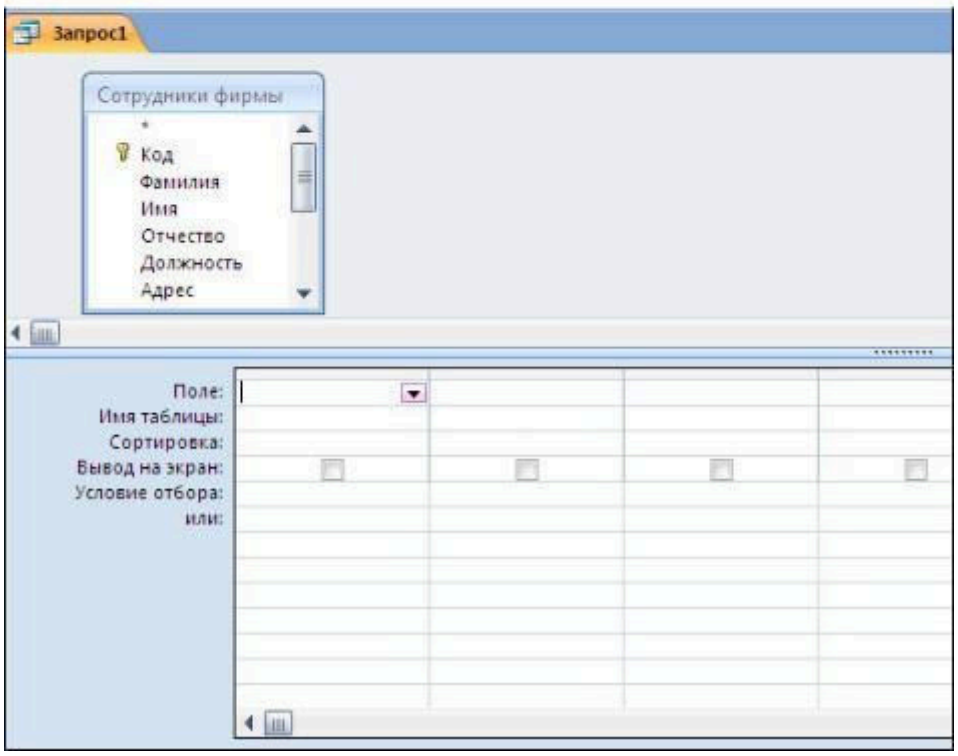
1.Откройте таблицу «Сотрудники фирмы».

2.Для заполнения полей Премия иЗарплата вызовите бланк запроса командойКонструктор запросов.

Краткая справка. Бланк запроса – это бланк, предназначенный

для определения запроса или фильтра в режиме Конструктор запроса. 3. В открывшемся диалоговом окнеДобавление таблицы выберите таблицу «Сотрудники фирмы», нажмите кнопкуДобавить и закройте это окно (рис. 2.6), при этом к бланку запроса добавится список полей



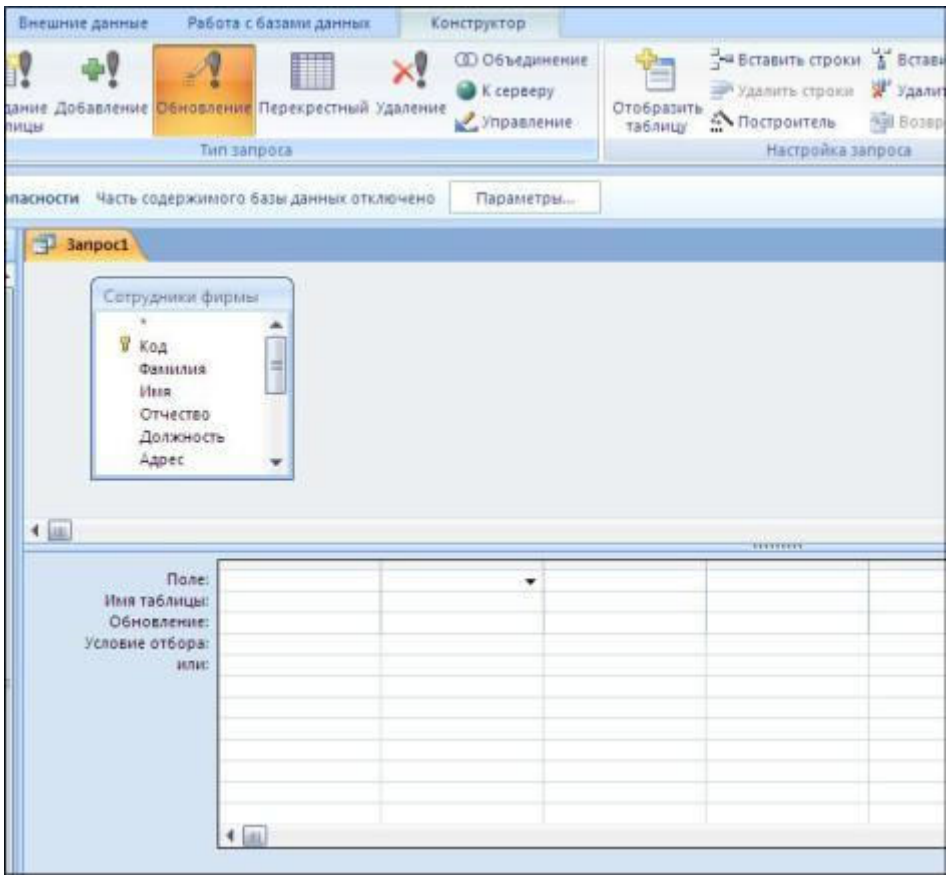




таблицы «Сотрудники фирмы» (рис. 2.7). По умолчанию откроется бланк запроса на выборку.

Краткая справка. Список полей (в форме и отчете) — окно небольшого размера, содержащее список всех полей в базовом источнике записей. В базе данных Microsoft Access имеется возможность отобразить список полей в режимеКонструктор форм, отчетов и запросов, а также в окнеСхема данных.

Рис. 2.6. Добавление списка полей таблицы «Сотрудники фирмы»

Рис. 2.7. Бланк запроса на выборку




4. В Конструкторе выберите командуОбновление (рис. 2.8). Обратите внимание на изменения в бланке запроса («Сортировка» изменилась на «Обновление»).

Рис. 2.8. Выбор запроса на обновление

5. Из списка полей в бланк запроса перетащите поля, которые нужно обновить — Премия иЗарплата; в строке «Обновление» введите расчетные формулы сначала для заполнения поляПремия, а затем — поляЗарплата (Премия составляет 27 % от Ставки, а Зарплата рассчитывается как сумма полейПремия иСтавка).

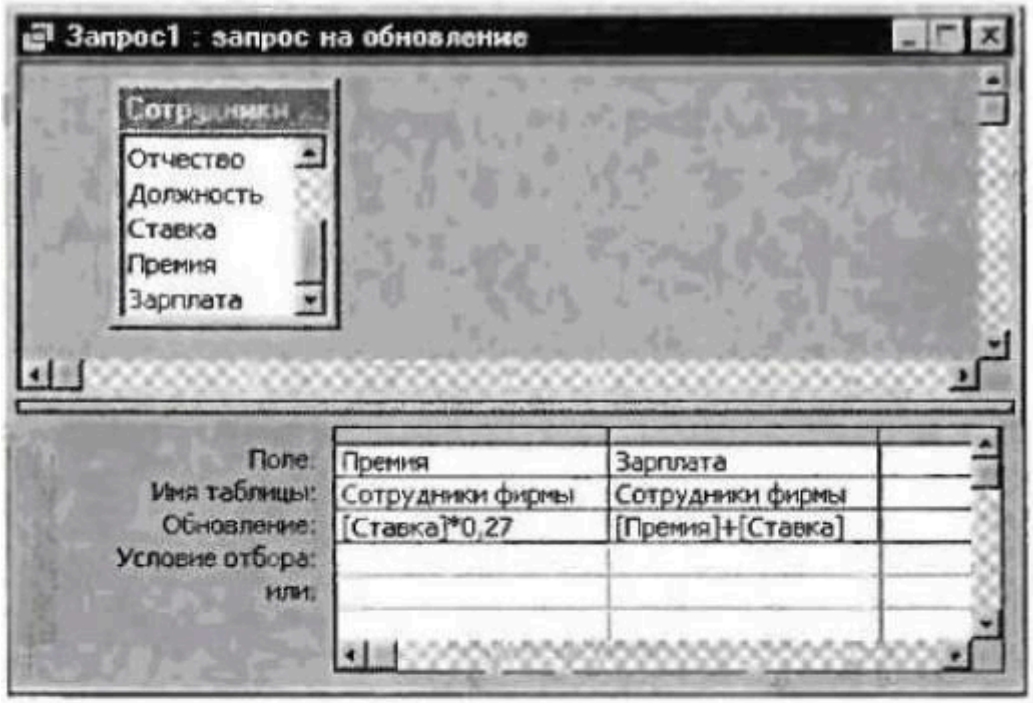
Для расчета Премии в строке «Обновление» наберите:[Премия] * 0,27;

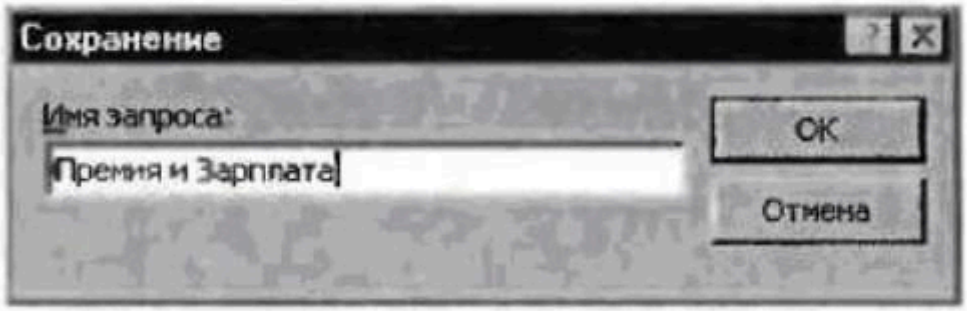
Для расчета Зарплаты наберите: [Премия] + [Ставка] (рис. 2.9). Сохраните запрос под именем «Премия и Зарплата» (рис. 2.10).

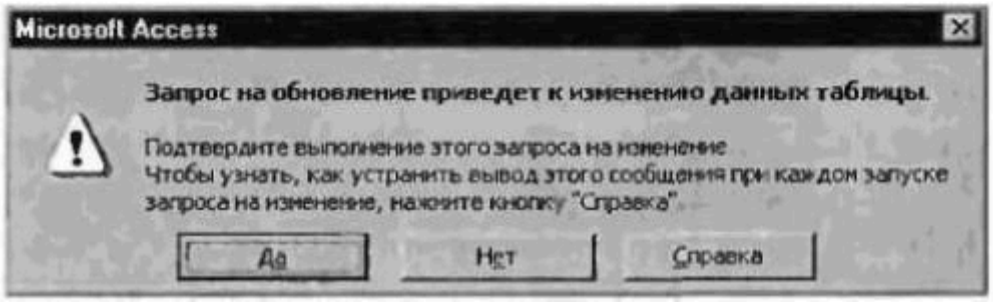
6.Проведите обновление по запросу, для чего дважды запустите на исполнение запрос на обновление «Премия и Зарплата». При этом подтвердите выполнение запроса кнопкой Да в открывающемся диалоговом окне (рис. 2.11).

7.Откройте таблицу «Сотрудники фирмы» и проверьте правильность расчетов. Если все сделано правильно, то поля Премия иЗарплата будут

заполнены рассчитанными результатами.



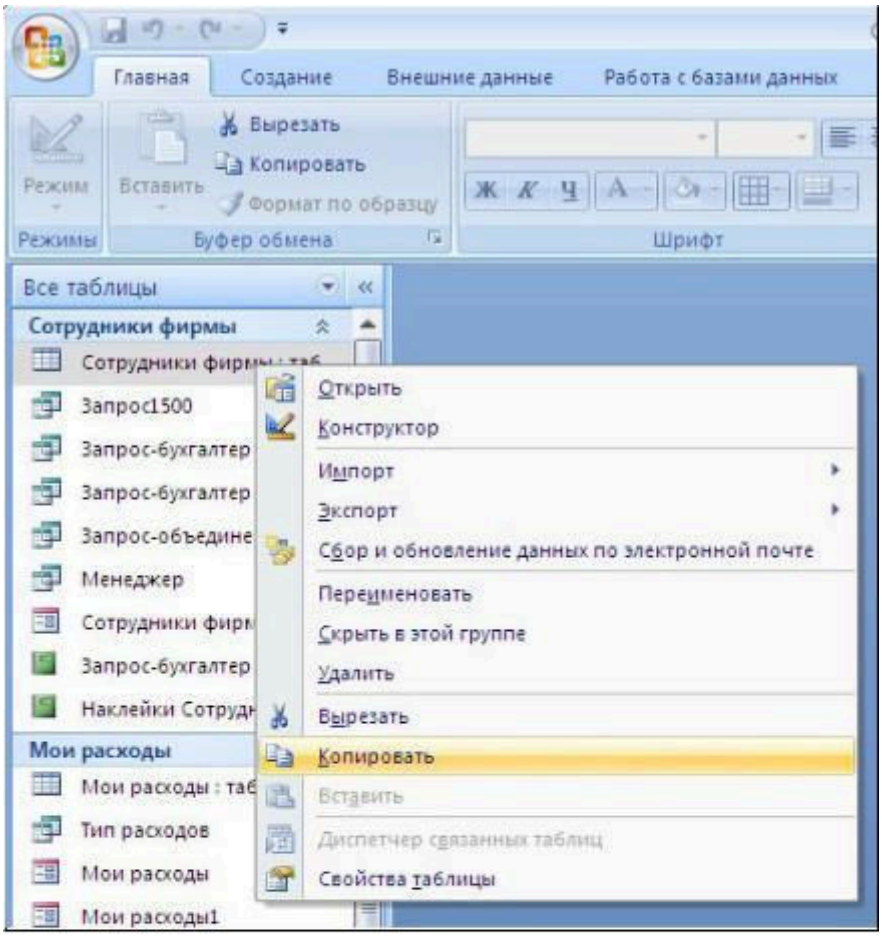

8. Измените последовательность полей: полеПримечание поместите перед полемСтавка. Выделить полеПримечание, мышью перетащить на новое место.

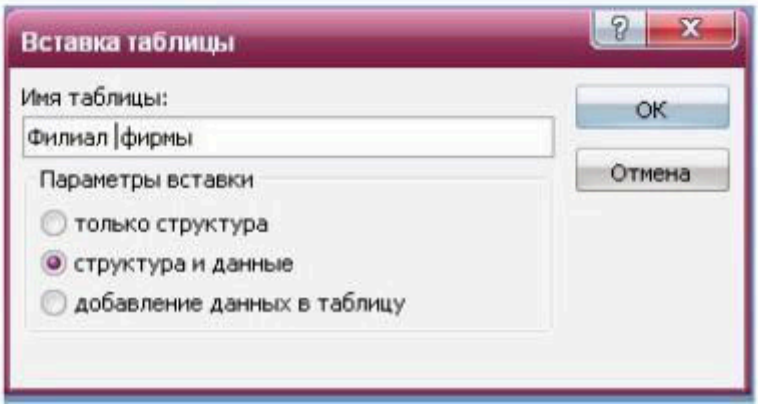
Рис. 2.9. Бланк запроса для расчета полей Премия иЗарплата

Рис. 2.10. Задание имени запроса при сохранении

Рис. 2.11. Окно подтверждения выполнения запроса на обновление

9. Сохраните изменения в таблице. В случае необходимости создайте




резервную копию БД на флешке.

Задание 2.3. Создать копию таблицы «Сотрудники фирмы». Новой таблице присвойте имя «Филиал фирмы». Произведите изменения в составе полей таблиц.

Порядок выполнения работы:

1.Запустите программу СУБД Microsoft Access и откройте свою созданную базу данных. Выберите объект базы – Таблицы.

2.Для копирования в окне База данных установите курсор на таблицу «Сотрудники фирмы» и выберите командуПравая кнопка мыши/Копировать (рис. 2.12), далееПравая кнопка мыши/Вставить.

Рис. 2.12. Копирование таблицы в окне базы данных

Рис. 2.13. Ввод имени копируемой таблицы

В появившемся окне Вставка таблицы введите новое имя таблицы «Филиал фирмы» и выберите переключатель «Структура и данные» (рис. 2.13).

3.Удалите часть полей в таблицах «Сотрудники фирмы_копия» и «Филиал фирмы», а также переместите поля в них в соответствии с заданием.

В таблице «Филиал фирмы» должны остаться поля: Код, Фамилия, Имя, Отчество, Должность, Домашний телефон, Табельный номер, Дата рождения, Дата найма. В таблице «Филиал фирмы» должны остаться поля:Код, Фамилия, Имя, Примечание, Ставка, Премия, Зарплата.

4.Просмотрите таблицы «Сотрудники фирмы» и «Филиал фирмы» в режимеПредварительный просмотр.

Сохраните изменения в таблицах.

Дополнительные задания Задание 2.4. В той же БД в таблице «Филиал фирмы» добавить новые

поля Доплата иИтого и произвести расчеты (созданием запроса на обновление) по формулам:

Доплата = 42 % от зарплаты (в строке «Обновление» поля Доплата наберите – [Зарплата]* 0,42);

Итого = Зарплата+Доплата (в строке «Обновление» поля Итого наберите – [Зарплата] + [Доплата]).

Задание 2.5. В той же БД в таблице «Филиал фирмы» произвести поиск фамилииРокотов и замену ее на фамилиюСтоляров.

Краткая справка. Для поиска и замены установите курсор в поле (столбец), по которому нужно выполнять поиск, и выполните команду

Правая кнопка мыши/Поиск. В открывшемся окне Поиск и заменана вкладке Поискв строку «Образец» введите фамилию Рокотов,а на вкладке Заменав строку «Заменить на» введите Столярови нажмите кнопку

Заменить все.

# Лабораторная работа № 24,25.

## Тема: Составление и выполнение запросов на выбор из одной таблицы

1. Разработать запросы, необходимые для решения задач базы данных;
2. Разработать запросы:
   - многотабличный запрос выборки с сортировкой и отбором данных,
   - многотабличный запрос выборки с внешним объединением,
   - параметрический запрос,
   - запрос с применением вычисляемых полей,
   - запрос группировкой и вычислением итогов,
   - перекрестный запрос, где применимо,
   - запрос на добавление (в SQL),
   - запрос на удаление (в SQL),
   - запрос на обновление (в SQL),
   - запрос на создание новой таблицы на основе существующей (в SQL),
   - запрос на объединение (UNION), где применимо,
   - запрос на создание новой таблицы (в SQL).
3. Изучить SQLна примере созданных запросов. Разработать вложенный запрос наSQL;

### Теория

**SQL. Запросы манипулирования данными**

**Запросы выборки**

Основу DML инструкций составляет инструкция SELECT.

SELECT [ALL|DISTINCT|DISTINCTROW] <список полей>

FROM <список таблиц> [IN <имя базы данных>]

[WHERE <условие отбора>]

[<Таблица1> INNER|RIGHT|LEFT JOIN <Таблица2>

ON <Таблица1>.<Поле1> = <Таблица2>.<Поле2>]

[GROUP BY <список полей>]

[HAVING <условие отбора>]

[ORDER BY <спецификация> [,<спецификация>] ...]

Инструкция SELECT позволяет производить выборку и вычисления над данными из одной или нескольких таблиц. Результатом выполнения инструкции является ответная таблица. Простейшая форма инструкции SELECT включает операторы SELECT и FROM. Оператор SELECT определяет поля, подлежащие выводу в выходной набор, а оператор FROM определяет имена таблиц, включенных в запрос. Имена полей и таблиц отделяются запятыми. Если в запросе участвуют несколько таблиц, то для исключения двусмысленности имена полей следует записывать в полной форме: <таблица>.<поле>

(например, Клиенты.Адресс). Для повышения эффективности вначале указываются меньшие таблицы. Если имена полей и таблиц включают пробелы, то их необходимо заключать в квадратные скобки. Список полей следует задавать в той последовательности, в которой они должны быть представлены в выходном наборе. Например:

SELECT Фамилия, Имя, Отчество, [год рождения] FROM Клиенты;

Для выбора всех полей применяется операция *: SELECT * FROM Клиенты;

Список данных может содержать имена полей, участвующих в запросе, а также выражения над столбцами или строковые константы. В выражениях (вычисляемых полях) могут принимать участие имена полей, знаки арифметических операций (+, – ,* ,/, ^), множество встроенных функций, константы, круглые скобки и следующие операции:

\ – возвращает целое от деления двух арифметических выражений (заранее округленных);

MOD – возвращает остаток от деления двух арифметических выражений (заранее округленных);

& – операция конкатенации;

IIF(условие, выражение1, выражение2) – Если условие истинно, то возвращается результат выражения1, в противном случае – выражения2. Например, если одно из полей, участвующих в сложении, будет пустым, то и весь результат будет пустым. Функция IIF позволяет исправлять значения пустых полей на 0: IIF(IsNull([поле]), 0, [поле]).

SELECT [Фамилия] & ' ' & [Имя] & ' ' & [Отчество] AS Полное_имя, “возраст”, Year(Date())-Year([год рождения]) AS Возраст FROM Клиенты;

Именам полей и таблиц можно назначать альтернативные имена (псевдонимы). Псевдонимы записываются через оператор AS (имя таблицы или поля AS псевдоним). В предыдущем примере псевдонимы использовались для задания имен вычисляемым полям. В противном случае имя вычисляемого поля имело бы вид: Выражение1 и т.д. В большинстве случаев псевдонимы используются для сокращения набора длинных имен. Особенно это эффективно для замены длинных имен таблиц, поскольку в многотабличных запросах приходится включать имена таблиц в описание каждого поля.

Если в запрос включены не все поля некоторой таблицы, то выходной набор может содержать одинаковые строки. Предикаты ALL, DISTINCT, DISTINCTROW (записываются сразу после оператора SELECT) служат для управления выводом повторяющихся строк. По умолчанию принимается предикат ALL, т.е. в ответную таблицу включаются все строки, в том числе и повторяющиеся. Предикаты DISTINCT и DISTINCTROW исключают повторяющиеся записи. Отличие состоит в том, что предикат DISTINCTROW применяется для многотабличных запросов. Если данные выбираются только из одной таблицы, то он игнорируется.

Оператор WHERE необязателен. Он задает условия, которым должны удовлетворять записи в результирующей таблице. Выражение <условие отбора> является логическим. Его элементами могут быть имена столбцов, операции сравнения <, <=, >, >=, =, <>, арифметические операции, логические операторы (NOT, AND, OR, XOR, IMP, EQV), скобки, функции IN, BETWEEN, LIKE, IS (NOT) NULL и множество встроенных функций. Строки заключаются в кавычки, а константы типа Дата/Время – в символы #.

Функция IN проверяет на равенство любому значению из списка: IN (“Минск”, “Москва”, “Киев”);

Функция BETWEEN задает диапазон значений. Границы диапазона разделяются оператором And: BETWEEN 50 And 100.

Функция LIKE проверяет на соответствие заданному шаблону символов. В качестве символов шаблона используются:

* – любое число произвольных символов. Может использоваться также %;

? – один произвольный символ. Может использоваться также _;

# – одна произвольная цифра;

[] – диапазон допустимых символов. К примеру, [А - Я], [3 - 9]. Если наоборот необходимо исключить эти символы, то перед ними ставится !: [!А - Я]. Например, LIKE “######”, LIKE “А*”, LIKE “####[К-НХВАРОСМТ][К-НХВАРОСМТ][К-НХВАРОСМТ]”.

Запрос может быть основан на нескольких таблицах. Простое включение полей из нескольких таблиц даст простой перебор всех их возможных комбинаций. Для двух таблиц общее число записей будет равно произведению числа записей в первой и второй таблицах. Но так как реляционная база данных практически всегда состоит из таблиц, связанных между собой посредством совпадающих значений полей, участвующих в связи, то для правильного объединения данных необходимо включить в запрос явное определение соответствующих связей. Связи можно задать с помощью двух способов: с помощью оператора INNER|RIGHT|LEFT JOIN и с помощью дополнительного условия выборки после оператора WHERE. Причем в SQL объединение данных можно произвести даже по неравенству, т.е. поддерживаются операции сравнения =, <>, <, <= ,> , >=. Оператор INNER|RIGHT|LEFT JOIN является необязательной частью инструкции SELECT и оформляется как часть оператора FROM.

SELECT Товары.[Наименование товара], Заказы.Дата, Заказы.[Полная цена] FROM Товары INNER JOIN Заказы ON Товары.Код_товара = Заказы.Код_товара;

Этот же запрос можно записать следующим образом (второй способ задания связи): SELECT Товары.[Наименование товара], Заказы.Дата, Заказы.[Полная цена] FROM Товары, Заказы WHERE Товары.Код_товара = Заказы.Код_товара;

Для большинства многотабличных запросов набор записей формируется на основе совпадающих значений полей, участвующих в связи, т.е. с помощью внутреннего объединения (INNER JOIN). Для двух таблиц, связанных отношением «один ко многим», из главной таблицы будут отобраны только те записи, которые имеют связанные записи в подчиненной таблице. Допустим, мы имеем две таблицы, «Клиенты» и «Заказы». С помощью внутреннего объединения мы получим сведения по клиентам, имеющим хотя бы один заказ. Но предположим, что мы хотим получить информацию по всем клиентам и посмотреть, кто заказывал что-либо, а кто нет. Ответ на такой вопрос позволяют дать запросы с внешним объединением (LEFT|RIGHT [OUTER] JOIN). Тогда в запрос будут включены все записи с таблицы «Клиенты» и те записи с таблицы «Заказы», которые имеют связанные записи в главной таблице.

SELECT Фамилия, Имя, [Дата заказа], Цена FROM Клиенты LEFT JOIN Заказы ON Клиенты.код_клиента = Заказы.код_клиента;

**Запросы с группировкой**

Иногда интерес будут представлять не каждая строка таблицы в отдельности, а итоговые значения по группам данных. Например, может понадобиться общая сумма продаж для клиентов, проживающих в определенном районе или интересно знать средний объем продаж по месяцам, чтобы выяснить тенденции сбыта. Получить ответы на такие вопросы можно с помощью итоговых запросов (запросов с группировкой). Оператор GROUP BY позволяет выделять группы в результирующем множестве записей. Группой являются записи с совпадающими значениями в полях, перечисленных за оператором GROUP BY. Оператор перегруппирует таблицу таким образом, чтобы в каждой группе все строки имели одно и тоже значение поля. Инструкция SELECT затем применяется уже к группам перекомпонованной таблицы. Каждое выражение во фразе SELECT должно принимать единственное значение для группы, т.е. оно может быть либо самим полем, либо арифметическим выражением, включающем это поле, либо константой, либо агрегатной функцией, возвращающей единственное значение для группы. В запросах с группировкой необходимо тщательно следить за включение полей во фразу SELECT, так как в противном случае можно не получить желаемого результата. Если во фразуSELECTбудет помещено хотя бы одно поле, которое не является единственным для группы, например ключевое поле подчиненной таблицы, то создание групп будет прервано, так как в результате каждая строка запроса будет уникальна.

В результате запроса только с группировкой по некоторому полю получится таблица, содержащая по одной записи для каждой группы. Группирование записей само по себе ничего не дает. Обычно производятся вычисления для групп. Для этой цели имеется ряд групповых (иначе агрегатных) функций:

SUM – вычисляет сумму всех значений заданного поля в каждой группе;

AVG – вычисляет среднее арифметическое всех значений заданного поля в каждой группе;

STDEV – вычисляет стандартное отклонение всех значений заданного поля в каждой группе;

VAR – вычисляет дисперсию всех значений заданного поля в каждой группе;

COUNT – вычисляет число записей, для которых значение заданного поля отлично от NULL. Для подсчета всех записей необходимо использовать операцию *: Count(*);

MIN – возвращает минимальное значение заданного поля в каждой группе;

MAX – возвращает максимальное значение заданного поля в каждой группе;

FIRST – возвращает первое значение заданного поля в каждой группе;

LAST – возвращает последнее значение заданного поля в каждой группе;

Групповые функции могут применяться сами по себе, без выполнения группировки. Тогда группой будет считаться все отобранные оператором WHERE записи. Например: SELECT Count(*) FROM Заказы – подсчитает все записи в таблице заказы,

SELECT Sum([Отпускная цена] + [Транс издержки]) As Полная_цена FROM Заказы WHERE Город = “Минск”; – подсчитает полную сумму цен по всем заказам, сделанным из Минска.

SELECT Клиенты.Фамилия, Sum(Заказы.Цена) AS Стоимость FROM Клиенты INNER JOIN Заказы ON Клиенты.КодКлиента = Заказы.КодКлиента WHERE Клиенты.Город ="Минск" GROUP BY Клиенты.Фамилия;

Оператор HAVING действует совместно с оператором GROUP BY и используется для дополнительной селекции записей во время определения групп. Он выполняет те же функции, что и WHERE, но уже в рамках выходного набора. Оператор HAVING устанавливает, какие записи, сгруппированные посредством GROUP BY, должны отображаться и участвовать в групповых операциях. Правила записи < условия отбора > аналогичны правилам формирования <условия отбора> оператора WHERE.

SELECT код_товара FROM Заказы GROUP BY код_товара HAVING Count(*) > 1; - отбирает коды товаров, покупаемых более чем одним покупателем.

Оператор ORDER BY задает порядок сортировки результирующего множества. Обычно он замыкает инструкцию SELECT. Каждая <спецификация> сортировки представляет собой пару вида: <имя поля> [ASC/DESC]. Большинство СУБД требуют, чтобы поле, участвующее в сортировке, присутствовало в выходном наборе.

SELECT Наименование FROM Товары ORDER BY Наименование;

**Параметрические запросы**

До сих пор условие отбора мы записывали явно после оператора WHERE. Но во многих случаях условие отбора становится известным только во время выполнения программы. SQL позволяет вводить условия отбора в виде параметров запроса. В этом случае такие запросы называются параметрическими или динамическими. Для увеличения эффективности выполнения таких запросов можно вызвать функцию PREPARE, которая подготовит запрос к запуску, т.е. зарезервирует необходимую память и проведет его оптимизацию. Тогда сразу после задания параметра запрос будет готов к запуску. Для освобождения зарезервированной памяти применяется функция UNPREPARE. Для задания параметра в SQL MS Access необходимо некоторый текст (который будет выводится на экран в окне задания параметра) заключить в квадратные скобки. Имя параметра должно отличаться от названий полей таблиц, включенных в запрос. Типы параметров можно явно определить с помощью инструкции PARAMETERS в виде [имя параметра] тип данных, [имя параметра] тип данных, …, что позволяет контролировать значения параметров на соответствие типу еще при их вводе. В таком случае инструкция PARAMETERS располагается перед описанием запроса.

PARAMETERS [Год:] INT;

SELECT Клиенты.Фамилия FROM Клиенты INNER JOIN Заказы ON Клиенты.КодКлиента = Заказы.КодКлиента WHERE Year(Заказы.Дата) = [Год:];

**Вложенные запросы**

Инструкции SELECT могут многократно вкладываться друг в друга. Вложенная инструкция SELECT записывается после оператора WHERE и служит для отбора записей основного запроса. SQL выполняет вложенный подзапрос и затем сравнивает каждую строку основного запроса с результатом вложенного. Вложенные запросы записываются внутри скобок. Например,

SELECT Фамилия, Имя FROM Клиенты WHERE Кредит < (SELECT AVG(Кредит) FROM Клиенты);

Если подчиненный запрос возвращает набор записей, то вместо операторов сравнения можно использовать функции ALL, SOME, ANY, EXISTS, IN, получившие по функции EXISTS название кванторов существования. Квантор существования EXISTS обычно записывается следующим образом EXISTS(SELECT * FROM …). Перед ним можно поставить NOT для инверсии результата. Выражение EXISTS считается истинным тогда и только тогда, когда результат вычисления подзапроса является непустым множеством. В качестве примера выберем фамилии покупателей, которым был продан компьютер «Pentium II 350»:

SELECT Фамилия FROM Клиенты WHERE EXISTS(SELECT * FROM Заказы WHERE Заказы.код_клиента = Клиенты.код_клиента And [Наименование товара] = ”Pentium II 350”);

Кванторы SOME и ANY (синонимы) используются для отбора в главном запросе записей, которые удовлетворяют сравнению с записями, отобранными подчиненным запросом. Например: SELECT * FROM Товары WHERE Цена > ANY(SELECT Цена FROM Заказано WHERE Скидка >= 0.25); Этот запрос отбирает все товары, цена которых больше, чем цена любого товара (т.е. самого недорогого), проданного со скидкой 25%.

Квантор ALL используются для отбора в главном запросе записей, которые удовлетворяют сравнению со всеми записями, отобранными подчиненным запросом.

SELECT Марка FROM Товары WHERE Цена > ALL(SELECT Цена FROM Заказано WHERE Скидка >= 0.25); Этот запрос отбирает все товары, цена которых больше, чем цена всех товаров (т.е. самого дорогого), проданных со скидкой 25%.

Квантор IN используются для отбора в главном запросе только тех записей, которые содержат значения, совпадающие с одним из отобранных подчиненным запросом.

SELECT Марка FROM Товары WHERE Цена IN(SELECT Цена FROM Заказано WHERE Скидка >= 0.25); Этот запрос отбирает все товары, цена которых совпадает с ценой товаров, проданных со скидкой 25%.

**Запросы действий**

С помощью запросов действий пользователь может изменять или переносить данные в таблицах, обновлять, добавлять или удалять группы записей, а также создавать новые таблицы. Существует четыре запроса действий: на добавление записей, на удаление записей, на обновления записей и на создание таблицы.

***Запрос на обновление***

С помощью запроса на обновление можно изменить группу записей, отобранных по заданному критерию. Инструкция запроса на обновление имеет формат вида:

UPDATE <имя таблицы>

SET <имя поля> = {<выражение> , NULL } [, SET <имя поля > = {<выражение> , NULL } ... ]

[WHERE <условие отбора>]

Новые значения полей в записях могут быть пустыми (NULL), либо вычисляться в соответствии с арифметическим выражением. Правила записи арифметических и логических выражений аналогичны соответствующим правилам для вычисляемых полей. Например,

UPDATE Товары

SET Наименование = “Pentium III 800”, Цена = Цена + 250

WHERE Наименование = “Pentium II 350”;

***Запрос на добавление***

С помощью запроса на добавление записи одной таблицы (все или отобранные запросом) можно добавить в конец другой таблицы. Этот запрос также позволяет добавить в таблицу всего одну запись, состоящую из литералов. Соответственно запрос на добавление имеет форматы двух видов:

INSERT INTO <имя таблицы> [(<список полей>)] VALUES (<список значений>) и

INSERT INTO <имя таблицы> [(<список полей>)] <инструкция SELECT>

В первом формате оператор INSERT предназначен для ввода одной записи состоящей из литералов или выражений. Порядок перечисления полей должен соответствовать порядку значений, перечисленных в списке значений оператора VALUES. При явном перечислении можно опускать некоторые поля. Если список полей опущен, то в списке значений должны быть перечислены все значения в порядке следования полей таблицы. Во втором формате оператор INSERT предназначен для добавления записей, отобранных из другой таблицы с помощью инструкции SELECT. Здесь также необходимо обеспечить соответствие полей (типов и размеров полей или, по крайней мере, возможность полноценной конвертации данных), перечисленных как после оператора INSERT INTO , так и после SELECT. Например,

INSERT INTO Товары(Наименование, Цена) VALUES («Pentium II 233», 450);

INSERT INTO Клиенты(Фамилия, Имя, Отчество) SELECT Фамилия, Имя, Отчество FROM Поставщики WHERE Город = “Минск”;

***Запрос на удаление***

С помощью запроса на удаление можно сразу удалить группу записей, удовлетворяющих определенному критерию. Этот запрос особенно эффективен при удалении большого

числа записей. С помощью этого запроса можно явно удалить записи только из одной таблицы. Но если было определено каскадное удаление, то будут также удалены связанные записи из подчиненных таблиц. Запрос на удаление имеет формат вида:

DELETE FROM <имя таблицы> [WHERE <условие отбора>]

Если необязательный оператор WHERE опущен, т. е. условие отбора удаляемых записей отсутствует, удалению подлежат все записи таблицы. Например,

DELETE FROM Товары WHERE Наименование LIKE “Celeron*”;

***Запрос на создание новой таблицы***

С помощью этого запроса можно создать новую таблицу на основе уже существующей и перенести в нее все или удовлетворяющие некоторому критерию записи. Новая таблица будет иметь ту же структуру. Обычно этот запрос применяется для создания архивных копий таблиц. Запрос на создать новой таблицы имеет формат вида:

SELECT <список полей> INTO <имя новой таблицы>

FROM <имя таблицы> [WHERE <условие отбора>].

В новую таблицу будут перенесены поля, перечисленные в списке полей. Например:

SELECT Клиенты.Фамилия, Клиенты.Имя, Клиенты.Отчество INTO Совершеннолетние FROM Клиенты WHERE Клиенты.[год рождения]>#1984/01/04#;

**Перекрестные запросы**

Инструкция TRANSFORM служит для создания перекрестного запроса. Перекрестные запросы позволяют создать таблицу (перекрестную), в которой можно будет просмотреть зависимость некоторых данных от двух параметров, один из которых откладывается по строкам, второй по столбцам.

В общем случае инструкция TRANSFORM записывается следующим образом:

TRANSFORM выражение с групповой функцией (вычисляет значения ячеек)

SELECT инструкция с применением GROUP BY (отбирает записи и задает имена строк)

PIVOT выражение, задающее имена столбцов.

К примеру, нам надо определить объем выручки от продажи каждого товара по месяцам. Тогда каждая строка полученной таблицы будет соответствовать определенному товару, в качестве столбцов будут названия месяцев, а в ячейках таблицы будут представлены соответствующие объемы продаж.

| Месяцы | Янв 2002 | Февр 2002 | Март 2002 | Апр 2002 |
|---|---|---|---|---|
| Товар | | | | |
| Молоко | 4363 | 674 | 8555 | 12 |

| | | | | |
|---|---|---|---|---|
| Мясо | 34636 | 8654 | 22 | 436 |
| Птица | 3363 | 54375 | 25252 | 131 |
| Рыба | 33656 | 34633 | 2554 | 0 |

TRANSFORM Sum(Заказы.Цена* Заказы.Количество) AS СуммаПоТовару

SELECT Товары.Товар FROM Товары INNER JOIN Заказы ON Товары.КодТовара = Заказы.КодТовара WHERE Заказы.[Дата заказа] BETWEEN #2002/1/1# And #2002/30/4# GROUP BY Товары.Товар

PIVOT Format([Дата заказа],"mmm yyyy");

В выражении PIVOT можно также применить функцию DatePart("m",[ДатаРазмещения],1,0), возвращающую номера месяцев.

**Запрос на объединение**

Оператор UNION позволяет объединить выходные наборы нескольких инструкций SELECT в одну результирующую таблицу. Этот запрос является аналогом операции объединения отношений $R = A \cup B$ в реляционной алгебре. При объединении записи, возвращаемые второй и последующими инструкциями SELECT, будут дописываться в конец первой, причем из результата выборки будут всегда исключаться повторяющиеся записи. В выходных наборах всех инструкций SELECT должно быть одинаковое количество полей с одинаковыми характеристиками. В крайнем случае некоторую инструкцию SELECT можно дополнить строковыми константами или вычисляемыми полями. В качестве заголовков столбцов для выходного набора будут приниматься имена полей первой инструкции. Для выполнения сортировки результирующего набора данных объединение может дополняться оператором ORDER BY, который записывается в конце последней инструкции SELECT.

SELECT Фамилия & " " & Имя & " " & Отчество As Название, Город FROM Клиенты

UNION SELECT Поставщик, Город FROM Поставщики

ORDER BY Название, Город;

Кроме UNION в SQL2 включены еще две операции реляционной алгебры: исключения – EXCEPT (аналог операции разности $R = A - B$. Отношение*R*включает записи, присутствующие в*A*и отсутствующие в*B*) и пересечения – INTERSECT (аналог операции пересечения $R = A \cap B$. Отношение*R*включает записи, присутствующие как в*A*, так и в*B*). Как и для объединения, операнды (таблицы) этих операций должны содержать соответствующий по типам набор полей.

**Создание таблиц**

Оператор создания таблицы имеет формат вида:

CREATE TABLE <имя таблицы> (<имя поля> <тип данных> [NOT NULL] [,<имя поля> <тип данных> [NOT NULL]] ...)

Обязательными операндами оператора являются имя создаваемой таблицы и имя хотя бы одного поля с указанием типа данных. При создании таблицы для отдельных полей могут указываться некоторые дополнительные правила контроля вводимых в них значений. Конструкция NOT NULL требует, чтобы в этом столбце должно быть определено значение. Например,

CREATE TABLE Товары( Код CHAR(5) NOT NULL, Тип CHAR(8), Наименование VARCHAR(20) NOT NULL, Цена DECIMAL(8,2) );

SQL поддерживает пять скалярных типов данных: символьный (строковый), битовый, численный, даты/время и интервал. Символьный тип данных позволяет определить строки фиксированной (CHAR(n) или CHARACTER(n)) и переменной длинны (VARCHAR(n) или CHAR VARYING(n) или CHARACTER VARYING(n)), где *n*означает максимальное количество символов. Спецификация VARYING позволяет сохранять в базе только вводимые символы, иначе введенная строка дополняется пробелами до*n*символов. Численный тип данных позволяет задавать целые числа разного размера (SMALLINT, INT или INTEGER) и вещественные числа разной точности (FLOAT [precision], REAL, DOUBLE PRECISION, NUMERIC(precision, scale), DECIMAL(precision, scale) или DEC (precision, scale)), где precision определяет полное число десятичных цифр, а scale – число цифр после десятичной точки. Битовые типы данных позволяют хранить последовательности двоичных цифр фиксированной (BIT(n)) и переменной длинны (BIT VARYING(n)). Для хранения данных типа даты/время введены три типа данных: DATE, TIME и TIMESTAMP. Типы DATE и TIME позволяют хранить соответственно дату и время по отдельности, а TIMESTAMP – вместе. Тип данных INTERVAL определен для хранения интервалов времени. При определении доменов типы данных задаются после оператора AS, например: Name AS VARCHAR(20).

Оператор изменения структуры таблицы имеет формат вида:

ALTER TABLE <имя таблицы>

( {ADD, MODIFY, DROP} <имя поля> [<тип данных>] [NOT NULL]

[,{ADD, MODIFY, DROP} <имя поля> [<тип данных>] [NOT NULL]]...)

Изменение структуры таблицы может состоять в добавлении (ADD), изменении (MODIFY) или удалении (DROP) одного или нескольких столбцов таблицы. Правила записи оператора ALTER TABLE такие же, как и оператора CREATE TABLE, разве что при удалении столбца указывать тип данных не требуется. Для примера добавим одно поле:

ALTER TABLE Товары(ADD Категория VARCHAR(20));

Оператор удаления таблицы имеет формат вида: DROP TABLE <имя таблицы>.

**QBE**

Запросы в Access можно создавать как с помощью визуальных средств – QBE (querybyexample– запрос по образцу), так и с помощью SQL (structuredquerylanguage– структурированный язык запросов). Окно QBE состоит из панели задания таблиц и панели спецификации запроса. Перед созданием запроса в QBE (режим конструктора) появляется окно выбора имеющихся таблиц и запросов. Для создания запроса необходимо выбрать по

крайней мере одну таблицу (следует выбирать только те таблицы из которым необходимо выбирать данные, и те, которые требуются для установления связей). Если между таблицами были определены связи в окне*Схема данных*, то они автоматически будут установлены и в запросе. В противном случае их можно установить аналогичным способом с помощью перетягивания требуемого поля из одной таблицы в другую.

В верхней строке спецификации запроса отображаются выбранные поля. Для выбора поля достаточно перетянуть его с помощью мыши из таблицы в требуемый столбец спецификации, представляющий описание одного поля запроса, или выбрать его из выпадающего списка. Для этого поля затем можно установить порядок сортировки, вывод на экран и установить критерии выбора этого поля. Перетянув выбранные поля в область спецификации запроса, мы таким образом определили запрос. Если вам требуются все поля некоторой таблицы, то можно перетянуть первую строку этой таблицы с изображением *, означающим “Все поля”. Остается только сохранить запрос и запустить его на выполнение.

С помощью такого запроса мы выберем указанные поля для всех записей. Чтобы ограничить число записей, можно воспользоваться условием отбора. Условие отбора можно задать для каждого поля, входящего в запрос. При этом все условия отбора объединяются по логической операции И. Для реализации условия отбора по ИЛИ служат нижние строки спецификации запроса.

В поле спецификации запроса можно определять также вычисляемые поля. Правила записи вычисляемых полей совпадают с определенными в SQL. Отличается лишь задание альтернативного имени. Альтернативное имя задается перед вычисляемым полем и отделяется двоеточием, например: Полное Имя: [Фамилия]&” “&[Имя]&” “&[Отчество]”. Это поле будет отображаться наряду с другими.

Для проведения внешнего объединения в режиме конструктора дважды нажмите на линию связи между таблицами. Появится диалоговое окно параметров объединения. Остается выбрать необходимый тип объединения и закрыть окно. На связи появится стрелка, направленная к таблице со стороны ∞.

При открытии запроса в режиме конструктора в меню появляется дополнительная группа команд меню Запрос, позволяющая создавать различные типы запросов. Можно визуально создавать все запросы действий, перекрестный запрос и некоторые виды запросов SQL. Дополнительно из этого меню можно открыть окно задания типов параметров (аналог инструкции SQL PARAMETERS). Методика визуального создания запросов действия и перекрестного запроса в QBE следующая. Вначале необходимо создать запрос выборки, отбирающий необходимые данные (полезно проверить его на выполнение и убедится, что вы выбираете действительно нужные записи) затем с помощью соответствующей команды меню перевести его в нужный вид. Поле спецификации запросов будет дополнено соответствующей строкой (Обновление, Удаление и Перекрестная таблица). Остается задать необходимые значения и запустить запрос на выполнение.

Запрос с группировкой можно создать по тому же сценарию. Вначале создается запрос выборки требуемых данных, затем вызывается команда Групповые операции (доступна из локального меню поля спецификации запроса и общего меню). Эта команда дополнит поле спецификации строкой Групповые операции. В этой строке автоматически устанавливается функция GroupBY (Группировка). Кроме функции группировки в этой строке для полей запроса можно задать вычисление групповых функций или задать

условие отбора. При задании группировки необходимо помнить о том, что в группировке могут участвовать поля, которые принимают единственное значение для группы.

Для полей запроса можно поменять некоторые свойства, определенные для таблицы. Дополнительно в свойствах запроса можно задать выбор только уникальных значений (предикат DISTINCT), уникальных строк (предикат DISTINCTROW), установить блокировку записей, фильтр, тип набора записей (определяет возможность редактирования запроса) и много другого.

## Лабораторная работа № 26.

## Тема: Составление и выполнение запросов на выбор из нескольких таблиц и с использованием функций

**Теоретические сведения**

Рассмотрим следующие вопросы:

- использование объединений в запросах к нескольким таблицам;
- создание вложенных запросов.

В реальных приложениях часто требуется использовать сразу несколько таблиц БД. Запросы, которые обращаются одновременно к нескольким таблицам, называются многотабличными или сложными запросами.

**Абсолютные ссылки на базы данных и таблицы**. В запросе можно прямо указывать необходимую БД и таблицу. Например, можно представить ссылку на столбец*u_surname*из таблицы*users*в виде*users.u_surname*.Аналогично можно уточнить БД, таблица из которой упоминается в запросе. Если необходимо, то вместе с БД и таблицей можно указать и столбец, например:

```
mysql> SELECT book.users.u_surname FROM users;
+-----------+
| u_surname |
+-----------+
| Иванов    |
| Лосев     |
| Симонов   |
| Кузнецов  |
| Петров    |
| Корнеев   |
+-----------+
6 rows in set (0.00 sec)
```

При использовании сложных запросов это позволяет избежать двусмысленности при указании источника необходимой информации.

**Использование объединений для запросов к нескольким таблицам**. Хорошо спроектированная реляционная БД эффективна из-за связей между таблицами. При выборе информации из нескольких таблиц такие связи называют объединениями.

В качестве примера объединения двух таблиц рассмотрим запрос, извлекающий из БД *book*фамилии покупателей вместе с номерами сделанных ими заказов:

```
mysql> SELECT orders.order_ID, users.u_surname
    -> FROM orders, users
    -> WHERE orders.o_user_ID=users.user_ID
    -> ORDER BY orders.order_ID;
+----------+-----------+
| order_ID | u_surname |
+----------+-----------+
|        1 | Симонов   |
|        2 | Корнеев   |
|        3 | Иванов    |
|        4 | Кузнецов  |
|        5 | Симонов   |
+----------+-----------+
5 rows in set (0.00 sec)
```

Выражение *WHERE*важно с точки зрения получения результата. Набор условий, используемых для объединения таблиц, называют условием объединения. В данном примере условие связывает таблицы*orders*и*users*по внешним ключам.

Объединение нескольких таблиц аналогично объединению двух таблиц. Например, необходимо выяснить, какому каталогу принадлежит товарная позиция из заказа, сделанного 10 февраля 2009 г. в 09:40:29:

```
mysql> SELECT catalogs.cat_name
    -> FROM catalogs,books,orders
    -> WHERE o_time='2009-02-10 09:40:29'
    -> AND catalogs.cat_ID=books.b_cat_ID
    -> AND orders.o_book_ID=books.book_ID;
+----------+
| cat_name |
+----------+
| Интернет |
+----------+
1 row in set (0.09 sec)
```

**Самообъединение таблиц**. Можно объединить таблицу саму с собой (когда интересуют связи между строками одной и той же таблицы). Пусть нужно выяснить, какие книги есть в каталоге, содержащем книгу с названием «Компьютерные сети». Для этого необходимо найти в таблице*books*номер каталога (*b_cat_ID*) с этой книгой, а затем посмотреть в таблице*books*книги этого каталога.

```
mysql> SELECT b2.b_name
    -> FROM books b1, books b2
    -> WHERE b1.b_name='Компьютерные сети'
    -> AND b1.b_cat_ID=b2.b_cat_ID;
+---------------------------------------+
| b_name                                |
+---------------------------------------+
| Компьютерные сети                     |
| Сети. Поиск неисправностей            |
| Безопасность сетей                    |
| Анализ и диагностика компьютерных сетей |
| Локальные вычислительные сети         |
+---------------------------------------+
5 rows in set (0.02 sec)
```

В этом запросе для таблицы *books*определены два разных псевдонима (две отдельных таблицы *b1*и*b2*, которые должны содержать одни и те же данные). После этого они объединяются, как любые другие таблицы. Сначала ищется строка в таблице*b1*, а затем в таблице*b2* – строки с тем же значением номера каталога.

**Основное объединение**. Набор таблиц, перечисленных в выражении*FROM*и разделенных запятыми, – это декартово произведение (полное или перекрестное объединение), которое возвращает полный набор комбинаций. Добавление к нему условного выражения*WHERE*превращает его в объединение по эквивалентности, ограничивающее число возвращаемых запросом строк.

Вместо запятой в выражении *FROM*можно использовать ключевое слово*JOIN*. В этом случае вместо*WHERE*лучше использовать ключевое слово*ON*:

```
mysql> SELECT orders.order_ID, users.u_surname
    -> FROM orders JOIN users
    -> ON orders.o_user_ID=users.user_ID
    -> ORDER BY orders.order_ID;
+----------+-----------+
| order_ID | u_surname |
+----------+-----------+
|        1 | Симонов   |
|        2 | Корнеев   |
|        3 | Иванов    |
|        4 | Кузнецов  |
|        5 | Симонов   |
+----------+-----------+
5 rows in set (0.03 sec)
```

Вместо *JOIN*с тем же результатом можно использовать*CROSS JOIN* (перекрестное объединение) или*INNER JOIN*(внутреннее объединение). Пример запроса, выдающего число товарных позиций в каталогах:

```
mysql> SELECT catalogs.cat_name, COUNT(book_ID)
    -> FROM catalogs JOIN books ON catalogs.cat_ID=books.b_cat_ID
    -> GROUP BY books.b_cat_ID;
+-----------------+----------------+
| cat_name        | COUNT(book_ID) |
+-----------------+----------------+
| Программирование |              9 |
| Интернет        |              6 |
| Базы данных     |              4 |
| Сети            |              5 |
| Мультимедиа     |              6 |
+-----------------+----------------+
5 rows in set (0.05 sec)
```

Допустим, происходит расширение ассортимента и в списке каталогов появляется новый каталог «Компьютеры»:

```
mysql> INSERT INTO catalogs VALUES (NULL,'Компьютеры');
Query OK, 1 row affected (0.05 sec)

mysql> SELECT * FROM catalogs;
+--------+------------------+
| cat_ID | cat_name         |
+--------+------------------+
|      1 | Программирование |
|      2 | Интернет         |
|      3 | Базы данных      |
|      4 | Сети             |
|      5 | Мультимедиа      |
|      6 | Компьютеры       |
+--------+------------------+
6 rows in set (0.00 sec)
```

Предыдущий запрос не отразит наличие нового каталога (таблица *books*не содержит записей, относящихся к новому каталогу). Выходом является использование левого объединения (таблица*catalogs*должна быть левой таблицей):

```
mysql> SELECT catalogs.cat_name, COUNT(book_ID)
    -> FROM catalogs LEFT JOIN books ON catalogs.cat_ID=books.b_cat_ID
    -> GROUP BY books.b_cat_ID;
+------------------+----------------+
| cat_name         | COUNT(book_ID) |
+------------------+----------------+
| Компьютеры       |              0 |
| Программирование |              9 |
| Интернет         |              6 |
| Базы данных      |              4 |
| Сети             |              5 |
| Мультимедиа      |              6 |
+------------------+----------------+
6 rows in set (0.00 sec)
```

Пусть нужно вывести список покупателей и число осуществленных ими покупок, причем покупателей необходимо отсортировать по убыванию числа заказов:

```
mysql> SELECT users.u_surname,users.u_name,users.u_patronymic,
    -> COUNT(orders.order_ID) AS total
    -> FROM users JOIN orders ON users.user_ID=orders.o_user_ID
    -> GROUP BY users.user_ID
    -> ORDER BY total DESC;
+-----------+-----------+---------------+-------+
| u_surname | u_name    | u_patronymic  | total |
+-----------+-----------+---------------+-------+
| Симонов   | Игорь     | Николаевич    |     2 |
| Иванов    | Александр | Валерьевич    |     1 |
| Кузнецов  | Максим    | Петрович      |     1 |
| Корнеев   | Александр | Александрович |     1 |
+-----------+-----------+---------------+-------+
4 rows in set (0.02 sec)
```

В список не входят покупатели, которые не сделали ни одной покупки. Чтобы вывести полный список покупателей, необходимо вместо перекрестного объединения таблиц *users*и*orders*использовать левое объединение (левой таблицей должна быть таблица*users*):

```
mysql> SELECT users.u_surname,users.u_name,users.u_patronymic,
    -> COUNT(orders.order_ID) AS total
    -> FROM users LEFT JOIN orders ON users.user_ID=orders.o_user_ID
    -> GROUP BY users.user_ID
    -> ORDER BY total DESC;
+-----------+-----------+---------------+-------+
| u_surname | u_name    | u_patronymic  | total |
+-----------+-----------+---------------+-------+
| Симонов   | Игорь     | Николаевич    |     2 |
| Иванов    | Александр | Валерьевич    |     1 |
| Корнеев   | Александр | Александрович |     1 |
| Кузнецов  | Максим    | Петрович      |     1 |
| Петров    | Анатолий  | Юрьевич       |     0 |
| Лосев     | Сергей    | Иванович      |     0 |
+-----------+-----------+---------------+-------+
6 rows in set (0.02 sec)
```

**Вложенный запрос**. Позволяет использовать результат, возвращаемый одним запросом, в другом запросе. Так как результат возвращает только оператор*select*, то в качестве вложенного запроса всегда выступает*SELECT*-запрос. В качестве внешнего запроса может выступать запрос с участием любого SQL-оператора:*select, insert, update, delete, create table*и др.

Пусть требуется вывести названия и цены товарных позиций из таблицы *books*для каталога «Базы данных» таблицы*catalogs*:

```
mysql> SELECT b_name,b_price FROM books
    -> WHERE b_cat_ID=(SELECT cat_ID FROM catalogs
    ->                 WHERE cat_name='Базы данных')
    -> ORDER BY b_price;
+-----------------------------------+---------+
| b_name                            | b_price |
+-----------------------------------+---------+
| Практикум по Access               |   87.00 |
| Базы данных. Разработка приложений|  189.00 |
| Раскрытие тайн SQL                |  200.00 |
| Базы данных                       |  326.00 |
+-----------------------------------+---------+
4 rows in set (0.03 sec)
```

Получить аналогичный результат можно при помощи многотабличного запроса, но имеется ряд задач, которые решаются только при помощи вложенных запросов. Вложенный запрос может применяться не только с условием *WHERE*, но и в конструкциях*DISTINCT, GROUP BY, ORDER BY, LIMIT*и т. д. Различают:

- вложенные запросы, возвращающие одно значение;
- вложенные запросы, возвращающие несколько строк.

В первом случае вложенный запрос возвращает скалярное значение или литерал, которое используется во внешнем запросе (подставляет результат на место своего выполнения). Например, необходимо определить название каталога, содержащего самую дорогую товарную позицию:

```
mysql> SELECT cat_name FROM catalogs
    -> WHERE cat_ID=(SELECT b_cat_ID FROM books
    -> WHERE b_price=(SELECT MAX(b_price) FROM books));
+-----------------+
| cat_name        |
+-----------------+
| Программирование |
+-----------------+
1 row in set (0.00 sec)
```

Наиболее часто вложенные запросы используются в операциях сравнения в условиях, которые задаются ключевыми словами *WHERE, HAVING*или*ON*.

Однако следующий вложенный запрос вернет ошибку:

```
mysql> SELECT cat_name FROM catalogs
    -> WHERE cat_ID=(SELECT b_cat_ID FROM books);
ERROR 1242 (21000): Subquery returns more than 1 row
```

Чтобы выбрать строки из таблицы *catalogs*, у которых первичный ключ совпадает с одним из значений, возвращаемых вложенным запросом, следует воспользоваться конструкцией*IN*:

```
mysql> SELECT cat_name FROM catalogs
    -> WHERE cat_ID IN (SELECT b_cat_ID FROM books GROUP BY b_cat_ID);
+------------------+
| cat_name         |
+------------------+
| Программирование |
| Интернет         |
| Базы данных      |
| Сети             |
| Мультимедиа      |
+------------------+
5 rows in set (0.17 sec)
```

Ключевое слово *ANY*может применяться с использованием любого оператора сравнения. Используется логика*ИЛИ*, т. е. достаточно, чтобы срабатывало хотя бы одно из многих условий. Запрос вида*WHERE X > ANY (SELECT Y ...)*можно интерпретировать как «где*X*больше хотя бы одного выбранного*Y*». Соответственно, запрос вида*WHERE X < ANY (SELECT Y ...)*интерпретируется как «где*X*меньше хотя бы одного выбранного*Y*». Рассмотрим запрос, возвращающий имена и фамилии покупателей, совершивших хотя бы одну покупку:

```
mysql> SELECT u_name, u_surname FROM users
    -> WHERE user_ID=ANY(SELECT o_user_ID FROM orders);
+----------+-----------+
| u_name   | u_surname |
+----------+-----------+
| Александр | Иванов    |
| Игорь    | Симонов   |
| Максим   | Кузнецов  |
| Александр | Корнеев   |
+----------+-----------+
4 rows in set (0.03 sec)
```

Ключевое слово *ALL*также может применяться с использованием любого оператора сравнения, но при этом используется логика*И*, то есть должны срабатывать все условия. Запрос вида*WHERE X > ALL (SELECT Y ...)*интерпретируется как «где*X*больше любого выбранного*Y*». Соответственно, запрос вида*WHERE X < ALL (SELECT Y ...)*интерпретируется как «где*X*меньше, чем все выбранные*Y*». Рассмотрим запрос, возвращающий все товарные позиции, цена которых превышает среднюю цену каждого из каталогов:

```
mysql> SELECT b_name, b_price FROM books
    -> WHERE b_price>ALL(SELECT AVG(b_price) FROM books
    -> GROUP BY b_cat_ID);
+----------------------------------+---------+
| b_name                           | b_price |
+----------------------------------+---------+
| Visual FoxPro 9.0                |  660.00 |
| Delphi. Полное руководство       |  500.00 |
| Совершенный код                  |  771.00 |
| Принципы маршрутизации в Internet |  428.00 |
| Компьютерные сети                |  630.00 |
| Сети. Поиск неисправностей       |  434.00 |
| Безопасность сетей               |  462.00 |
+----------------------------------+---------+
7 rows in set (0.03 sec)
```

Результирующая таблица, возвращаемая вложенным запросом, может не содержать ни одной строки. Для проверки этого факта могут использоваться ключевые слова *EXISTS*и *NOT EXISTS*.

Запрос, формирующий список покупателей, совершивших хотя бы одну покупку, можно записать следующим образом:

```
mysql> SELECT u_name, u_surname FROM users
    -> WHERE EXISTS (SELECT * FROM orders
    -> WHERE orders.o_user_ID=users.user_ID);
+-----------+-----------+
| u_name    | u_surname |
+-----------+-----------+
| Александр | Иванов    |
| Игорь     | Симонов   |
| Максим    | Кузнецов  |
| Александр | Корнеев   |
+-----------+-----------+
4 rows in set (0.00 sec)
```

**Практическая работа**

При выполнении лабораторной работы необходимо:

- для заданной предметной области построить многотабличный запрос на выборку с использованием объединения;
- для заданной предметной области построить запрос на выборку, содержащий вложенный запрос;
- составить отчет по лабораторной работе.

**Пример выполнения работы**

1. Создадим многотабличный запрос на выборку, который выводит фамилии, имена и отчества покупателей магазина, сделавших менее двух покупок:

```
mysql> SELECT users.u_surname,users.u_name,users.u_patronymic,
    -> COUNT(orders.order_ID) AS total
    -> FROM users LEFT JOIN orders ON users.user_ID=orders.o_user_ID
    -> GROUP BY users.user_ID
    -> HAVING total<2
    -> ORDER BY total DESC;
+-----------+-----------+---------------+-------+
| u_surname | u_name    | u_patronymic  | total |
+-----------+-----------+---------------+-------+
| Иванов    | Александр | Валерьевич    |     1 |
| Корнеев   | Александр | Александрович |     1 |
| Кузнецов  | Максим    | Петрович      |     1 |
| Петров    | Анатолий  | Юрьевич       |     0 |
| Лосев     | Сергей    | Иванович      |     0 |
+-----------+-----------+---------------+-------+
5 rows in set (0.02 sec)
```

2. Создадим запрос на выборку с вложенным запросом, выводящим перечень книг, которые не заказывались покупателями:

```
mysql> SELECT book_ID, b_name, b_price FROM books
    -> WHERE NOT EXISTS (SELECT * FROM orders
    -> WHERE orders.o_book_ID=books.book_ID);
+---------+-----------------------------------------+---------+
| book_ID | b_name                                  | b_price |
+---------+-----------------------------------------+---------+
|       1 | JavaScript в кармане                    |   42.00 |
|       2 | Visual FoxPro 9.0                       |  660.00 |
|       3 | C++ Как он есть                         |  218.00 |
|       4 | Создание приложений с помощью C#        |  169.00 |
|       5 | Delphi. Народные советы                 |  243.00 |
|       6 | Delphi. Полное руководство              |  500.00 |
|       7 | Профессиональное программирование на PHP |  309.00 |
|       9 | Практика программирования               |  214.00 |
|      11 | Поиск в Internet                        |  107.00 |
|      12 | Web-конструирование                     |  177.00 |
|      13 | Самоучитель Интернет                    |  121.00 |
|      14 | Популярные интернет-браузеры            |   82.00 |
|      15 | Общение в Интернете                     |   85.00 |
|      16 | Базы данных                             |  326.00 |
|      17 | Базы данных. Разработка приложений      |  189.00 |
|      18 | Раскрытие тайн SQL                      |  200.00 |
|      19 | Практикум по Access                     |   87.00 |
|      21 | Сети. Поиск неисправностей              |  434.00 |
|      22 | Безопасность сетей                      |  462.00 |
|      23 | Анализ и диагностика компьютерных сетей |  344.00 |
|      24 | Локальные вычислительные сети           |   82.00 |
|      25 | Цифровая фотография                     |  149.00 |
|      26 | Музыкальный компьютер для гитариста     |  217.00 |
|      27 | Видео на ПК                             |  231.00 |
|      28 | Мультипликация во Flash                 |  211.00 |
|      29 | Запись CD и DVD                         |  167.00 |
|      30 | Запись и обработка звука на компьютере  |   51.00 |
+---------+-----------------------------------------+---------+
27 rows in set (0.03 sec)
```

## Лабораторная работа №27 .

## Составление и выполнение запросов на модификацию данных.

Цель: изучить синтаксис конструкции инструкций SQL на модификацию данных.

Язык SQL ориентирован на выполнение операций над группами записей, хотя в некоторых случаях их можно проводить и над отдельной записью.

Запросы действия представляют собой достаточно мощное средство, так как позволяют оперировать не только отдельными строками, но и набором строк. С помощью запросов действия пользователь может добавить, удалить или обновить блоки данных. Существует три вида запросов действия:

·INSERT INTO – запрос добавления;

·DELETE – запрос удаления;

·UPDATE – запрос обновления.

Запрос добавления.

Оператор INSERT применяется для добавления записей в таблицу. Существует две формы оператора.

а) Первая форма оператора INSERT с параметромVALUES предназначена для вставки единственной строки в указанную таблицу.

Формат оператора:

INSERT INTO <имя_таблицы> [(имя_столбца [,...n]]

VALUES (значение[,...n])

Список столбцов указывает столбцы, которым будут присвоены значения в добавляемых записях. Список может быть опущен, тогда подразумеваются все столбцы таблицы (кроме объявленных как счетчик), причем в определенном порядке, установленном при создании таблицы. Если в операторе INSERT указывается конкретный список имен полей, то любые пропущенные в нем столбцы должны быть объявлены при создании таблицы как допускающие значениеNULL, за исключением тех случаев, когда при описании столбца использовался параметрDEFAULT. Список значений должен следующим образом соответствовать списку столбцов:

·количество элементов в обоих списках должно быть одинаковым;

·должно существовать прямое соответствие между позицией одного и то же элемента в обоих списках, поэтому первый элемент списка значений должен относиться к первому столбцу в списке столбцов, второй – ко второму столбцу и т.д.

·типы данных элементов в списке значений должны быть совместимы с типами данных соответствующих столбцов таблицы.

1) Добавить в таблицу «Предмет» новую запись:

INSERT INTO Предмет(Код_предмета, Имя_предмета, Часы)

VALUES("77", "Защита информации", "300")

2)Если столбцы таблицы «Предмет» указаны в том же составе и в том же порядке, в котором они перечислены при создании таблицы «Предмет», то оператор можно упростить:

INSERT INTO Предмет

VALUES("78","Информатика","350")

б) Вторая форма оператора INSERT с параметромSELECT позволяет скопировать множество строк из одной таблицы в другую.

Формат оператора:

INSERT INTO <имя_таблицы> [(имя_столбца [,...n]] <SELECT_оператор>

Предложение SELECT может представлять собой любой допустимый операторSELECT. Вставляемые в указанную таблицу строки в точности должны соответствовать строкам результирующей таблицы, созданной при выполнении вложенного запроса. Все ограничения, указанные выше для первой формы оператораSELECT, применимы и в этом случае.

Поскольку оператор SELECT в общем случае возвращает множество записей, то операторINSERT в такой форме приводит к добавлению в таблицу аналогичного числа новых записей.

3) Создать запрос на создание таблицы «Итоговая_ведомость»:

| Имя поля | Тип поля | Размер поля (Свойства поля) | Примечание |
|---|---|---|---|
| Номер_студбилета | Числовой | Целое (ключ) | Not null |
| Средняя_оценка | Числовой | С плавающей точкой (Float) | Not null |

4)Вставить в таблицу «Итоговая_ведомость» сведения о средней оценке каждого студента:

INSERT INTO Итоговая_ведомость ( Номер_студбилета, Средняя_оценка )

SELECT [Студент].[Номер_студбилета], Avg([Ведомость].[Оценка]) AS Средняя_оценка
FROM Студент INNER JOIN Ведомость ON
[Студент].[Номер_студбилета]=[Ведомость].[Номер_студбилета]

GROUP BY [Студент].[Номер_студбилета];

5)Добавить в таблицу «Итоговая_ведомость» столбцы «Фамилия», «Имя» и «Максимальная_оценка»

6)Удалить из таблицы «Итоговая_ведомость» все записи

7)Вставить в таблицу «Итоговая_ведомость» данные о фамилии и имени студента, его средней и максимальной оценке

Запрос удаления.

Оператор DELETE предназначен для удаления группы записей из таблицы.

Формат оператора:

<оператор_удаления>::=DELETE

FROM <имя_таблицы>

[WHERE <условие_отбора>]

Здесь параметр имя_таблицы представляет собой либо имя таблицы базы данных, либо имя удаляемого представления.

Если предложение WHERE присутствует, удаляются записи из таблицы, удовлетворяющие условию отбора. Если опустить предложениеWHERE, из таблицы будут удалены все записи, однако сама таблица сохранится.

8)Удалить те записи о студентах в таблице «Итоговая_ведомость», чьи средние оценки менее 5 баллов

DELETE

FROM ИТОГОВАЯ_ВЕДОМОСТЬ

WHERE Средняя_оценка<5:

9)Удалить всех студентов из таблицы «Итоговая_ведомость», чья группа ПО-06:

10)Удалить тех студентов из таблицы «Итоговая ведомость», у которых нет телефона.

11)Удалить все записи в таблице «Итоговая_ведомость»

Запрос обновления

Оператор UPDATE применяется для изменения значений в группе записей или в одной записи указанной таблицы.

Формат оператора:

UPDATE имя_таблицы SET имя_столбца=<выражение>[,...n]

[WHERE <условие_отбора>]

Параметр имя_таблицы – это либо имя таблицы базы данных, либо имя обновляемого представления. В предложении SET указываются имена одного и более столбцов, данные в которых необходимо изменить. ПредложениеWHERE является необязательным. Если оно опущено, значения указанных столбцов будут изменены во всех строках таблицы. Если предложениеWHERE присутствует, то обновлены будут только те строки, которые удовлетворяют условию отбора. Выражение представляет собой новое значение соответствующего столбца и должно быть совместимо с ним по типу данных.

12)Изменить количество часов на 400 для всех предметов, количество часов в которых >= 400:

UPDATE Предмет

SET Предмет.Часы=400

WHERE Предмет.Часы>=400

13)Заменить всем студентам с фамилией «Иванов» все полученные ими оценки на 2.

14)В таблице «Ведомость» всем студентам уменьшить оценки по информатике на 1 балл.

Введение в понятие "целостность данных"

Выполнение операторов модификации данных в таблицах базы данных INSERT, DELETE иUPDATE может привести к нарушению целостности данных и их корректности, т.е. к потере их достоверности и непротиворечивости.

Чтобы информация, хранящаяся в базе данных, была однозначной и непротиворечивой, в реляционной модели устанавливаются некоторые ограничительные условия – правила, определяющие возможные значения данных и обеспечивающие логическую основу для поддержания корректных значений. Ограничения целостности позволяют свести к минимуму ошибки, возникающие при обновлении и обработке данных.

В базе данных, построенной на реляционной модели, задается ряд правил целостности, которые, по сути, являются ограничениями для всех допустимых состояний базы данных и гарантируют корректность данных. Рассмотрим следующие типы ограничений целостности данных:

·обязательные данные;

·ограничения для доменов полей;

·корпоративные ограничения;

·целостность сущностей;

·ссылочная целостность.

Обязательные данные

Некоторые поля всегда должны содержать одно из допустимых значений, другими словами, эти поля не могут иметь пустого значения.

Ограничения для доменов полей

Каждое поле имеет свой домен, представляющий собой набор его допустимых значений.

Корпоративные ограничения целостности

Существует понятие "корпоративные ограничения целостности" как дополнительные правила поддержки целостности данных, определяемые пользователями, принятые на

предприятии или администраторами баз данных. Ограничения предприятия называются бизнес-правилами.

Целостность сущностей

Это ограничение целостности касается первичных ключей базовых таблиц. По определению, первичный ключ – минимальный идентификатор (одно или несколько полей), который используется для уникальной идентификации записей в таблице. Таким образом, никакое подмножество первичного ключа не может быть достаточным для уникальной идентификации записей.

Целостность сущностей определяет, что в базовой таблице ни одно поле первичного ключа не может содержать отсутствующих значений, обозначенных NULL.

Если допустить присутствие определителя NULL в любой части первичного ключа, это равносильно утверждению, что не все его поля необходимы для уникальной идентификации записей, и противоречит определению первичного ключа.

Ссылочная целостность

Указанное ограничение целостности касается внешних ключей. Внешний ключ – это поле (или множество полей) одной таблицы, являющееся ключом другой (или той же самой) таблицы. Внешние ключи используются для установления логических связей между таблицами. Связь устанавливается путем присвоения значений внешнего ключа одной таблицы значениям ключа другой.

Между двумя или более таблицами базы данных могут существовать отношения подчиненности, которые определяют, что для каждой записи главной таблицы (называемой еще родительской) может существовать одна или несколько записей в подчиненной таблице (называемой такжедочерней).

Существует три разновидности связи между таблицами базы данных:

·"один-ко-многим";

·"один-к-одному";

·"многие-ко-многим".

Отношение "один-ко-многим"имеет место, когда одной записи родительской таблицы может соответствовать несколько записей дочерней. Связь"один-ко-многим"иногда называют связью"многие-к-одному".И в том, и в другом случае сущность связи между таблицами остается неизменной.

Связь "один-ко-многим"наиболее распространена для реляционных баз данных. Она позволяет моделировать также иерархические структуры данных.

Отношение "один-к-одному"имеет место, когда одной записи в родительской таблице соответствует одна запись в дочерней. Это отношение встречается намного реже, чем отношение"один-ко-многим".Его используют, если не хотят, чтобы таблица БД "распухала" от второстепенной информации. Использование связи"один-к-одному"приводит к тому, что для чтения связанной информации в нескольких таблицах

приходится производить несколько операций чтения вместо одной, когда данные хранятся в одной таблице.

Отношение "многие-ко-многим"имеет место в следующих случаях:

·одной записи в родительской таблице соответствует более одной записи в дочерней таблице;

·одной записи в дочерней таблице соответствует более одной записи в родительской таблице.

Считается, что всякая связь "многие-ко-многим"может быть заменена на связь"один-ко-многим" (одну или несколько).

Часто связь между таблицами устанавливается по первичному ключу, т.е. значениям внешнего ключа одной таблицы присваиваются значения первичного ключа другой. Однако это не является обязательным – в общем случае связь может устанавливаться и с помощью вторичных ключей. Кроме того, при установлении связей между таблицами не требуется непременная уникальность ключа, обеспечивающего установление связи. Поля внешнего ключа не обязаны иметь те же имена, что и имена ключей, которым они соответствуют. Внешний ключ может ссылаться на свою собственную таблицу – в таком случае внешний ключ называется рекурсивным.

Ссылочная целостность определяет: если в таблице существует внешний ключ, то его значение должно либо соответствовать значению первичного ключа некоторой записи в базовой таблице, либо задаваться определителем NULL.

Существует несколько важных моментов, связанных с внешними ключами. Во-первых,следует проанализировать, допустимо ли использование во внешних ключах пустых значений. В общем случае, если участие дочерней таблицы в связи является обязательным, то рекомендуется запрещать применение пустых значений в соответствующем внешнем ключе. В то же время, если имеет место частичное участие дочерней таблицы в связи, то помещение пустых значений в поле внешнего ключа должно быть разрешено. Например, если в операции фиксации сделок некоторой торговой фирмы

необходимо указать покупателя, то поле КодКлиента должно иметь атрибутNOT NULL. Если допускается продажа или покупка товара без указания клиента, то для поля КодКлиента можно указать атрибутNULL.

Следующая проблема связана с организацией поддержки ссылочной целостности при выполнении операций модификации данных в базе. Здесь возможны следующие ситуации:

1.Вставка новой строки в дочернюю таблицу. Для обеспечения ссылочной целостности необходимо убедиться, что значение внешнего ключа новой строки дочерней таблицы равно пустому значению либо некоторому конкретному значению, присутствующему в поле первичного ключа одной из строк родительской таблицы.

2.Удаление строки из дочерней таблицы. Никаких нарушений ссылочной целостности не происходит.

3.Обновление внешнего ключа в строке дочерней таблицы. Этот случай подобен описанной выше первой ситуации. Для сохранения ссылочной целостности необходимо убедиться, что значение внешнего ключа в обновленной строке дочерней таблицы равно пустому значению либо некоторому конкретному

значению, присутствующему в поле первичного ключа одной из строк родительской таблицы.

4.Вставка строки в родительскую таблицу. Такая вставка не может вызвать нарушения ссылочной целостности. Добавленная строка просто становится родительским объектом, не имеющим дочерних объектов.

5.Удаление строки из родительской таблицы. Ссылочная целостность окажется нарушенной, если в дочерней таблице будут существовать строки, ссылающиеся на удаленную строку родительской таблицы. В этом случае может использоваться одна из следующих стратегий:

·NO ACTION. Удаление строки из родительской таблицы запрещается, если в

дочерней таблице существует хотя бы одна ссылающаяся на нее строка.

·CASCADE. При удалении строки из родительской таблицы автоматически

удаляются все ссылающиеся на нее строки дочерней таблицы. Если любая из удаляемых строк дочерней таблицы выступает в качестве родительской стороны в какой-либодругой связи, то операция удаления применяется ко всем строкам дочерней таблицы этой связи и т.д. Другими словами, удаление строки родительской таблицы автоматически распространяется на любые дочерние таблицы.

·SET NULL. При удалении строки из родительской таблицы во всех ссылающихся

на нее строках дочернего отношения в поле внешнего ключа, соответствующего первичному ключу удаленной строки, записывается пустое значение. Следовательно, удаление строк из родительской таблицы вызовет занесение пустого значения в соответствующее поле дочерней таблицы. Эта стратегия может использоваться, только когда в поле внешнего ключа дочерней таблицы разрешается помещать пустые значения.

·SET DEFAULT. При удалении строки из родительской таблицы в поле внешнего

ключа всех ссылающихся на нее строк дочерней таблицы автоматически помещается значение, указанное для этого поля как значение по умолчанию. Таким образом, удаление строки из родительской таблицы вызывает помещение принимаемого по умолчанию значения в поле внешнего ключа всех строк дочерней таблицы, ссылающихся на удаленную строку. Эта стратегия применима лишь в тех случаях, когда полю внешнего ключа дочерней таблицы назначено некоторое значение, принимаемое по умолчанию.

·NO CHECK. При удалении строки из родительской таблицы никаких действий по

сохранению ссылочной целостности данных не предпринимается.

6.Обновление первичного ключа в строке родительской таблицы. Если значение первичного ключа некоторой строки родительской таблицы будет обновлено, нарушение ссылочной целостности случится при том условии, что в дочернем отношении существуют строки, ссылающиеся на исходное значение первичного ключа. Для сохранения ссылочной целостности может применяться любая из описанных выше стратегий. При использовании стратегии CASCADE обновление значения первичного ключа в строке родительской таблицы будет отображено в любой строке дочерней таблицы, ссылающейся на данную строку.

Существует и другой вид целостности – смысловая (семантическая) целостность базы данных. Требование смысловой целостности определяет, что данные в базе данных должны изменяться таким образом, чтобы не нарушалась сложившаяся между ними смысловая связь.

Уровень поддержания целостности данных в разных системах существенно варьируется.

Идеология архитектуры клиент-сервертребует переноса максимально возможного числа правил целостности данных на сервер. К преимуществам такого подхода относятся:

·гарантия целостности базы данных, поскольку все правила сосредоточены в одном месте (в базе данных);

·автоматическое применение определенных на сервере ограничений целостности для любых приложений;

·отсутствие различных реализаций ограничений в разных клиентских приложениях, работающих с базой данных;

·быстрое срабатывание ограничений, поскольку они реализованы на сервере и, следовательно, нет необходимости посылать данные клиенту, увеличивая при этом сетевой трафик;

·доступность внесенных в ограничения на сервере изменений для всех клиентских приложений, работающих с базой данных, и отсутствие необходимости повторного распространения измененных приложений клиентов среди пользователей.

К недостаткам хранения ограничений целостности на сервере можно отнести:

·отсутствие у клиентского приложения возможности реагировать на некоторые ошибочные ситуации, возникающие на сервере при реализации тех или иных правил (например, ошибок при выполнении хранимых процедур на сервере);

·ограниченность возможностей языка SQL и языка хранимых процедур и триггеров для реализации всех возникающих потребностей определения целостности данных.

На практике в клиентских приложениях реализуют лишь такие правила, которые тяжело или невозможно реализовать с применением средств сервера. Все остальные ограничения целостности данных переносятся на сервер.

## Лабораторная работа №28.

## тема: Составление и выполнение вложенных запросов.

## 1.Цель:

- Изучить возможности SQL  по формулировке и обработке подзапросов.
- Приобрести практический опыт по формулировке и обработке подзапросов с использованием SQL.

## 2.Теоретические основы

***Запрос*** – это операция, которая позволяет отыскивать данные из одной или несколько таблиц. При наличии вложенных запросов запрос верхнего уровня называется предложением SELECT, а запрос, вложенный в предложение SELECT называется подзапросом**. Таким образом, *подзапрос*** (вложенный запрос) – это запрос, результат которого передается в качестве аргумента в другой запрос. Подзапросы позволяют связывать в единое целое несколько запросов.

Подзапросы используются для:

- определения множества строк, который должны быть вставлены в целевую таблицу в предложениях INSERT или CREATE TABLE;
- определения одного или более значений, присваиваемых существующим строка в предложении UPDATE;
- получения значений для фраз WHERE, HAVING или START WITH в предложениях SELECT, UPDATE, и DELETE;
- определения значений указанного столбца в списке INSERT ... VALUES;
- определения таблицы, которая используются соответствующим запросом.

Это производится путем размещения подзапроса во фразе FROM соответствующего

запроса как если бы это было именем таблицы. Вы можете также использовать таким образом подзапросы вместо таблиц в предложениях INSERT, UDPATE и DELETE.

Используемые таким образом подзапросы могут использовать переменные связывания (correlation variables), однако только такие, которые определены только в самом подзапросе, ссылки на внешние переменные не допустимы. Внешние ссылки (подзапросы с левой корреляцией - left-correlated subqueries) допустимы только во фразе FROM предложения SELECT .

Подзапрос дает ответ на содержательные запросы, имеющие сложную структуру. Например, для определения, кто работает на кафедре Иванова, вы сначала используете подзапрос для определения кафедры, на которой работает Иванов, а затем отвечаете на основной запрос путем формулировки предложения SELECT.

Подзапрос может содержать другие подзапросы. Oracle не ограничивает глубину вложенности подзапросов.

Если таблица в подзапросе имеет такое же имя, что и таблица внешнего запроса, то для ссылки на столбцы внешнего запроса их необходимо уточнять именем таблицы или алиасом таблицы. Чтобы ваши запросы было легче воспринимать, всегда квалифицируйте столбцы в подзапросе именем или алиасом таблицы.

Oracle выполняет ***корреляционные (связанный) подзапрос***, когда подзапрос ссылается на столбец таблицы внешнего запроса. Связанный подзапрос вычисляется для каждой строки, обрабатываемой внешним предложением. Внешним предложением может быть SELECT, UPDATE или DELETE.

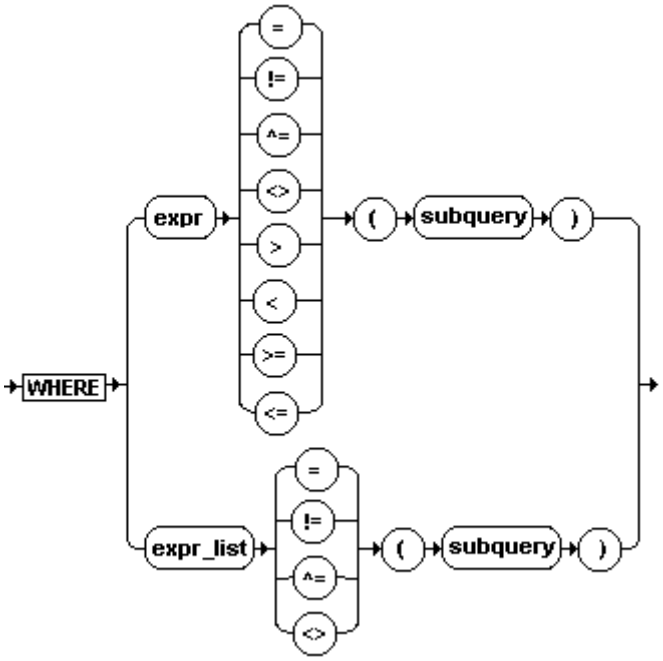
Связанный подзапрос дает ответы на такие содержательные запросы, ответы которых требуют вычисления подзапросов для каждой строки внешнего запроса. Например, связанный подзапрос используется для определения преподавателей, которые зарабатывают больше, чем средняя зарплата по кафедре. В этом случае связанный подзапрос для каждого преподавателя вычисляет среднюю зарплату на его кафедре.

Далее мы обсудим использование подзапросов в предложении SELECT.
^

## 2.1.Подзапрос во фразе WHERE

### 2.1.1.Подзапрос в простом условии сравнения

**Синтаксис:**




**Описание:**

При использовании простых условий сравнения с подзапросом во фразе WHERE применяются следующие правила:

- Подзапрос должен возвращать единственную строку.
- Если левая часть равна *expr*, то подзапрос должен возвращать единственную строку с единственным значением с типом, совместимым с типом *expr*.
- Если левая часть является списком выражений (*expr_list*), то подзапрос должен возвращать единственную строку со списком значений, который соответствует по количеству и типу значениям из *expr_list*. В этом случае оператор сравнения дает TRUE, если каждое значение в *expr_list* равно (в случае =) или не равно (в случае !=, ^=, <>) каждому значение, возвращаемому подзапросом.

**Примеры**:

1. Выбрать кафедры, которые располагаются в том же корпусе, что факультет информатики:

SELECT Name

^ FROM DEPARTMENT

WHERE Building = (SELECT Building

FROM FACULTY

WHERE UPPER(Name) = 'INFORMATICS');

2. Выбрать факультеты, чьи фонды меньше фонда кафедры CAD:

```
SELECT Name

FROM FACULTY

WHERE Fund < (SELECT Fund

^ FROM DEPARTMENT

WHERE UPPER(Name) = 'CAD');
```

3. Выбрать преподавателей, у которых salary + commission превышает более чем на 100 половину salary + commission преподавателя Bill:

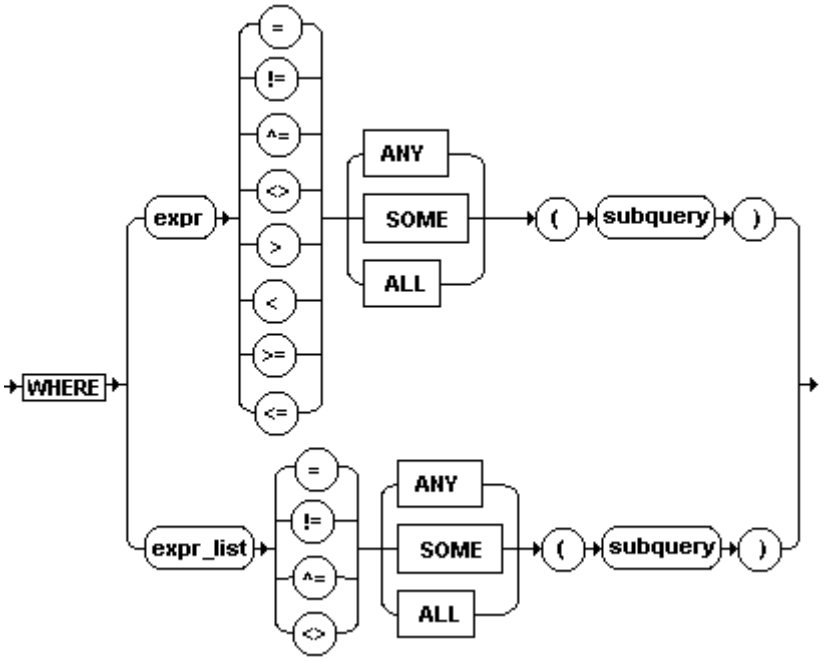
```
SELECT Name

^ FROM TEACHER

WHERE Salary + Commission + 100 > (SELECT (Salary + Commission) / 2

FROM TEACHER

WHERE UPPER(Name) = 'BILL');
```

4. Выбрать преподавателей, которые работают на той же кафедре, что и Bill и занимают ту же должность, что и Bill:

```
SELECT Name

FROM TEACHER

WHERE (DepNo, Post) = (SELECT DepNo, Post

FROM TEACHER

WHERE UPPER(Name) = 'BILL');
^
```

## 2.1.2.Подзапрос в условии сравнения групп

**Синтаксис:**




**Описание:**

При использовании условий сравнения групп с подзапросом во фразе WHERE применяются следующие правила:

- Подзапрос может возвращать ноль или более строк.
- Если левая часть равна *expr*, то подзапрос должен возвращать строки с единственным значением, которые совместимы по типу с *expr*.
- Если левая часть равна *expr_list*, то подзапрос должен возвращать строки со списком значений, который соответствует по количеству и типу с *expr_list*.

ANY и SOME эквивалентны и сравнивают значение слева с каждым значением списка справа, возвращаемого подзапросом. Подзапрос может вернуть ноль или более строк. Условие равно TRUE, если по крайней мере одна строка подзапроса удовлетворяет условию (соответствует оператору сравнения) по отношению к значению (списку значений) определенному левым операндом, в противном получаем FALSE. Если подзапрос не возвращает строк, то получаем FALSE.

ALL сравнивают значение слева с каждым значением списка справа, возвращаемого подзапросом. Дает TRUE, если ВСЕ строки, возвращаемые подзапросом, удовлетворяют условию (соответствуют оператору сравнения) по отношению к значению (списку значений) определенному левым операндом, в противном получаем FALSE. Если подзапрос не возвращает строк, то получаем TRUE

**Примеры:**

1. Выдать кафедры, фонд которых больше фонда по крайней мере одного из факультетов:

SELECT Name

^ FROM DEPARTMENT

WHERE Fund > ANY (SELECT Fund FROM FACULTY);

**ANY и агрегатные функции**. Обратите внимание, что *левое значение меньше, чем максимальное значение из множества, задаваемого правым операндом", а оператор >ANY эквивалентен следующему утверждению "левое значение больше, чем минимальное значение из множества, задаваемого правым операндом". Поэтому операторы ANY могут быть выражены через функции MAX и MIN в подзапросе.*

*2. Выдать кафедры, фонд которых больше фонда по крайней мере одного из факультетов:*

*SELECT Name*

*^ FROM DEPARTMENT*

*WHERE Fund > ANY (SELECT Fund FROM FACULTY);*

*SELECT Name*

*FROM DEPARTMENT*

*WHERE Fund > (SELECT MIN(Fund) FROM FACULTY);*

*3. Выдать группы, которые имеют рейтинг больше, чем рейтинг всех групп пятого курса кафедры "DBMS":*

*SELECT Num*

*FROM SGROUP*

*WHERE Rating >ALL (SELECT Rating*

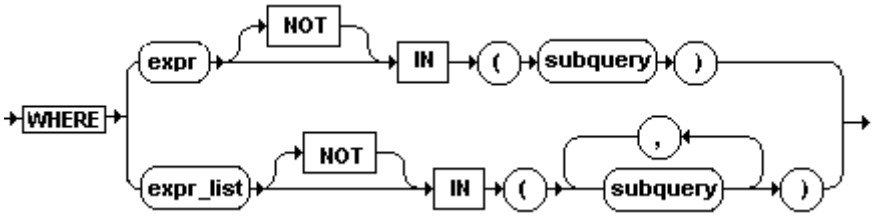
*FROM SGROUP, DEPARTMENT*

*WHERE SGROUP.DepNo = DEPARTMENT.DepNo AND*

*UPPER(DEPARTMENT.Name) = 'DBMS' AND SGROUP.Course = 5);*
*^*

## *2.1.3.Подзапрос в условии проверки вхождения элемента во множество*

***Синтаксис:***




***Описание:***

*Это условие в таком синтаксисе проверят вхождение элемента (списка элементов) во множество (множество списков), создаваемое подзапросом.*

***Пример:***

*1. Выбрать преподавателей, которые имеют лекции по крайней мере одному такому предмету, по которым читает лекции преподаватель Bill:*

*SELECT Name*

*FROM TEACHER T, LECTURE L*

*WHERE T.TchNo = L.TchNo AND*

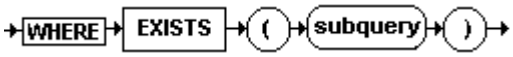
*SbjNo IN (SELECT SbjNo*

*FROM TEACHER TCH, LECTURE LEC*

*WHERE TCH.TchNo = LEC.TchNo AND UPPER(TCH.Name) = 'BILL');*
^

### *2.1.4.Подзапрос в условии EXISTS*

***Синтаксис:***




***Описание:***

*Дает TRUE, если подзапрос возвращает по крайней мере одну строку.*

*Так как EXISTS обычно используется в связанных подзапросах, мы его обсудим подробнее позже.*
^

## *2.2.Связанные подзапросы*

*Для того, чтобы связать подзапрос с внешним запросом (предложением), необходимо в подзапросе была ссылка на столбец внешнего запроса. Подзапрос вычисляется для каждой строки, обрабатываемой внешним запросом (предложением). В качестве*

*внешнего предложения могут выступать SELECT, UPDATE или DELETE.*

*Следующие примеры дают общий синтаксис использования связанных подзапросов:*

*SELECT select_list*

*FROM table1 t_alias1*

*WHERE expr operator*

*(SELECT column_list*

*FROM table2 t_alias2*

*WHERE t_alias1.column operator t_alias2.column);*

*UPDATE table1 t_alias1*

*SET column =*

*(SELECT expr*

*FROM table2 t_alias2*

*WHERE t_alias1.column = t_alias2.column);*

*DELETE FROM table1 t_alias1*

*WHERE column operator*

*(SELECT expr*

*FROM table2 t_alias2*

*WHERE t_alias1.column = t_alias2.column);*

*Далее мы обсудим использование связанных подзапросов во фразе WHERE предложения SELECT.*
^

### *2.2.1.Связанные подзапросы во фразе WHERE*

***Примеры:***

*1. Выдать преподавателей, которые имеют по крайней мере одну лекцию:*

*SELECT Name*

*FROM TEACHER*

*^ WHERE EXISTS (SELECT **

*FROM LECTURE*

*WHERE LECTURE.TchNo = TEACHER.TchNo);*

*Здесь в условии LECTURE.TchNo = TEACHER.TchNo подзапроса мы ссылаемся на внешний запрос. Поэтому подзапрос является связанным.*

*2. Выдать преподавателей, которые не имеют ни одной лекции:*

*SELECT Name*

*FROM TEACHER*

*WHERE NOT EXISTS (SELECT **

*FROM LECTURE*

*WHERE LECTURE.TchNo = TEACHER.TchNo);*
^

# *2.3.Простые и связанные подзапросы во фразе HAVING*

*Вы можете использовать простые и связанные подзапросы во фразе HAVING.*

*Если вы используете связанный подзапрос в фразе HAVING, то в подзапросе можно ссылаться на те столбцы внешнего запроса, которые могут использоваться в фразе HAVING (обычно это столбцы, по которым производится группирование).*

***Примеры:***

*1. Перечислить факультеты, у которых сумма фондов финансирования всех их кафедр превышает более чем на 20000 фонд финансирования той кафедры факультета, которая имеет максимальный фонд.*

*SELECT F1.Name*

*FROM FACULTY F1, DEPARTMENT D1*

*WHERE F1.FacNo = D1.FacNo*

*GROUP BY F1.Name*

*HAVING SUM(D1.Fund) > (SELECT 200000 + MAX(D2.Fund)*

*FROM FACULTY F2, DEPARTMENT D2*

*WHERE F2.FacNo = D2.FacNo AND F1.Name = F2.Name);*
^

## *2.4.Простые подзапросы во фразе FROM*

*Фраза FROM может содержать не только список имен таблиц, но и подзапросы. Для ссылки на такие таблицы-подзапросы следует приписать подзапросу алиас.*

*Имеется класс запросов, которые не могут быть выражены без подзапроса во фразе FROM. К ним, в частности, относятся такие запросы, который требуют независимого вычисления двух или более запросов, и затем совместного использования результатов такого запроса.*

***Пример:***

*Выдать средний фонд финансирования факультетов и среднюю зарплату преподавателей:*

*SELECT Fac.AvgFund, Tch.AvgSalary*

*FROM (SELECT AVG(Fund) AS AvgFund FROM FACULTY) Fac,*

*(SELECT AVG(Salary) AS AvgSalary FROM TEACHER) Tch;*

# *3.Задание*

*Сформулируйте предложения SQL SELECT, которые соответствуют следующим содержательным запросам:*

***Простые и вложенные запросы:***

1. *Выдать кафедры, которые располагаются в том же корпусе, что и факультет информатики*
2. *Выдать факультеты, которые имеют фонд финансирования меньше, чем фонд финансирования кафедры SE.*

***Одна и та же таблица во внешнем и вложенном запросе:***

3. *Выдать преподавателей, у которых Salary+Commission больше, чем половина Salary + Commission преподавателя Andrew.*

***Связанные запросы во фразе WHERE***

4. *Выдать факультеты, которые имеют кафедры в корпусе 5.*

5. *Выдать преподавателей, которые имеют менее 3 лекций на 1-й неделе.*
6. *Выдать корпуса, в которых располагается только один факультет.*

***EXISTS во фразе WHERE:***

7. *Выдать преподавателя, который имеет по крайней мере одну лекцию в понедельник первой недели.*
8. *Выдать преподавателей-профессоров, которые не являются кураторами групп первого курса.*

***ANY, SOME и ALL во фразе WHERE***

9. *Выдать преподавателей кафедры CAD, у которых ставка меньше по крайней мере одного из преподавателей кафедры SE (использовать оператор < ANY).*
10. *Выдать кафедры, у которых фонд меньше на 20000 фонда по крайней мере одного из факультетов (привести два варианта: с оператором*

11. *Выдать группы, у которых рейтинг больше, чем во всех группах пятого курса кафедры (привести два варианта: с оператором >ANY и с оператором EXISTS)*

***Агрегатные функции в подзапросе***

12. *Выдать преподавателей кафедры SE, у которых Salary+Commission больше, чем среднее Salary+Commission в вузе.*
13. *Выдать факультеты, у которых менее 3 кафедр.*
14. *Выдать преподавателей, которые имеют более 2 лекций на 1-й неделе.*

***Подзапросы во фразе HAVING***

15. *Выдать преподавателей кафедры SE, которые имеют больше лекций, чем любой из преподавателей кафедры DBMS.*

^

# *4.Контрольные вопросы*

*Ответьте на следующие вопросы:*

1. *В каких фразах предложения SELECT может использоваться подзапрос?*
2. *Что такое связанный подзапрос? Как подзапрос становится связанным? Как он вычисляется?*
3. *С какими операторами может использоваться подзапрос во фразе WHERE?*
4. *Какие правила использования подзапроса в простых условий сравнения?*
5. *Какие правила связывания подзапроса во фразе HAVING?*
6. *Приведите пример, когда запрос не может быть выражен иначе, чем использование подзапроса во фразе FROM.*

## Лабораторная работа № 29,30

## Тема: Создание отчетов по запросу к таблицам с использованием Компонента QUICKREPORT

***Цель работы:*** изучить и получить навыки использования компонентов, входящих в генератор отчетов Quick Report .

### Теоретические сведения

В первой части лабораторного практикума было показано, как получить доступ к информации, хранящейся в базе данных, и создать приложения для отображения и редактирования этой информации, используя удобный и понятный пользовательский интерфейс. Наряду с этим на практике требуется обеспечить вывод выбранной из базы данных информации не только в экранные формы, но и на печать.

Информацию, выводимую на печать или в файл и представленную в удобном для восприятия виде, называют отчетом. Создание отчетов является одной из основных функций, выполняемых приложениями, работающими с базами данных.

Для создания отчетов используются специальные компоненты, которые значительно облегчают эту задачу, выполняя все основные функции по форматированию, предварительному просмотру и выводу на печать информации из некоторого набора данных. Поэтому разработка отчетов обычно сводится к определению структуры и внешнего вида отчета.

Разработка отчета во многом схожа с разработкой формы для ввода данных, поскольку выполнение основных функций уже предусмотрено в соответствующих компонентах, которые требуется только соответствующим образом разместить и настроить. Многие компоненты, используемые при создании отчетов, подобны компонентам отображения данных, применяемым при разработке форм. Поэтому, приступая к разработке отчета, следует определить следующие его параметры [1]:

1. информацию, которая должна содержаться в отчете;
2. таблицы, содержащие необходимые данные;
3. внешний вид создаваемого отчета;
4. поля, по которым производится упорядочение и группировка данных в отчете;
5. содержание итоговой части отчета, если в ней есть необходимость;
6. дополнительную информацию, отображаемую в отчете: заголовки, поясняющие надписи, разделительные линии, рисунки и т.д.

Перед разработкой отчета целесообразно нарисовать эскиз отчета на бумаге, чтобы определить перечень компонентов, которые нужны для создания отчета.

Компоненты для создания отчетов позволяют формировать отчеты, которые условно можно разделить на две группы: табличные отчеты и отчеты в свободной форме.

В табличном отчете информация упорядочивается по строкам и столбцам. Такие отчеты фактически повторяют структуру таблиц базы данных или выборки таблиц.

Отчеты, выполненные в свободной форме, содержат информацию, располагающуюся в произвольной форме. Примером отчетов такого типа могут служить этикетки с почтовыми адресами для конвертов и письма, где поля таблицы базы данных должны размещаться в специально отведенных местах. При реализации отчетов в свободной форме данные также представляются в определенном порядке, но результат упорядочения не является таблицей.

Помимо значений полей из таблиц базы данных или вычисляемых полей отчет может содержать также и другие объекты, например, графики и диаграммы, рисунки, поясняющие надписи.

В системе для создания отчетов имеется набор компонентов, собранных на странице QReportпалитры компонентов и входящих в состав генератора отчетов QuickReport. Все компоненты генератора отчетов QuickReport можно разделить на четыре группы:

1. базовый компонент TQuickRep, являющийся контейнером для компонентов полос отчета и обеспечивающий генерацию и печать отчета;
2. полосы отчета - специальные компоненты, являющиеся контейнерами для элементов отображения данных и формирующие структуру отчета;
3. компоненты отображения данных, используемые для визуализации информации, содержащейся в таблицах базы данных, а также для вывода любой дополнительной информации;
4. фильтры - невизуальные компоненты, применяемые для экспорта отчета в файлы некоторых распространенных форматов.

Центральным компонентом отчета является TQuickRep, определяющий формирование отчета в целом. С помощью других компонентов создаются составные части отчета, например[2]:

- TQRBand - полоса для расположения данных, заголовков, титула отчета и пр. Отчет, в основном, строится из компонентов TQRBand, которые реализуют:
    - полосу заголовка отчета (Title);
    - полосу верхнего колонтитула (Page header);
    - полосу заголовка группы (Group header);
    - полосу названий столбцов отчета (Column header);

    - полосу детальной информации, предназначенную для отображения данных самого нижнего уровня детализации (Detail band);
    - полосу итогов для группы (Group footer);
    - полосу нижнего колонтитула (Page footer);
    - полосу итогов для отчета (Summary);
- TQRStringBand - имеет то же назначение, что и TQRBand. Отличается встроеннымм списком строкItems, содержимое которого становится видимым в режиме печати и предварительного просмотра, если на компонент TQRStringBand помещен компонентTQRExpr.Для каждой строки вItems выводится своя полосаTQRStringBand;
- TQRSubDetail -определяет полосу, в которой располагаются данные подчиненной таблицы при реализации в отчете связи"главная-подчиненная" для таблиц базы данных;
- TQRChildBand - дочерняя (подчиненная) полоса. Привязывается к родительской (главной) полосе и служит для ее расширения. Любая полоса может стать родительской, если ее свойствоHasChild задать равным True;
- TQRGroup- применяется для группировки данных в отчете;
- TQRLabel - позволяет разместить в отчете произвольную текстовую строку, например, название столбца таблицы в соответствующей полосе отчета;
- TQRDBText - служит для вывода в отчет содержимого поля таблицы базы данных;
- TQRExpr - применяется для вывода значений, являющихся результатом вычисления выражений. Алгоритм вычисления выражений строится при помощи редактора формул этого компонента;
- TQRSysData - служит для включения в отчет системной величины: даты, времени, номера страницы и т.п.;
- TQRDBImage- служит для вывода в отчете графической информации, хранящейся в столбце (поле) таблицы базы данных;
- TQRShape -служит для вывода в отчете графических фигур, например, прямоугольников;
- TQRPreview - базовый компонент для создания нестандартных окон предварительного просмотра отчета. Стандартное окно создается с помощью методаPreview компонента TQuickRep.

### *Компоненты генератора отчетов QuickReport*

***Компонент TQuickRep***, размещенный на форме, представляется сеткой отчета, в которую затем помещаются составные части отчета , например, полосы TQRBand. Основные свойствакомпонента TQuickRep приведены в табл.1.

Многие из этих свойств можно установить на этапе конструирования с помощью редактора свойств, если в локальном меню компонента TQuickRep выбрать пункт Report settings (рис.1).

Компонентом TQuickRep можно управлять с помощью имеющихся у него методов. В частности, метод Preview активизирует окно предварительного просмотра отчета (рис.2), а методPrint инициирует печать отчета на принтере. Чтобы на этапе конструирования просмотреть в окне предварительного просмотра содержимое отчета в том виде, как он будет выводиться на печать, нужно выббрать пунктPreview в локальном менюкомпонента TQuickRep. При этом некоторые данные (например, значения вычисляемых полей) отображаться не будут, поскольку они вычисляются только во время выполнения приложения.

Стандартное окно предварительного просмотра отчета не всегда удобно, поскольку его элементы управления невозможно изменить (например, русифицировать). Вместо этого окно можно использовать компонент TQRPreview, предназначенный для создания нестандартного окна просмотра. Чтобы отобразить отчет в нестандартном окне просмотра, нужнов форму с компонентом TQuickRepпоместить компонентTQRPreview, настроить его и предусмотреть его активизацию методом Show.

***Компоненты TQRBand***задают полосы отчета и используются для размещения на них таких компонентов, отображающих информацию, как TQRLabel, TQRDBText, TQRDBImage и т.п. Основные свойствакомпонента TQRBand приведены в табл.2.

Свойство BandType указывает назначение полосы, например:rbTitle - полоса содержит заголовок отчета;rbPageHeader - полоса содержит верхний колонтитул;rbDetail - полоса содержит информацию из набора данных, заданного свойством DataSetкомпонента TQuickRep(полоса выводится каждый раз при переходе на новую запись в наборе данных; эта полоса повторяется для всех записей набора данных, начиная с первой записи и кончая последней; позиционирование на первую запись и последовательный их перебор осуществляется компонентом TQuickRep автоматически).

Поместить компонентTQRBand в сетку отчета и указать назначение представляемой им полосы отчета можно двумя способами:

1. выбрать компонентTQRBand в палитре компонентов, разместить его на компоненте TQuickRep и выбрать значение свойства BandType, соответствующее назначению полосы;
2. в группе Bands окна установки параметров отчета (см. рис.1) пометить "галочкой"переключатель, соответствующий включаемой в отчет полосе. Все указанные таким способом полосы будут помещены в сетку отчета после закрытия окна.

КомпонентTQRBand может обрабатывать только событияAfterPrint и BeforePrint,которые наступают соответственно после окончания и перед началом печати или просмотра полосы отчета.

***Компонент TQRExpr*** позволяет поместить в отчет результат вычисления выражения, которое формируется с помощью редактора формул. Окно редактора формул (рис.3) активизируется кнопкой, имеющейся в поле свойстваExpression этого компонента, которое отображается в окне инспектора объектов.

Поле Enter expression служит для ввода и редактирования выражения, которое может состоять из имен полей (столбцов) таблиц, функций и переменных, связанных знаками операций, набираемыми с помощью среднего ряда кнопок (табл.3). Имя поля таблицы (Data field), функция (Function)или переменная(Variable) добавляется в выражение с помощью вспомогательного окна, которое появляется при нажатии на соответствующую кнопку верхнегоряда.

Компонент TQRExpr может использоваться для вычисления итогов для групп строк в таблице. Чтобы функция, заданная свойствомExpression, применялась отдельно к каждой группе, необходимо свойству ResetAfterPrint компонента TQRExpr присвоить значение True.

***Компонент TQRSysData*** используется для включения в отчет вспомогательной или системной информации, вид которой определяется свойством Data этого компонента. Возможные значения свойства Data указаны в табл.4.

***Компонент TQRGroup*** предназначен для группировки данных с целью подведения некоторых итогов для каждой группе. Он является потомком класса TQRBandи представляет собой полосуGroupHeader, которая печатается в начале каждой группы и предшествует в сетке отчета полосеDetail. Важнейшими свойствами компонента TQRGroup являются Expression и FooterBand.

Свойство Expression, задаваемое с помощью редактора формул (см. рис.3), используется как признак группировки данных и представляет собой выражение, по значению которого данные объединяются в группу. Если требуется показать в отчете значение этого выражения, то в полосеGroupHeader следует разместить компонент TQRExpr, у которого свойство Expression будет таким же, как у компонента TQRGroup. В качестве выражения можно использовать только индекс, имеющийся у таблицы, данные из которой группируются. Этот индекс должен быть определен в качестве текущего для компонента доступа к данным таблицы, указанного в свойствеDataSet компонента TQuickRep. Например, для компонента TTable текущий индекс определяется свойством IndexName или IndexFieldNames.

Чтобы итоги для группы отображались в отчете, помимо полосы GroupHeader в сетке отчета необходимо разместить полосу GroupFooter и расположить на ней компоненты TQRExpr для вычисления агрегатных (Statistical) функций: average - среднее значение для столбца (поля) или выражения, заданного в качестве аргумента; COUNT - количество строк (записей); Max -максимальное значение для столбца или выражения, заданного в качестве аргумента; MIN - минимальное значение для столбца или выражения, заданного в качестве аргумента; SUM - сумма значений для столбца или выражения, заданного в качестве аргумента.

Полоса GroupFooter образуется при размещении компонентаTQRBand в сетке отчета, если для этого компонента задать свойство BandTypeравным rbGroupFooter. Соответствие полос GroupHeader и GroupFooter для каждой группы задается свойством FooterBand компонента TQRGroup, которое ссылается на компонентTQRBand, являющийся полосой GroupFooter, содержащей итоги для определенной группы.

Задавая пары полос GroupHeader и GroupFooter, можно сформировать несколько вложенных друг в друга групп, если такая вложенность обеспечивается текущим индексом таблицы.

***Компонент TQRSubDetail*** предназначен для показа в отчете информации из подчиненной таблицы. Его свойство DataSetуказывает имя компонента доступа к подчиненной таблице, информация из которого будет выводиться в полосу, заданную компонентом TQRSubDetail. В остальном компонент TQRSubDetail аналогичен компонентуTQRBand, у которого свойствоBandType имеет значение rbDetail. Для строк подчиненной таблицы можно задать группировку с помощью компонента TQRGroup и соответствующего компонентаTQRBand, настроенного на представление полосы GroupFooter.

Если необходимо задать заголовок и итоги для информации, выводимой в полосе, представленной компонентом TQRSubDetail, то следует воспользоваться составным свойством Bands, в которое входят два логических свойстваHasHeader и HasFooter, указывающих на наличие или отсутствие заголовка и итогов соответственно.

***Компонент TQRShape*** можно использовать для отображения внутренних вертикальных линий, разделяющих столбцы таблицы, разместив его в соответствующем месте конкретной полосы отчета. Далее следует задать свойствуShapeкомпонента значение qrsVertLine (вертикальная линия) и установить в свойствеHeight высоту линии, равную высоте полосы, на которой размещен компонент TQRShape.

Чтобы таблица в отчете отображалась с разделительными линиями, обрамляющими полосы отчета, удобно использовать составноесвойство Frame, имеющееся у всех компонентов, представляющих полосы в сетке отчета, и содержащее 7 свойств: логические свойстваDrawBottom, DrawTop, DrawLeft, DrawRight, управляющие отображением нижней, верхней, левой и правой обрамляющих линий полосы соответственно; свойстваColor, Width и Style, управляющие цветом, шириной и типом линии. Все эти составляющие свойства обрамления доступны в инспекторе объектов.

## Технология создания отчетов

***Создание простейшего отчета.***Рассмотрим пример построения отчета, содержащего информацию о жителях из таблицыPerson учебной базы данных TUTOR_DATABASE, разработанной в первой части лабораторного практикума.

Разместим на форме компонент TTable, свяжем его с таблицейPerson.DB и откроем таблицу, задав для этогокомпонента, получившего имяTable1,значение свойстваActive равным True. Затем разместим на форме компонент TQuickRep и свяжем его с таблицей с помощью свойства DataSet.

Из локального меню компонента TQuickRep выберем пункт Report settings и в одноименном окне (см. рис.1) в группе элементов Bands зададим наличие в отчете полосы заголовка отчета(Title),полосы для детальной информации (Detail band) и полосы итогов для отчета (Summary), щелкнув мышью по соответствующим переключателям. После нажатия на кнопку ОК в компоненте TQuickRep появятся три компонентаTQRBand (TitleBand1, DetailBand1 иSummaryBand1).

Разместим в полосе отчета TitleBand1 компонент TQRLabel, установим в свойствеCaption этого компонента значение***Список жителей***и выберем в свойствеFont полужирный наклонный шрифт высотой 14.

Затем разместим в полосе отчета DetailBand1 шесть компонентов TQRDBTextдля вывода данных из шести полей текущей записи (строки) таблицы Person, содержащей шесть столбцов (Nom, FIO, RDate, Pol, SumD, Adr), и свяжем каждый компонент с соответствующим полем. Для этого в свойство DataSet каждого компонента TQRDBText установим значение Table1, а в свойство DataField - имя нужного поля.

И наконец, разместим в полосе отчета SummaryBand1 по два компонента TQRLabel и TQRExpr. Для первого компонента TQRLabel зададим значение свойства Caption равным ***Всего жителей:***, адля второго -***Сумма доходов:***. С помощью редактора формул для компонентов TQRExpr в свойстве Expression сформируем выражения для подведения итогов: для первого компонента TQRExprзададимагрегатную функцию COUNT, для второго -агрегатную функцию SUM(Table1.SumD).

На рис.4 показаны компоненты отчета с установленными свойствами.

Для просмотра получившегося отчета щелкнем по нему правой кнопкой мыши и из локального меню выберем пункт Preview. В результате откроется окно предварительного просмотра отчета, вид которого показан на рис.2. Первые три кнопки на инструментальной панели этого окна позволяют изменять масштаб отображения отчета, следующие четыре кнопки - последовательно показывать одну из страниц отчета. Далее первая группа из двух кнопок предназначена для печати отчета, вторая группа - для сохранения и загрузки отчета, а кнопка Closeзакрывает окно.

Чтобы в процессе выполнения приложения окно предварительного просмотра отчета открывалось при активизации формы, необходимо задать процедуру обработки события OnShowдля формы:

```
procedure TForm1.FormShow (Sender:TObject);

begin

QuickRep1.Preview;

end;
```

а чтобы после выхода из окна предварительного просмотра отчета закрывалась бы и форма, на которой создан отчет, нужно предусмотреть процедуру обработки события OnAfterPreview для компонента TQuickRep:

```
procedure TForm1.QuickRep1AfterPreview (Sender: TObject);

begin

Form1.Close;

end;
```

Данные, выводимые в отчете, всегда сортируются в соответствии с текущим индексом, который задается с помощью свойства IndexName компонентов доступа к данным (в рассматриваемом примере это компонентTable1). По умолчанию сортировка производится в соответствии с первичным ключом. Можно отсортировать строки отчета по фамилиям, если указать для свойства IndexName значение индекса PERSON03_FIO, который имеется у таблицыPerson.

***Создание отчета с группировкой данных.***В отчете, выводящем список жителей, сгруппируем жителей по адресам квартир и для каждой группы подсчитаем количество жителей квартиры и суммарную величину их общих доходов.

Для этого поместим в сетку созданного ранее простейшего отчета компонент TQRGroup. Появившаяся при этом полоса GroupHeader будет заголовком группы с именем QRGroup1. Размещенная на ней информация будет печататься каждый раз в начале новой группы. Оставим эту полосу незанятой и уменьшим ее высоту (свойство Height) до нуля, чтобы скрыть наличие этой полосы в отчете, поскольку в нашем случае информация об адресе квартиры будет печататься не в начале группы строк, а содержаться в самих строках.

Группировка строк таблицы по значению поля, хранящего адрес квартиры, указывается свойством Expression компонентаQRGroup1, являющегося заголовком группы. Значение

этого свойства с помощью редактора формул (см. рис.3) установим равным Table1.Adr, чтобы в группу включались строки с одинаковым значением поля Adr.

Строки таблицы перед группировкой должны быть отсортированы по полю, указанному в свойстве Expression компонентаQRGroup1, поэтому в компонентеTable1 в качестве текущего индекса таблицы укажем поле Adr.

Разместим в сетке отчета компонент TQRBand, который автоматически получит имя QRBand1, и свяжем его с заголовком группыQRGroup1, установив длязаголовка группысвойствоFooterBand равным QRBand1. В результате у компонента QRBand1 свойство BandTypeавтоматически станет равным rbGroupFooter, и в отчете появится полоса для итогов группы GroupFooter, связанная с заголовком группы.

На полосе GroupFooter разместим две пары компонентов TQRLabel и TQRExpr. Для первой пары зададим свойство Captionкомпонента TQRLabel равным ***Количество жителей***, а свойство Expression компонента TQRExpr равным функцииCOUNT;для второй пары компонентов зададим те же свойства равными***Общие доходы*** и функцииSUM(Table1.SumD). Свойству ResetAfterPrint обоихкомпонентов TQRExpr присвоим значение True, чтобы агрегатные функции, указанные в свойстве Expression, применялись отдельно к каждой группе

На рис.5 показаны компоненты отчета с установленными свойствами (на рисунке наличие полосы GroupHeader в отчете не скрыто).Для просмотра получившегося отчета выполним приложение,выбрав в главном меню системыDelphi команду Run | Run или нажав на клавиатуре клавишу F9.

***Создание отчета с информацией из главной и подчиненной таблиц.***Сформируем отчет, выводящий сведения о квартирах(из таблицыFlat учебной базы данных) и установленных в квартирах телефонах (из таблицыTPhone, которая подчинена главной таблицеFlat).

В форму нового проекта поместим два компонента TTable (с именами Table1 и Table2) и компонентыTDataSource и TQuickRep (с именами DataSource1 и QuickRep1).Настроимкомпоненты TTable, чтобыкомпонент Table1ассоциировался с таблицейFlat, а компонент Table2 -с таблицейTPhone, и установим между ними связь"главная-подчиненная" через компонент DataSource1, соединенный с Table1. Укажем в качестве основного источника информации для отчета таблицу Flat, задав свойство DataSet компонентаQuickRep1 равнымTable1, и откроем обе таблицы, установив для них значение свойства Active, равное True.

В сетке отчета (вкомпонентеQuickRep1) разместим полосу заголовка отчета(Title) и полосу детальной информации (Detail band). В полосе заголовка отчета расположим компонентTQRLabel с надписью ***Квартиры и телефоны***, а полосу детальной информации поместим четыре компонентаTQRDBText и свяжем их соответственно с полямиAdr, NRooms, Skv, KCategory из таблицы Flat. В этой же полосе разместим слева от каждого из компонентовTQRDBText по одному компонентуTQRLabel с надписями ***Адрес, Число комнат, Площадь, Категория***.

Поместим в сетку отчета компонентTQRSubDetail и свяжем его с подчиненной таблицей TPhone, установив его свойство DataSet равным Table2. В области компонентаTQRSubDetail расположим компонент TQRLabel с надписью ***Телефон*** и компонент TQRDBText, связанный с полем Ntel таблицыTPhone.

Компоненты отчета показаны на рис.6. В результирующем отчете (рис.7) для каждой строки таблицы Flat выводится информация из подчиненной ей строки таблицы TPhone; если подчиненной строки нет, то номер телефона в отчете отсутствует.

***Создание отчета в свободной форме.*** Отчет в свободной форме строится таким же способом, что и табличный отчет, но данные, выводимые в полосу детальной информации (Detail band), не упорядочиваются по столбцам, а располагаются произвольно. Отчеты в свободной форме обычно применяются для вывода каких-либо бланков, в которых информация не представлена в виде таблиц. Типичными примерами отчетов в свободной форме являются письма со стандартным текстом иэтикетки с почтовыми адресами для конвертов. Такие отчеты содержат только полосу детальной информации.

При создании отчета в свободной форме удобно использовать специальный шаблон QuickReport Labels.Воспользуемся этим шаблоном для печати этикеток с почтовыми адресами для конвертов с письмами, которые рассылаются жителям 8-го микрорайона.

Командой File | New Applicationглавного меню системыDelphiсоздадим новое приложение с формойForm1.Выберем команду File | Newглавного меню ишаблон QuickReport Labels в открывшемся окне хранилища объектов на закладке Forms.В результатесозданное приложение будет дополнено формойQRLabelForm с заготовкой отчета для печати этикеток, на которой размещены компонент TQuickRep (имя компонентаQuickRep1), содержащий полосу детальной информации (имя компонентаDetailBand1), компонент TQRLabel (имя компонентаQRLabel2) и компонент TTable (имя компонентаMasterTable).

Настроим компонент MasterTable, чтобы он был связан с таблицейPerson.DB учебной базы данных TUTOR_DATABASE,и откроем таблицу, задав для этогокомпонентазначение свойстваActive равным True. КомпонентQRLabel2 используем для размещения на этикетке надписи ***Адрес и получатель***.

На полосу DetailBand1 поместим два компонента TQRDBText и зададим для них значение свойстваDataSet равнымMasterTable, связав их таким образом с таблицей Person, а в свойстве DataField этих компонентов укажем поля Adr и FIO (рис.8).

Для вывода в отчет адресов и фамилий жителей только 8-го микрорайона воспользуемся процедурой обработки событияBeforePrint полосы DetailBand1. Этой процедуре передается параметр PrintBand, имеющий типBoolean. Изменяя значение этого параметра, можно управлять выводом полос (строк) в отчет. Если процедура присвоит параметру значениеFalse, то полоса не будет отображаться в отчете. Поэтому выбор строк таблицыPersonc информацией о жителях 8-го микрорайона можно задать одним оператором присваивания:

```
procedure TQRLabelForm.DetailBand1BeforePrint

(Sender: TQRCustomBand;var PrintBand: Boolean);

begin

PrintBand := Pos(', 8',MasterTable['Adr'])<>0; {Шаблон *,_8*}

end;
```

Эта процедура будет вызываться только во время выполнения программы, поэтому при предварительном просмотре из среды разработки системы Delphiв отчет будут включены все строки, независимо от содержащейся в них информации(рис.9).

Кроме этой процедуры для формы QRLabelForm и компонента QuickRep1 предусмотрим соответственно процедуры обработки событий OnShow и OnAfterPreview:

```
procedure TQRLabelForm.FormShow (Sender:TObject);

begin

QuickRep1.Preview;

end;

procedure TQRLabelForm.QuickRep1AfterPreview (Sender: TObject);

begin

QRLabelForm.Close;

end;
```

На форме Form1 разместим кнопку и зададим для нее процедуру обработки событияOnClick, которая будет показывать форму QRLabelForm:

```
procedure TForm1.Button1Click(Sender: TObject);

begin

QRLabelForm.ShowModal;

end;
```

Выберем в главном меню команду File | Use Unit,чтобы к формеForm1 подключить модуль Unit2, в котором определено имя формы QRLabelForm.

Выполним созданное приложение и убедимся в формировании отчета, состоящего из набора этикеток, в соответствием с заданным условием (рис.10).

## Лабораторное задание и порядок выполнения работы

1. Ознакомиться с назначением компонентов генератора отчетов QuickReport и технологией создания отчетов; освоить технологию, создав отчеты, рассмотренные в лабораторной работе.
2. Воспользовавшись созданным простейшим отчетом, разработать приложение, которое формирует отчет с таблицей Person, в котором выводятся заголовки столбцов на русском языке (Номер, ФИО, Дата рожд. и т.д.), а строки отсортированы по адресам квартир. Предусмотреть обрамления строк и столбцов отчета линиями.

3. Изучить назначение параметров, устанавливаемых с помощью редактора свойств компонента TQuickRep, и разработать приложение, которое формирует отчет с таблицей Person, в котором выводятся верхний и нижний колонтитулы, номера страниц, дата и время формирования отчета.
4. Разработать приложение, которое формирует трехколоночный отчет для таблицы Person, в котором выводятся только адреса квартир, количество жителей и среднедушевой доход жителей каждой квартиры (использовать компонент TQRGroup и индексирование таблицы по адресу квартиры; полосу детальной информации сделать невидимой или пустой, но сохранить в сетке отчета).
5. Разработать приложение, которое формирует отчет с таблицами Flat и Person, в котором выводятся сведения о квартире и жителях, зарегистрированных в ней (использовать компонент TQRSubDetail и соединение таблиц как главной и подчиненной).
6. Модифицировать прикладную систему, спроектированную в последней работе первой части лабораторного практикума в соответствии с конкретным вариантом задания, реализовав в ней процедуры формирования отчетов, связанные с пунктами меню"Отчеты".
7. Оформить отчет по результатам выполнения лабораторной работы.

## Требования к отчету

Отчет должен содержать:

1. название и цель работы;
2. сведения о компонентах, используемых для формирования отчетов;
3. сведения о настройке компонентов, использованных в разработанных приложениях;
4. эскизы отчетов, формируемых модифицированной прикладной системой, спроектированной в последней работе первой части лабораторного практикума для конкретного варианта задания, и схему реализованных прикладных процессов выдачи отчетов.

**Практические работа по дисциплине ОП.05**

**«Основы программирования и баз данных»**

## Практическая работа №1

**Тема:** Составление алгоритмов и программ с использованием массивов и строк.

**Цель:** формирование знаний и умений по работе с блок-схемами алгоритмов для составления программ.

Пояснения.

Понятие алгоритма в программировании является фундаментальным. Для алгоритма важен не только набор определенных действий, но и то, как они организованы, т.е. в каком порядке они выполняются. Это одно из общих свойств алгоритма. Другое общее свойство алгоритма состоит в том, что каждое последующее действие выполняется лишь после завершения предшествующего.

### Свойства алгоритма

При составлении и записи алгоритма необходимо обеспечить, чтобы он обладал рядом свойств. Рассмотрим эти свойства на следующем примере: пусть требуется вычислить сумму двух целых чисел и вывести на экран результат.

***Однозначность алгоритма,*** под которой понимается единственность толкования исполнителем правил выполнения действий и порядка их выполнения. Чтобы алгоритм обладал этим свойством, он должен быть записан командами из системы команд исполнителя.

Для нашего примера исполнитель алгоритма должен понимать такую запись действий, как сложить числа A и B.

***Конечность алгоритма*** - обязательность завершения каждого из действий, составляющих алгоритм, и завершимость выполнения алгоритма в целом, т.е. алгоритм должен заканчиваться после конечного числа шагов.

***Результативность алгоритма,*** предполагающая, что выполнение алгоритма должно завершиться получением определенных результатов. Алгоритм в нашем примере обладает этим свойством, так как для целых чисел A и B всегда будет вычислена сумма.

***Массовость,*** т. е. возможность применения данного алгоритма для решения целого класса задач, отвечающих общей постановке задачи. Для того чтобы

алгоритм обладал свойством массовости, следует составлять алгоритм, используя обозначения величин и избегая конкретных значений.

***Правильность алгоритма,*** под которой понимается способность алгоритма давать правильные результаты решения поставленных задач. Представленный в примере алгоритм обладает свойством правильности, так как в нем использована правильная формула сложения целых чисел, и для любой пары целых чисел результат выполнения алгоритма будет равен их сумме.

***Определенность алгоритма***. Каждый шаг алгоритма должен быть определен**.**

***Входные данные алгоритма.*** Алгоритм должен иметь некоторое (может быть равное 0) число входных данных.

***Выходные данные алгоритма.*** Результатом выполнения алгоритма должна быть одна или несколько выходных величин, зависящих от исходных данных.

***Эффективность алгоритма***. Алгоритм должен быть эффективным, т.е. результат должен быть получен наименьшим числом наиболее простых операций.

Основные блоки, используемые при составлении алгоритмов

Название

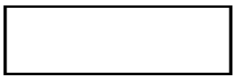
Обозначение

Назначение

Пуск, Останов

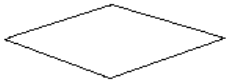
Начало-конец алгоритма

Процесс



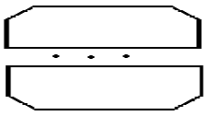
Любое вычислительное действие

Решение



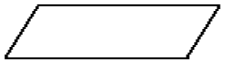
Проверка условия

Модификатор



Цикл

Ввод-вывод



Ввод-вывод данных

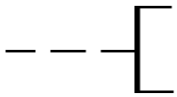
Документ

Вывод на печатающее устройство

Соединитель

Используется на линиях разрыва

Комментарий



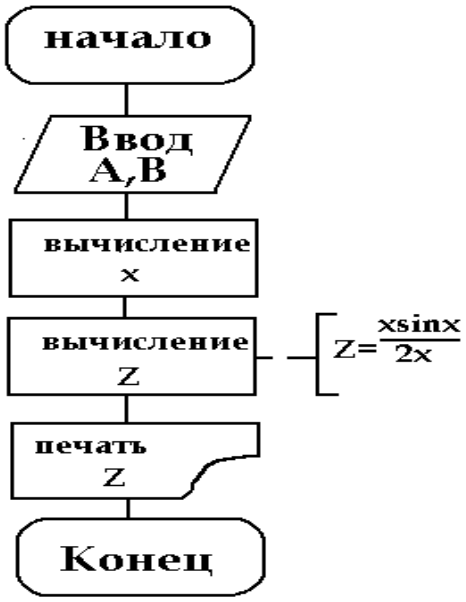
Комментарий

*Пример.* Составить схему алгоритма работы программы вычисления значения выражения:

Для начала для построения блок –схемы алгоритма опишем последовательность действий, необходимых для решения данной задачи:

- начало
- ввод чисел a, b
- вычисление x
- вычисление z
- вывод результата
- конец

Исходя из этого составляем блок-схему алгоритма согласно ГОСТ, используя соответствующие блоки.




Необходимые принадлежности.

1. Персональный компьютер с программным обеспечением.

2. Справочная литература.

Работа в аудитории.

1. Изучить теоретические сведения по теме ”Построение блок-схем алгоритмов”.
2. Решить с использованием персонального компьютера и программного обеспечения задания для практической работы.
3. Ответить письменно на контрольные вопросы.

Задание.

1) Составить программу для определения суммы цифр заданного трехзначного числа.

2) Составить программу, в которой для заданного q вычисляется один из корней уравнения Ln(ctgx-1)=q.

3) Вычислить периметр, площадь и углы прямоугольного треугольника по заданным длинам катетов.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Алгоритм программы;
6. Заключение

Контрольные вопросы.

1. Свойства алгоритма.
2. Блок схемы. Понятие и правила построения.
3. Примеры построения блок-схем алгоритмов.

## Практическое занятие №2

**Тема:** Составление алгоритма работы программы с использованием ветвления.

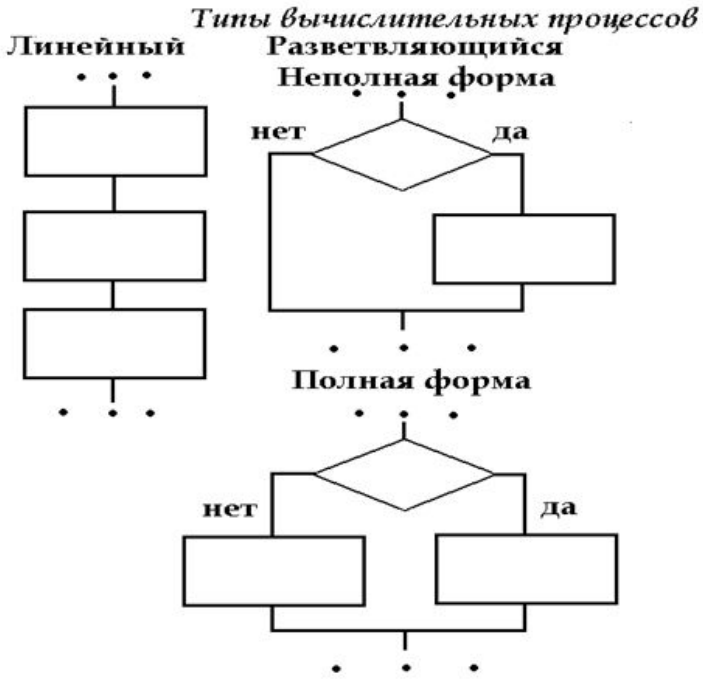
**Цель:** формирование знаний и умений по работе с блок-схемами алгоритмов для составления программ с использованием ветвления.

Пояснения.

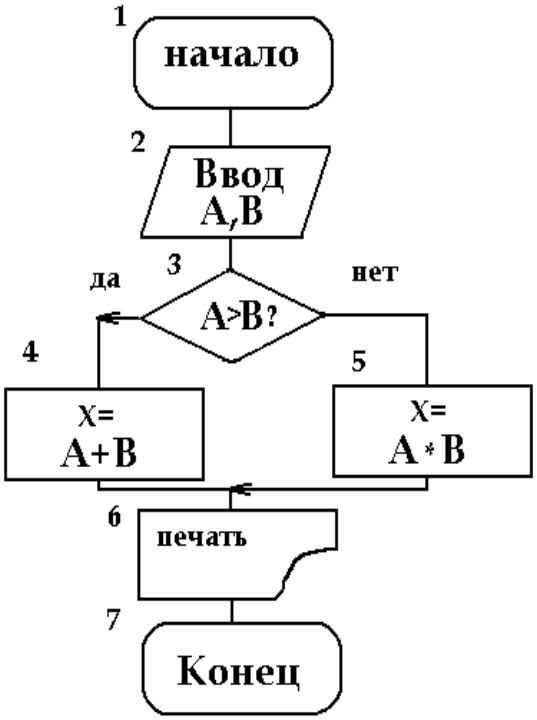
*Линейные алгоритмы* - это алгоритм, в котором все его действия выполняются одно за другим, т.е. последовательно.

Однако в большинстве вычислительных процессов мы сталкиваемся с тем, что выбор хода дальнейших действий определяется результатом предыдущих. Такие алгоритмы называются разветвляющимися.

*Разветвляющиеся алгоритмы* - это алгоритмы, в которых в зависимости от выполнения или не выполнения некоторого условия совершается одна или другая последовательность действий.




Пример. Составить схему алгоритма вычисления значения: x=a+b при a>b, x=a*b, при a<=b.




Необходимые принадлежности.

1. Персональный компьютер с программным обеспечением.

2. Справочная литература.

Работа в аудитории.

4. Изучить теоретические сведения по теме ”Построение блок-схем алгоритмов”.
5. Решить с использованием персонального компьютера и программного обеспечения задания для практической работы.
6. Ответить письменно на контрольные вопросы.

Задание.

Составить алгоритм работы программы решения уравнения $ax^2+b=0$ (неравенства $ax^2+b > 0$) 3 способами: словарным описанием, блок-схемой и на языке псевдокода. При отсутствии решения или бесчисленном множестве решений должен быть напечатан соответствующий текст.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Алгоритм программы;
6. Заключение

Контрольные вопросы.

1. Определение линейного алгоритма.
2. Определение разветвляющегося алгоритма.
3. Понятие псевдокода. Объяснить достоинства и недостатки псевдокода.

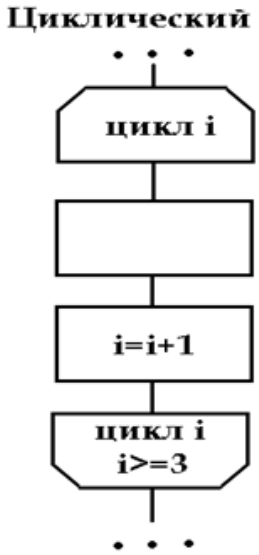
## Практическое занятие №3

**Тема:** Составление алгоритма работы программы с использованием цикла.

**Цель:** формирование знаний и умений по работе с блок-схемами алгоритмов для составления программ с использованием цикла.

Пояснения.

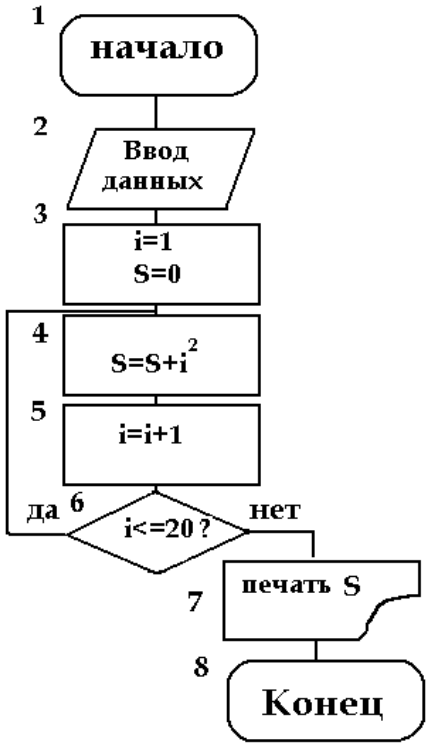
*Циклические алгоритмы*- алгоритмы, в которых одна и та же последовательность действий совершается несколько раз до тех пор, пока выполняются некоторые условия. На рисунке представлено графическое представление вычислительных процессов




Составить схему алгоритма вычисления значения:

Для начала для построения блок – схемы алгоритма опишем последовательность действий, необходимых для решения данной задачи:

Исходя из этого составляем блок-схему алгоритма согласно ГОСТ, используя соответствующие блоки.



Необходимые принадлежности.

1. Персональный компьютер с программным обеспечением.

2. Справочная литература.

Работа в аудитории.

1. Изучить теоретические сведения по теме ”Построение блок-схем алгоритмов”.
2. Решить с использованием персонального компьютера и программного обеспечения задания для практической работы.
3. Ответить письменно на контрольные вопросы.

Задание.

Составить алгоритм работы программы нахождения делителей заданного числа N 3 способами: словарным описанием, блок-схемой и на языке псевдокода.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Алгоритм программы;
6. Заключение

Контрольные вопросы.

1. Определение циклических алгоритмов.

2. Типы циклических алгоритмов.

3. В чём состоят сложности при написании псевдокода циклических алгоритмов.

## Практическое занятие №4

Тема: «Изучение интегрированной среды программирования»

Цель: Освоение интегрированной среды программирования, использование для разработки и отладки программ.

Пояснения.

Можно создавать файлы с исходным кодом на C# с помощью обычного текстового редактора, например, Блокнота, и компилировать их в управляемые

модули с помощью компилятора командной строки, который является составной частью .NET Framework. Однако наиболее удобно для этих целей использовать VS, потому что:

1. VS автоматически выполняет все шаги, необходимые для компиляции исходного кода.
2. Текстовый редактор VS настроен для работы с теми языками, которые поддерживаются VS, например C#, поэтому он может интеллектуально обнаруживать ошибки и подсказывать в процессе ввода, какой именно код необходим.
3. В состав VS входят программы, позволяющие создавать Windows- и Web-приложения путем простого перетаскивания мышью элементов пользовательского интерфейса.
4. Многие типы проектов, создание которых возможно на C#, могут разрабатываться на основе "каркасного" кода, заранее включаемого в программу. Вместо того чтобы каждый раз начинать с нуля, VS позволяет использовать уже имеющиеся файлы с исходным кодом, что уменьшает временные затраты на создание проекта.

**Создание первого проекта**

Для создания проекта следует запустить VS, а затем в главном меню VS выбрать команду ***File – New - Project*** . После чего откроется диалоговое меню ***New Project*** (см. рис.1).

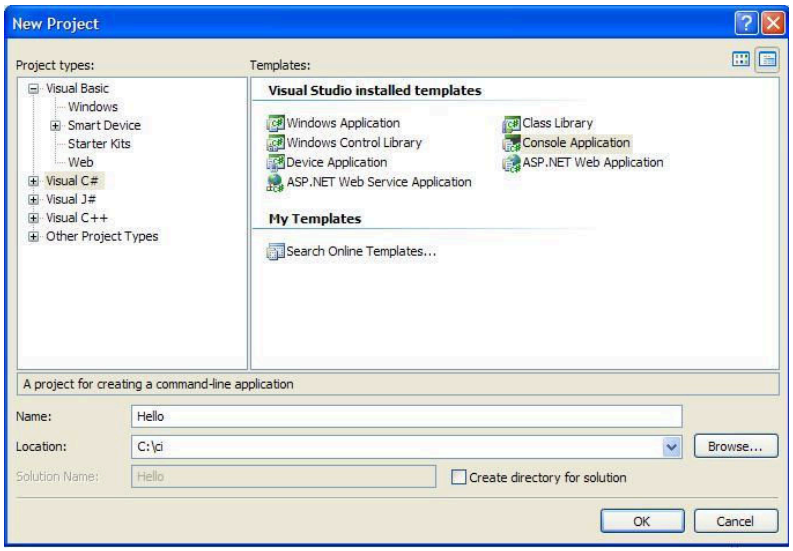



Рис.1.

В поле ***Project types*** следует выбрать *Visual C#*, в поле ***Templates*** – *Console Application*.

В строчке ***Name*** введите имя приложения *Hello*. Обратите внимание на то, что это же имя появится в строчке ***Solution Name***. Уберите галочку в поле ***Create***

***directory for Application*** (пока мы создаем простое приложение, и нам нет необходимости усложнять его структуру).

В строке ***Location*** определите положение на диске, куда нужно сохранять ваш проект. И нажмите кнопку ***OK***. Примерный вид экрана изображен на рис 2.

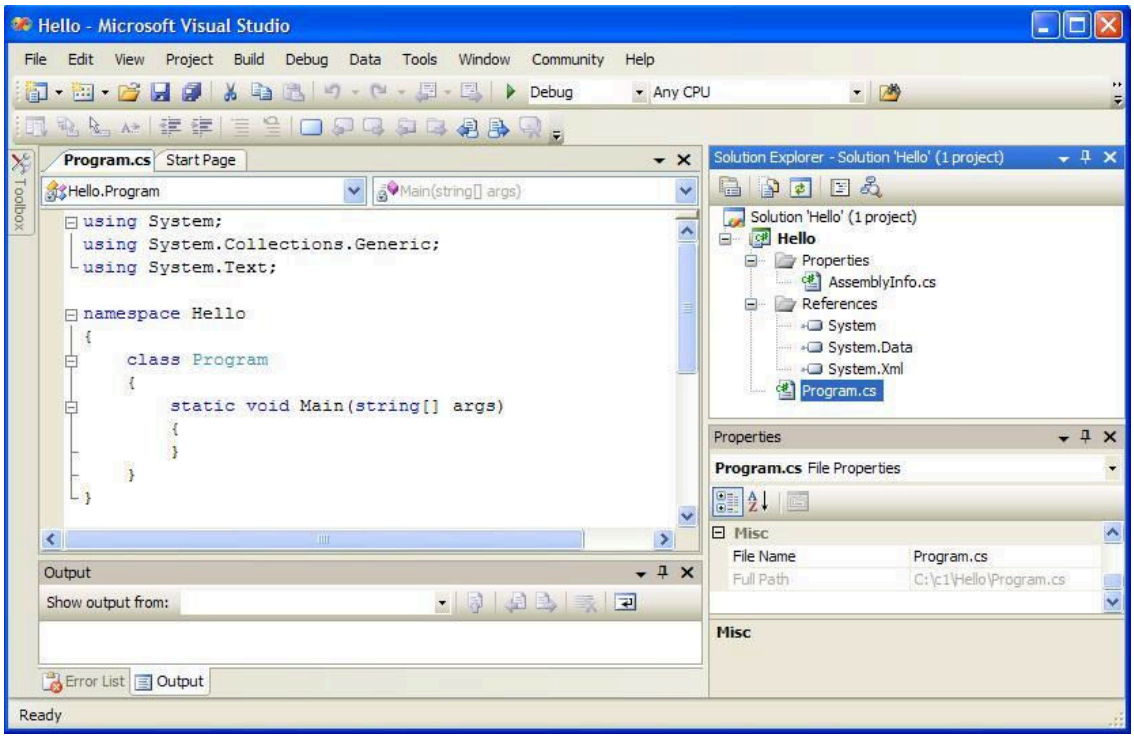



Рис. 2.

В правой верхней части располагается окно управления проектом ***Solution Explorer***. Если оно закрыто, то его можно включить командой ***View - Solution Explorer***. В этом окне перечислены все ресурсы, входящие в проект:

1) AssemblyInfo.cs – информация о *сборке*.

Компилятор в качестве результата своего выполнения создает так называемую *сборку* – файл с расширением exe или dll, который содержит IL-код и метаданные.

2. System, System.Data, System.Xml – ссылки на стандартные библиотеки.
3. Program.cs - текст программы на языке C#.

***Замечание***. В других версиях VS сюда же включается файл с расширением ico, отвечающий за вид ярлыка приложения.

В правой нижней части экрана располагается окно свойств ***Properties***. Если оно закрыто, то его можно включить командой ***View - Properties***. В этом окне отображаются важнейшие характеристики выделенного элемента.

Основное пространство экрана занимает окно редактора, в котором располагается текст программы, созданный средой автоматически. Текст представляет собой каркас, в который программист будет добавлять нужный код. При этом зарезервированные слова отображаются синим цветом, комментарии – зеленым, основной текст – черным.

Текст структурирован. Щелкнув на знак минус, мы скроем блок кода, щелкнув на знаке плюс – откроем.

Откроем папку, содержащую проект, и рассмотрим ее структуру. Файлы, выделенные жирным шрифтом, появятся только после компиляции.

На данном этапе особый интерес для нас будут представлять следующие файлы:

1. *Hello.sln* – основной файл, отвечающий за весь проект. Если необходимо открыть проект для редактирования, то нужно выбрать именно этот файл. Остальные файлы откроются автоматически.
2. *Program.cs* – файл, в котором содержится исходный код - код, написанный на языке C#. Именно с этим файлом мы и будем непосредственно работать.
3. *Hello.exe* – файл, в котором содержатся сгенерированный IL-код и метаданные проекта. Другими словами, этот файл и есть готовое приложение, которое может выполняться на любом компьютере, на котором установлена платформа .Net.

Теперь рассмотрим сам текст программы.

***using System*** – это директива, которая разрешает использовать имена стандартных классов из пространства имен ***System*** непосредственно без указания имени пространства, в котором они были определены.

Ключевое слово ***namespace*** создает для проекта свое собственное пространство имен, которое по умолчанию называется именем проекта. В нашем случае пространство имен называется Hello. Однако программист вправе указать другое имя. Пространство имен ограничивает область применения имен, делая его осмысленным только в рамках данного пространства. Это сделано для того, чтобы можно было давать имена программным объектам, не заботясь о том, что они совпадут с именами в других приложениях. Таким образом, пространства имен позволяют избегать конфликта имен программных объектов, что особенно важно при взаимодействии приложений.

C# - объектно-ориентированный язык, поэтому написанная на нем программа будет представлять собой совокупность взаимодействующих между собой классов. Автоматически был создан класс с именем ***Program*** (в других версиях среды может создаваться класс с именем ***Class1***).

Данный класс содержит только один метод - метод ***Main()***. Метод Main() является точкой входа в программу, т.е. именно с данного метода начнется выполнение приложения. Каждая программа на языке C# должна иметь метод Main ().

***Замечание***. Технически возможно иметь несколько методов Main() в одной программе, в этом случае потребуется с помощью параметра командной строки сообщить компилятору C#, какой именно метод Main() является точкой входа в программу.

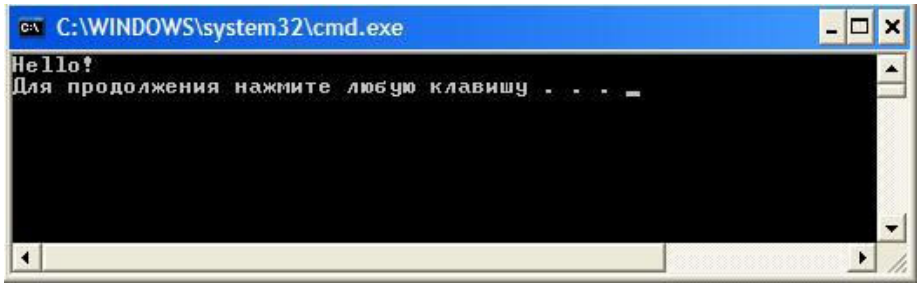
Метод Main() имеет одну важную особенность. Перед объявлением типа возвращаемого значения ***void*** (который означает, что метод не возвращает значение) стоит ключевое слово ***static***, которое означает что метод Main() можно вызывать, не создавая объект типа ***Program***.

***Замечание***. В некоторых версиях требуется, чтобы перед словом ***static*** стояло слово ***public***.

Добавим в метод следующий код: ***Console.WriteLine("Hello!");***

Здесь *Console* имя стандартного класса из пространства имен System. Его метод *WriteLine* выводит на экран текст, заданный в кавычках

Для запуска программы следует нажать клавишу F5 или выполнить команду ***Debug-Start Debugging.*** Если программа выполнена без ошибок, то сообщение выведется в консольное окно, которое мелькнет и быстро закроется. Чтобы просмотреть сообщение в нормальном режиме нужно нажать клавиши Ctrl+F5 или выполнить команду ***Debug-Start Without Debugging.*** В нашем случае откроется следующее консольное окно:




Если код программы будет содержать ошибки, например, пропущена точка с запятой после команды вывода, то после нажатия клавиши F5 откроется диалоговое окно, в котором выведется сообщение о том, что обнаружена ошибка, и вопрос, продолжать ли работу дальше. Если вы ответите ***Yes***, то будет выполнена предыдущая удачно скомпилированная версия программы. Иначе процесс будет остановлен и управление передано окну списка ошибок ***Error List***.

## Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

## Работа в аудитории.

1. Запустить интегральную среду разработки.

2. Выполнить задание.

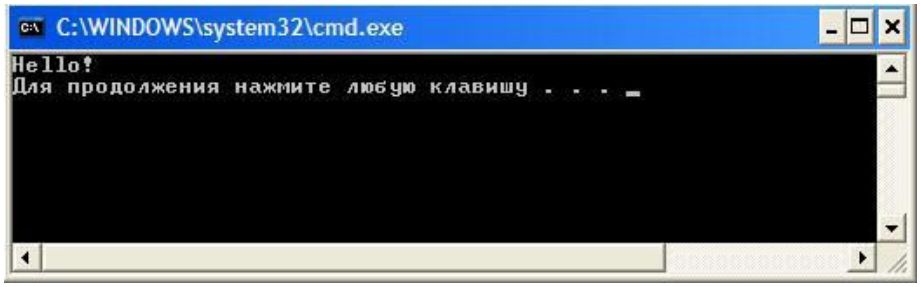
3. Продемонстрировать работоспособное приложение преподавателю.

## Задание

1. Ознакомиться с меню и интерфейсом интегрированной среды программирования.

2. Создать консольный проект с именем MyFirstProgram.

3. Выполнить реализацию приложения согласно образцу:




## Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Исходный код приложения;
6. Заключение.

## Контрольные вопросы

1. Назначение ИСП.
2. В чём удобство использования ИСП.
3. Какие типы проектов существуют в ИСП.
4. Структура проекта. Назначение основных файлов.
5. Запуск отладки приложения.

## Практическое занятие № 5

## Тема: «Составление требований к программному продукту»

**Цель:** научатся применять основные шаги и техники в процессе разработки требований от дополнительных и взаимозаменяемых, а так же отличать обязательные требования от необязательных.

Пояснения.

Требования к программному обеспечению — совокупность утверждений относительно атрибутов, свойств или качеств программной системы, подлежащей реализации. Создаются в процессе разработки требований к программному обеспечению, в результате анализа требований.

Требования могут выражаться в виде текстовых утверждений и графических моделей.

В классическом техническом подходе совокупность требований используется на стадии проектирования ПО. Требования также используются в процессе проверки ПО, так как тесты основываются на определённых требованиях.

Этапу разработки требований, возможно, предшествовало технико-экономическое обоснование, или концептуальная фаза анализа проекта. Фаза разработки требований может быть разбита на выявление требований (сбор, понимание, рассмотрение и выяснение потребностей заинтересованных лиц), анализ (проверка целостности и законченности), спецификация (документирование требований) и проверка правильности.

Функциональный характер — требования к поведению системы

- Бизнес-требования
- Пользовательские требования
- Функциональные требования

Нефункциональный характер — требования к характеру поведения системы

- Бизнес-правила — определяют ограничения, проистекающие из предметной области и свойств автоматизируемого объекта (предприятия)
- Системные требования и ограничения — определения элементарных операций, которые должна иметь система, а также различных условий, которым она может удовлетворять. К системным ограничениям относятся ограничения на программные интерфейсы, требования к атрибутам качества, требования к применяемому оборудованию и ПО.
- Атрибуты качества
- Внешние системы и интерфейсы
- Ограничения

**Проверка требований**

Все требования должны быть поддающимися проверке. Наиболее общепринятая методика проверки — тесты. Если проверка тестами невозможна, тогда должен использоваться другой метод проверки (анализ, демонстрация, осмотр или обзор дизайна).

Определённые требования, по своей сути, не являются поддающимися проверке. Они включают требования, которые говорят, что система *никогда* не должна или *всегда* должна показывать специфическое свойство. Надлежащее тестирование этих требований потребовало бы бесконечного цикла тестирования. Такие требования должны быть переопределены так, чтобы они стали поддающимися проверке. Как указано выше, все требования должны быть поддающимися проверке.

Нефункциональные требования, которые являются неподдающимися проверке на программном уровне, все равно должны быть сохранены как документация намерений клиента; Такие требования к продукту могут быть преобразованы в требования к процессу. Например, нефункциональное требование, чтобы ПО не содержало «потайных ходов», может быть удовлетворено заменой на требование использовать парное программирование.

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. На основании выбранного программного продукта разработать:

- Требования функционального характера
- Требования нефункционального характера.

2.Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

Разработать требования к программному продукту. Подробно расписать требования функционально и нефункционального характера.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;

5. Требования к программному продукту;
6. Заключение

Контрольные вопросы.

1. Дать определение, что такое требования к программному обеспечению.
2. В каком виде могут выражаться требования?
3. На какой стадии используются совокупность требований?
4. Перечислите требования, носящие функциональный характер.
5. Перечислите требования, носящие нефункциональный характер.

## Практическое занятие № 6

## Тема: Проектирование программы с использованием операций языка.

**Цель:** научить проектировать программу с использованием стандартных операций языка программирования.

Пояснения.

Операции языка C# приведены в порядке убывания приоритетов. Операции с разными приоритетами разделены чертой.

Операция: Описание

Доступ к элементу x()

Вызов метода или делегата x[]

Доступ к элементу x++

Постфиксный инкремент x--

Постфиксный декремент new

Выделение памяти typeof

Получение типа checked

Проверяемый код unchecked

Непроверяемый код +

Унарный плюс -

Арифметическое отрицание !

Логическое отрицание

~ Поразрядное отрицание

++x Префиксный инкремент

--x Префиксный декремент

(тип) x Преобразование типа

* Умножение

/ Деление

% Остаток от деления

<< Сдвиг влево

>> Сдвиг вправо

< Меньше

> Больше

<= Меньше или равно

>= Больше или равно

Is Проверка принадлежности типу

As Приведение типа

= Равно

!= Не равно

| | |
|---|---|
| & | Поразрядное И |
| ^ | Поразрядное исключающее ИЛИ |
| \| | Поразрядное ИЛИ |
| && | Логическое И |
| \|\| | Логическое ИЛИ |
| ? : | Условная операция |

| | |
|---|---|
| = | Простое присваивание |
| *= | Умножение с присваиванием |
| /= | Деление с присваиванием |
| %= | Остаток от деления с присваиванием |
| += | Сложение с присваиванием |
| -= | Вычитание с присваиванием |
| <<= | Сдвиг влево с присваиванием |
| >>= | Сдвиг вправо с присваиванием |
| &= | Поразрядное И с присваиванием |
| ^= | Поразрядное исключающее ИЛИ с присваиванием |
| \|= | Поразрядное ИЛИ с присваиванием |

Пример разработанной программы.

Периметр квадрата, площадь которого равна a; using System; namespace Example

{class Program

{static void Main()

{Console. Write("s= ");

float s = float. Parse(Console. ReadLine());

double p = 4 * Math. Sqrt(s);

Console. WriteLine("p=" + p);

}}}

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Спроектировать программу согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

Спроектировать программу для решения:

1. площадь прямоугольного треугольника по двум катетам a, b.
2. радиус окружности, длина которой равна l;
3. среднее арифметическое кубов двух данных чисел;

## Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект программы;
6. Заключение

## Контрольные вопросы.

1. Допустимы ли следующие способы записи ++(++i), (i--)--, ++(i--) и т.д. И почему.

2. Допустимы ли следующие способы записи !(-i), -(!a). И почему.

3. Выясните, чему равен результат данного выражения: 1) i

И объясните, как получился данный результат.

4. Объясните, какое значение примет переменная t в данном фрагменте программы:

```
int a=10, b=3;

bool t=(a>=b && a!=2*b || a<0);
```

5. Объясните, какие значения примут переменные t и b после выполнения данного фрагмента программы:

```
int a=10, b=3;

int t=(a++)-b;

int b+=t*a;
```

## Практическое занятие № 7

## Тема: Проектирование программы с использованием операторов языка.

**Цель:** научиться проектировать программы с использованием операторов языка программирования.

Пояснения.

Программа на языке C# состоит из последовательности операторов, каждый из которых определяет законченное описание некоторого действия и заканчивается точкой с запятой. Все операторы можно разделить на 4 группы: операторы следования, операторы ветвления, операторы цикла и операторы передачи управления.

Операторы следования выполняются компилятором в естественном порядке: начиная с первого до последнего. К операторам следования относятся: выражение и составной оператор.

Операторы ветвления позволяют изменить порядок выполнения операторов в программе. К операторам ветвления относятся условный оператор *if* и оператор выбора *switch.*

Операторы цикла используются для организации многократно повторяющихся вычислений. К операторам цикла относятся: цикл с предусловием while, цикл с постусловием do while, цикл с параметром for и цикл перебора foreach.

В C# есть несколько операторов, изменяющих естественный порядок выполнения команд: оператор безусловного перехода *goto*, оператор выхода *break*, оператор перехода к следующей итерации цикла *continue*, оператор возврата из метода *return* и оператор генерации исключения *throw.*

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

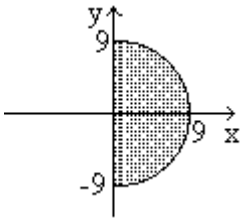
1. Спроектировать программу согласно заданию.

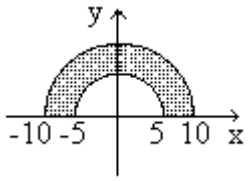
2. Составить письменно отчёт и продемонстрировать преподавателю.

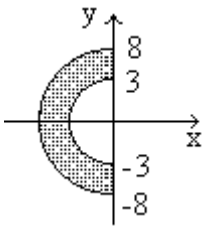
Задание.

Дана точка на плоскости с координатами (x, y). Составить программу, которая выдает одно из сообщений «Да», «Нет», «На границе» в зависимости от того,

лежит ли точка внутри заштрихованной области, вне заштрихованной области или на ее границе.

1. 


2. 


3. 


Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект программы;
6. Заключение

Контрольные вопросы.

1. Какие группы операторов существуют в языке программирования.

2. Приведите пример операторов следования и их применение.

3. Приведите пример операторов ветвления и их применение.

4. Приведите пример операторов цикла и их применение.

5. Приведите пример операторов передачи управления и их применение.

## Практическое занятие № 8

## Тема: Проектирование программы с использованием классов и методов.

**Цель:** научится проектировать приложения с использованием классов и методов.

Пояснения.

Класс - это шаблон, который определяет форму объекта. Класс должен определять только одну логическую сущность.

Ключевые слова C#, указывающие уровень доступности - классификаторы доступности :

**Классификатор доступности C#**

**Описание** public

Помечает метод, как доступный из объектной переменной, а также из всех производных классов private

Помечает метод, как доступный только из класса, определяющего этот метод. В C# любой член по умолчанию определяется, как private protected

Помечает метод, как доступный для определяющего класса, а так же для любого производного класса. Однако защищенные методы не доступны из объектной переменной internal

Определяет метод, как доступный для любого типа только внутри данного компоновочного блока, но не снаружи protected internal

Определяет метод, доступ к которому ограничивается рамками текущего компоновочного блока или типами, созданными из определяющего класса в данном компоновочном блоке

Классификатор доступности необязателен и если он не указан, то по умолчанию подразумевается, что член закрыт (private) в рамках класса, где он определен.

Наследуемый класс не может иметь более открытый классификатор доступности, чем его предок.

Методы - это процедуры(подпрограммы), которые манипулируют данными, определенными в классе, и во многих случаях обеспечивают доступ к этим данным.

В качестве имени метода можно использовать любой допустимый идентификатор, отличный от тех, которые уже использованы для других

элементов программы в пределах текущей видимости. В качестве имен методов нельзя использовать ключевые слова C#.

Элемент **доступ** означает классификатор доступа.

Элемент **тип_возврата** указывает тип значения, возвращаемого методом. Это может быть любой допустимый тип, включая типы классов, создаваемые программистом. Если метод не возвращает значения, необходимо указать тип void.

Элемент **список_параметров** представляет собой последовательность пар, состоящих из типа данных и идентификатора, разделенных запятыми. Параметры - это переменные, которые получают значения аргументов, передаваемых методу при вызове. Если метод не имеет параметров, **список_параметров** остается пустым.

***Пример***.

```
using System;

namespace Hello

{class Program

{static double min(double a, double b)

{return (a < b) ? a : b;

}static void Main(string[] args)

{Console. Write("x=");

double x = double.Parse(Console.ReadLine());

Console.Write("y=");

double y = double.Parse(Console.ReadLine());

double z = min(3 * x, 2 * y) + min(x - y, x + y);

Console.WriteLine("z=" + z); }}}
```

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Спроектировать программу согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

1. Разработать метод f(x), который вычисляет значение по следующей формуле: $f(x)=x^3-\sin x$. Определить, в какой из точек a или b, функция принимает наибольшее значение

2.Разработать метод f(x), который вычисляет значение по следующей формуле: $f(x)=\cos(2x)+\sin(x-3)$. Определить, в какой из точек a или b, функция принимает наименьшее значение.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект программы;
6. Заключение

Контрольные вопросы.

1. Приведите запись сигнатуры метода.

2. Дать определение понятию конструктор.

3. Дать определение понятие формального и фактического параметра. Объяснить их различия.

4. Какой оператор отвечает за возврат значения из метода

5.Какой оператор показывает, что возвращаемое значение отсутствует.

**Практическое занятие № № 9 - 10**

**Тема: Проектирование программы для работы с массивами и строками.**

**Цель:** научится проектировать программы с использованием массивов и строк.

Пояснения.

Массивы являются коллекциями объектов одного типа. Поскольку длина массивов практически не ограничена, они могут использоваться для хранения тысяч или даже миллионов объектов, но размер массива должен быть указан при его создании. Каждый элемент массива доступен по числовому индексу, указывающему позицию или ячейку, в которой объект хранится в массиве. Массивы могут хранить как ссылочные типы, так и типы значений.

Строка C# представляет собой группу одного или нескольких знаков, объявленных с помощью ключевого слова string, которое является ускоренным методом языка C# для класса System.String. В отличие от массивов знаков в C или C++, строки в C# гораздо проще в использовании и менее подвержены ошибкам программирования.

Строковые объекты являются неизменяемыми: после создания их нельзя изменить. Методы, работающие со строками, возвращают новые строковые объекты. Поэтому в целях повышения производительности большие объемы работы по объединению строк или другие операции следует выполнять в классе StringBuilder, как показано в далее в примерах кода.

***Пример***

```
using System;

using System.Text;

namespace ConsoleApplication

{class Class

{static void Main()

{Console.WriteLine ("Введите строку: ");

StringBuilder a = new StringBuilder (Console.ReadLine());

Console.WriteLine ("Исходная строка: "+a);

Console.WriteLine ("Введите символ x: ");
```

```
char x=char.Parse (Console.ReadLine());

Console.WriteLine ("Введите символ y: ");

char y=char.Parse (Console.ReadLine());

for (int i=0; i

if (a[i]==x){a.Insert (i+1,y); ++i;}

Console.WriteLine ("Измененная строка: "+a);

}}}
```

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Спроектировать программу согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

1. Разработать программу, которая для заданной строки s определяет, какой из двух заданных символов встречается чаще в строке.

2. Разработать программу, которая для заданной строки s определяет, сколько различных символов встречается в строке.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект программы;
6. Заключение

Контрольные вопросы.

1. В какой кодировке хранится строка?

2. Какие методы отвечают за преобразование символов в верхний и нижний регистр.

3. За что отвечает экземплярный метод Split?

4. Приведите примеры способов инициализации строки.

5. В чём отличия класса String от StringBuilder?

## Практическое занятие № 11

## Тема: Использование средств манипуляции реляционной алгебры при работе с БД.

**Цель:** используя средства манипуляции реляционной алгебры добиться научиться эффективно, работать с БД.

Пояснения.

Объединением двух совместимых по объединению отношений R1 и R2 является отношение R3, содержащее множество всех кортежей, принадлежащих или R1, или R2, или обоим вместе. Другой вариант завершения этого определения «…, принадлежащих R1 и тех кортежей R2, которые не принадлежат R1». Последнее определение подчеркивает, что в результирующем отношении не должно быть совпадающих кортежей (дублей). Учитывая, что в отношении по определению не может быть одинаковых кортежей, далее мы не будем подчеркивать это требование к результирующему отношению. Естественно, что R3 также совместимо по объединению с R1 и R2.

Пересечением двух совместимых по объединению отношений R1 и R2 является отношение R3, содержащее кортежи, принадлежащие как R1, так и R2.

Разностью двух совместимых по объединению отношений R1 и R2 является отношение R3, кортежи которого принадлежат R1, но не принадлежат R2 (т.е. кортежи из R1, которых нет в R2).

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

1.Заполнить произвольными данными 10 строк таблицы. Выполнить объединение таблиц БД по правилам реляционной алгебры.

**ПРЕПОДАВАТЕЛЬ**

**Личный номер преподавателя**

Ф.И.О. преподавателя

Дата рождения

Пол

Адрес

**ОБУЧАЮЩИЙСЯ**

**Личный номер**

Ф.И.О. научного

Дата рождения

Пол

Адрес

2. Заполнить произвольными данными 10 строк таблицы. Выполнить пересечение таблиц БД по правилам реляционной алгебры.

**СТУДЕНТ**

Ф.И.О. студента

**Серия и номер паспорта**

Пол

Дата рождения

Семейное положение

**ЖИЛЕЦ**

Ф.И.О. жильца

**Серия и номер паспорта**

Пол

Датарождения

Семейное положение

3. Выполнить операцию разности чтобы в результате вычитания отношения СТУДЕНТ из отношения ЖИЛЕЦ получили отношение НЕ СТУДЕНТ, содержащее сведения о всех проживающих в общежитии, но не являющихся студентами.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Дать определение понятию кортеж.

2. Дать определение понятию объединение отношений.

3. Дать определение понятию пересечение отношений.

4. Дать определение понятию разность отношений.

## Практическое занятие № 12

## Тема: Использование реляционного исчисления при работе с БД.

**Цель:** научиться использовать средства реляционного исчисления при работе с БД.

Пояснения.

Декартовым произведением отношения A со схемой и отношения B со схемой является отношение C, со схемой равной объединению схем отношений A и B, кортежи которого получены путем конкатенации (присоединения) каждого кортежа отношения B с каждым кортежем отношения A.

Операция деления одного отношения (делимого) на другое отношение (делитель) может быть выполнена, если все множество атрибутов схемы делителя является подмножеством атрибутов схемы делимого.

Результирующее отношение содержит только те атрибуты делимого, которых нет в делителе. В него включаются только те кортежи, декартово произведение которых с делителем содержится в делимом (является подмножеством делимого).

Выбор (селекция) – операция получения нового отношения с той же схемой, что и исходное отношение, но, кортежи которого являются подмножеством кортежей исходного. В него включаются только те кортежи исходного отношения, значения определенных атрибутов которых удовлетворяют заданным ограничениям.

Ограничения могут быть заданы в виде логического выражения, элементами которого являются простейшие условия типа «Имя атрибута – знак сравнения – значение атрибута» (в общем случаев «Выражение – знак сравнения – выражение»), которые могут соединяться булевыми операторами И (AND), ИЛИ (OR), иметь знак отрицания (NOT), а также использовать круглые скобки для изменения старшинства выполнения булевых операций по правилам математической логики.

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

1. Пусть имеем отношение **КС,** содержащее список участников команды факультета **Компьютерные сети** по шахматам и отношение **ИБ,** содержащее аналогичный список команды факультета **Информационная безопасность**. Вычислить декартово произведение **КС × ИБ** имеющее название **ВСТРЕЧИ,** содержащее список пар участников, которые должны играть друг с другом. Полученный результат письменно внести в отчёт.

**КС ИБ**

Ф.И.О. игрока

КС

Разряд

Дата рождения

Ф.И.О. игрока

ИБ

Разряд

Дата рождения

2. Выполнить операцию деления согласно представленным таблицам.
Полученный результат письменно внести в отчёт.

ДЕЛИМОЕ ДЕЛИТЕЛЬ

**Ф.И.О.**

**Иностранный язык**

**Степень владения языком**

.

**Иностранный язык**

**Степень владения языком**

Иванов

Английский свободно

Английский со словарем

Сидоров

Английский свободно

Турецкий свободно

Сидоров

Японский разговорный

Сидоров

Турецкий со словарем

Кузнецова

Английский со словарем

Кузнецова

Турецкий свободно

Иванова

Немецкий разговорный

Иванова

Турецкий свободно

Иванова

Английский  со словарем

3. Требуется отобрать кортежи, относящиеся к первому году учебы, причем для студентов, не получающих стипендию, но имеющих достаточно высокий

рейтинг – от 800 до 900 баллов. Полученный результат письменно внести в отчёт.

Код студента

Ф.И.О. студента

Номер семестра

Тип стипендии в семестре

Рейтинг за семестр

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Разъяснить суть операции декартово произведение при работе с БД.

2. Разъяснить суть операции деление при работе с БД.

3. Разъяснить суть операции селекции при работе с БД.

4. Разъяснить суть операции ограничения при работе с БД.

## Практическое занятие №13

## Тема: Проектирование концептуальной модели базы данных.

Цель: научиться проектировать концептуальную модель БД.

Пояснения.

Концептуальная (содержательная) модель — это абстрактная модель, определяющая структуру моделируемой системы, свойства её элементов и причинно-следственные связи, присущие системе и существенные для достижения цели моделирования.

Рассмотрим проектирование пошагово концептуальной модели.

1. **Выделить основные абстракции (сущность, атрибут, связь) в предметной области и определить их параметры.**

Определим следующие сущности: СТУДЕНТ, ЭКЗАМЕН, ОЦЕНКА

Определим атрибуты сущностей. Пусть для упрощения сущность СТУДЕНТ характеризуется только фамилией. Фамилию мы и возьмем в качестве атрибута. Так как фамилия может неоднозначно идентифицировать объект, введем дополнительный атрибут Код студента, уникальный для каждого студента. Таким образом, сущность СТУДЕНТ характеризуется двумя атрибутами код студента, фамилия.

Аналогично определим сущность ЭКЗАМЕН с атрибутами код экзамена, предмет, дата экзамена и сущность ОЦЕНКА с атрибутом значение оценки (оценка). Между этими сущностями существуют следующие связи: студент сдавал экзамен, студент получил оценку, по экзамену получены следующие оценки.

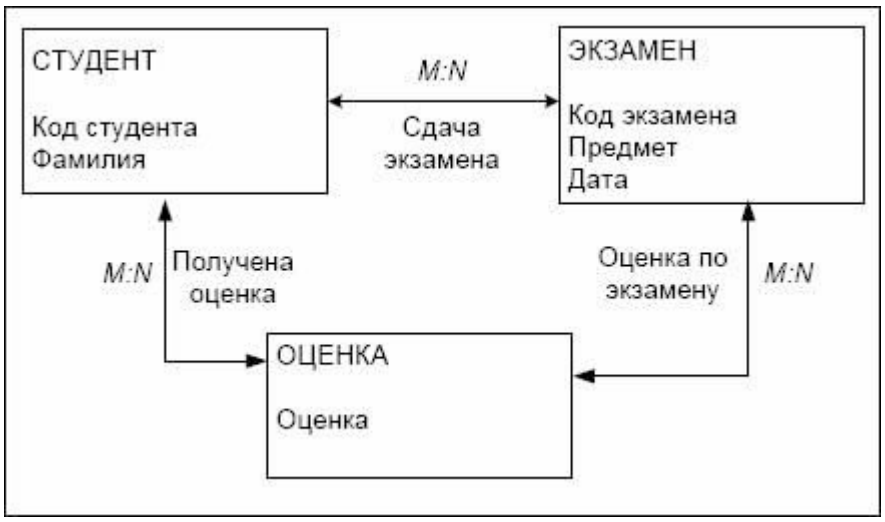
2.**Сформировать максимально полный перечень возможных запросов к базе данных на основе анализа предметной области.**

По смыслу задачи к базе данных возможны следующие запросы:

- Какие оценки получил студент с заданной фамилией (кодом);
- Какие студенты получили заданное значение оценки;
- Какие экзамены сдал студент с заданной фамилией (кодом);
- Какую оценку по конкретному предмету получил студент с заданной фамилией (кодом).

В данном примере остановимся на этих запросах.

**3.Построить концептуальную модель.**




По этой диаграмме можно ответить на все вопросы, кроме последнего. Для реализации и последнего запроса в перспективе введем новую агрегированную сущность. Определим эту сущность как ЭКЗАМЕНАЦИОННАЯ ВЕДОМОСТЬ с атрибутами код студента, фамилия, код экзамена, предмет, дата экзамена, оценка.

**4. Описать домены (допустимые множества значений, которые могут принимать атрибуты), указывая типы соответствующих данных и их характеристики.**

*Код студента* принимает значения из множества целых чисел.

*Фамилия* принимает символьное значение, максимальная длина 20 символов.

*Код экзамена* принимает значения из множества целых чисел.

*Предмет* принимает символьное значение, максимальная длина 20 символов.

*Дата экзамена* принимает значение дата в формате 00.00.00.

*Оценка* принимает целое значение от 2 до 5.

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

Спроектировать концептуальную модель базы данных страховой компании.

*Описание предметной области*

Вы работаете в страховой компании. Вашей задачей является отслеживание ее финансовой деятельности. 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.

*Возможный набор сущностей*

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).

Контрольные вопросы.

1.Дать определение понятию концептуальная модель.

2.Какие действия включает первый шаг построения концептуальной модели?

3. Какие действия включает второй шаг построения концептуальной модели?

4.Какие действия включает третий шаг построения концептуальной модели?

5.Какие действия включает четвёртый шаг построения концептуальной модели?

## Практическое занятие № 14

## Тема: Проектирование реляционной модели базы данных.

**Цель:** научиться проектировать реляционную модель баз данных.

Пояснения.

Реляционная база данных — база данных, основанная на реляционной модели данных. Слово «реляционный» происходит от англ. relation (отношение). Для работы с реляционными БД применяют реляционные СУБД.

Рассмотрим построение реляционной базы данных на примере.

Табличная база данных "Комплектующие компьютера и поставщики" содержит информацию о различных комплектующих и имеет поля: "Счетчик", "Наименование", "Описание", "Название фирмы", "Адрес", "Цена" (в рублях) - табл. 1.

Таблица 1. Комплектующие компьютера и поставщики Счетчик

Наименование

Описание

Название фирмы

Адрес

Цена

1 Системный блок Pentium

Фирма 1

Адрес 1

10000

2 Системный блок Pentium

Фирма 2

Адрес 2

9000

3 Монитор 15"

Фирма 1

Адрес 1

5000

4 Монитор 15"

Фирма 2

Адрес 2

6000

5 Клавиатура 104 кл.

Фирма 1

Адрес 1

250

6 Клавиатура 104 кл.

Фирма 2

Адрес 2

300

7 Мышь 3кн

Фирма 1

Адрес 1

100

8 Мышь 3 кн

Фирма 2

Адрес 2

150

Мы видим, что почти половину объема таблицы составляет избыточная, дублированная информация.

Проанализируем причину дублирования. Комплектующие компьютера имеют два неотъемлемых свойства: "Наименование" и "Описание". "Название фирмы", "Адрес" и "Цена" не являются свойствами комплектующих компьютера, они являются свойствами поставщика.

Естественно разделить исходную таблицу на две: "Комплектующие" (табл. 2) и "Поставщики" (табл. 3).

Каждая таблица должна содержать, по крайней мере, одно *ключевое поле*, содержимое которого уникально для каждой записи в этой таблице. В таблицу "Комплектующие" введем поле "Код комплектующих". Именно это поле будет ключевым в данной таблице.

Таблица 2. Комплектующие Код комплектующих

Наименование

Описание К1 Системный блок Pentium

К2 Монитор

15"

КЗ Клавиатура 104 кл.

К4 Мышь Зкн.

В таблицу "Поставщики" введем дополнительное поле "Код поставщика". Именно это поле будет ключевым в данной таблице.

Таблица 3. Поставщики

| Код поставщика | Название фирмы | Адрес |
|---|---|---|
| П1 | Фирма1 | Адрес 1 |
| П2 | Фирма2 | Адрес2 |

**Связывание таблиц**

После создания различных таблиц, содержащих данные, относящиеся к различным аспектам базы данных, необходимо обеспечить целостность базы данных. Для этого надо *связать* таблицы между собой.

При связи "один-ко-многим" каждой записи в одной (главной) таблице могут соответствовать несколько записей в другой (подчиненной) таблице, а запись в подчиненной таблице не может иметь более одной соответствующей ей записи в главной таблице.

Если одной записи в первой таблице могут соответствовать несколько записей во второй таблице и, наоборот, одной записи во второй таблице - несколько записей в первой таблице, то реализуется связь "многие-ко-многим".

В нашем случае реализуется именно такая связь. Одной записи в таблице "Комплектующие" соответствуют две записи в таблице "Поставщики", так как устройства одного типа продаются двумя фирмами. Одной же записи таблицы "Поставщики" соответствуют четыре записи таблицы "Комплектующие", так как одна фирма продает устройства четырех типов.

Две таблицы, находящиеся в отношении "многие-ко-многим", могут быть связаны только с помощью третьей (связующей) таблицы. Таблицы "Комплектующие" и "Поставщики" можно связать в отношении "многие-ко-многим" путем создания двух связей "один-ко-многим" по отношению к таблице "Цена".

Таблицы "Комплектующие" и "Поставщики" будут являться главными по отношению к таблице "Цена".

Связь между таблицами устанавливает отношения между совпадающими значениями в полях с одинаковыми именами. С ключевым полем главной таблицы (*первичный ключ*) связывается одноименное поле подчиненной таблицы (*внешний ключ*).

В главной таблице "Комплектующие" поле "Код комплектующих" является первичным ключом, соответственно в подчиненной таблице "Цена" должно существовать одноименное поле, которое является внешним ключом.

Таблица "Поставщики" также является главной по отношению к таблице "Цена". Ее поле "Код поставщика" является первичным ключом, соответственно в подчиненной таблице "Цена" должно существовать одноименное поле, которое является внешним ключом.

Таким образом, таблица "Цена" должна содержать следующие поля (табл. 4):

- "Счетчик" (ключевое поле);
- "Код комплектующих" (поле внешнего ключа для таблицы "Комплектующие");
- "Код поставщика" (поле внешнего ключа для таблицы "Поставщики");
- "Цена" (числовое поле).

Таблица 4. Цена Код комплектующих

Код комплектующих

Код поставщика

Цена

1 К1

П1 9000

2 К1

П2 10000

3

К2

П1 5000

4 К2

П2

6000

5 К3

П1 250

6 К3

П2 300

7 К4

П1 100

8 К4

П2 150

Межтабличная связь *обеспечивает целостность данных*. Связанные таблицы представляют собой единую базу данных, в которой можно создавать новые таблицы, а также запросы и отчеты, содержащие данные из связанных таблиц.

*Базы данных, состоящие из связанных двумерных таблиц, принято называть* ***реляционными****.*

Прежде чем приступить к созданию реляционной базы данных, необходимо продумать ее *проект*. Проект представляет собой модель будущей БД, состоящей из объектов и их связей, необходимых для выполнения поставленных задач.

Процесс проектирования включает, прежде всего, определение перечня необходимых таблиц и задание их структуры, а также установление типа связей между этими таблицами.

## Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

## Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

## Задание.

Спроектрировать проект реляционной базы данных "Коллекция аудиозаписей", которая бы содержала главную таблицу "Список аудио-CD" и подчиненную таблицу "Содержание аудио-CD".

## Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

## Контрольные вопросы.

1. Дать определение понятию реляционная модель данных.
2. Почему в некоторых случаях целесообразно использовать многотабличные, а не однотабличные базы данных?
2. Какие типы связей между таблицами возможны в реляционных базах данных?
4. Дать определение понятию ключевое поле.

## Практическое занятие № 15

## Тема: Построение баз данных с использованием СУБД MS Access

**Цель:** научится разрабатывать базы данных с помощью системы управления базами данных MS Access.

Пояснения.

Microsoft Office Access или просто Microsoft Access — реляционная СУБД корпорации Microsoft. Имеет широкий спектр функций, включая связанные запросы, связь с внешними таблицами и базами данных. Благодаря встроенному языку VBA, в самом Access можно писать приложения, работающие с базами данных.

Основные компоненты MS Access:

- построитель таблиц;
- построитель экранных форм;
- построитель SQL-запросов (язык SQL в MS Access не соответствует стандарту ANSI);
- построитель отчётов, выводимых на печать.

Они могут вызывать скрипты на языке VBA, поэтому MS Access позволяет разрабатывать приложения и БД практически «с нуля» или написать оболочку для внешней БД.

Microsoft Jet Database Engine (англ.), которая используется в качестве движка базы данных MS Access является файл-серверной СУБД и потому применима лишь к приложениям, работающим с небольшими объемами данных и при небольшом числе пользователей, одновременно работающих с этим данными. Непосредственно в Access отсутствует ряд механизмов, необходимых в многопользовательских БД, таких, например, как триггеры.

Access, при работе с базой данных, иначе взаимодействует с жёстким (или гибким) диском, нежели другие программы.

В других программах, файл-документ, при открытии, полностью загружается в оперативную память, и новая редакция этого файла (изменённый файл) целиком записывается на диск только при нажатии кнопки «сохранить».

В Access новая редакция содержимого изменённой ячейки таблицы записывается на диск (сохраняется) сразу, как только курсор клавиатуры будет помещён в другую ячейку (или новая редакция изменённой записи записывается на диск сразу, как только курсор клавиатуры будет поставлен в другую запись (в другую строку)?). Таким образом, если внезапно отключат электричество, то пропадёт только изменение той записи, которую не успели покинуть.

Целостность данных в Access обеспечивается также за счет механизма транзакций.

Кнопка «Сохранить» в Access тоже есть, но в Access в режиме просмотра данных она нужна, в первую очередь, для сохранения изменённого режима показа таблицы или другого объекта — то есть, для сохранения таких изменений, как:

- изменение ширины столбцов и высоты строк,
- перестановка столбцов в режиме просмотра данных, «закрепление» столбцов и освобождение закреплённых столбцов,
- изменение сортировки,
- применение нового фильтра,
- изменение шрифта; цвета текста, сетки и фона
- и т. п.

Кроме того, в Access эта кнопка нужна в режиме «Конструктор» для сохранения изменений структуры объекта базы данных, сделанных в этом режиме.

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

1. Откройте **Access**, создайте **новую** базу данных, сохраните в своей папке с именем **Student.** Изучите **окно базы данных**, найдите вкладки объектов, ознакомьтесь с формой представления данных и панелью инструментов.
2. **В режиме конструктора** создайте макет таблицы, содержащей следующие поля:

Таблица 1

Код студента

Фамилия

Имя

Группа

Адрес

Телефон

Фото

Примечания

Имена полей задайте такие же, как в таблице 1.

Свойства полей:

**Код студента:** *тип* ***счетчик****, подпись поля - №, индексированное, без повторений.*

**Фамилия:** *тип* ***текстовый***, *размер поля 50, обязательное, без пустых строк, индексированное.*

**Имя:** *тип* ***текстовый***, *размер поля 25, обязательное, без пустых строк.*

**Группа:** *тип* ***текстовый****, размер поля 7, подпись № Группы, маска ввода КОМ-999, обязательное, без пустых строк, индексированное.*

**Адрес:** *тип* ***текстовый***, *размер поля 200, не обязательное, не индексированное.*

**Телефон:** *тип* ***текстовый***, *размер поля 8, маска ввода 99-99-99, не обязательное, не индексированное.*

**Фото:** *тип* ***OLE – объект.***

**Примечания:** *тип* ***мето.***

Закройте таблицу, присвойте имя **Студенты**.

3. Для ввода данных в таблицу создайте **автоформу в столбец** с именем **Формуляр**. Она отображает все поля таблицы. Обратите внимание, как используются подписи полей. Откройте форму и введите 10-12 записей, проверяя правильность описания полей. В данных должны быть повторения фамилий, номеров групп (используйте три номера группы). Для ввода фото используйте вставку объекта из **Clip Gallery.** Если необходимо сделать исправления, откройте **таблицу в режиме конструктора** и внесите необходимые изменения в макет таблицы.
4. Войдите в **режим конструктора форм**. Отредактируйте формуляр: текстовые поля расположены в столбце слева один под другим, фотографию и примечания поместите справа один под другим. Уменьшите размер полей фотографии и примечаний, а также измените в макете формы свойство рамки объекта «Установка размеров» на значение

*«По размеру рамки»*. Окно свойства вызывается нажатием правой кнопки мыши на активном элементе формы.

5. Для просмотра записей таблицы создайте с использованием **мастера форм** ленточную форму, не включая в нее фотографии и примечания. Задайте имя **Ленточная форма.**
6. Выполните в этой форме операции, характерные для баз данных.

А). **Сортировка записей** выполняется по значению одного поля, активного в данный момент.

Выполните сортировку записей:

- по фамилиям,
- по номерам групп,
- по номерам телефонов.

Б). **Отбор записей** выполняется с использованием **фильтров**. Чтобы задать условия отбора, используется кнопка . Чтобы выполнить отбор или отменить условия отбора, используется кнопка . В бланке отбора можно задать условия, выполняемые одновременно (задаются в одной строке бланка) или условия, выполняемые порознь (задаются через закладку «**Или**»). Условия отбора могут быть выбраны из списка или записаны в виде отношений или логических выражений.

Выполните последовательно отбор:

- Всех студентов, учащихся в одной группе (например, 364).
- Всех студентов, учащихся в двух каких-нибудь группах (например, 364 **Или** 365).
- Студента, например, Петрова, учащегося в указанной группе.
- Всех студентов, фамилии которых начинаются на букву «А» или «Б».Используется маска *,например А* **Или** Б*.

В). Пункт меню **Записи – Расширенный фильтр** позволяет выполнить отбор записей с использованием **бланка фильтра** (запроса). В бланке фильтра будет добавлена таблица Студенты. Для фильтра можно задать поля, по которым выполняется отбор, сортировки по значениям полей, и условия отбора. Для отбора данных поле таблицы двойным щелчком переносится в нижнюю часть бланка. В строке «Сортировка» выбирается способ сортировки по указанному полю. В строке «Условие отбора» накладывается произвольное условие на значения поля. Если условия записаны в одной строке бланка, они накладываются друг на друга, то есть должны быть выполнены одновременно. Для записи условий, выполняемых врозь, используется строка бланка «Или». Для записи условий используется построитель выражений, вызываемый правой кнопкой мыши в области условия . Вызовите построитель выражений и ознакомьтесь с его окном. Для сравнения текстовых строк используется операция **Like**, например, Like Петров, или Like П*.

Выполните последовательно отбор данных, как в пункте Б). Каждый результат отбора сохраните (меню **Файл - Сохранить как**) в **форме отчета** с именами, соответственно, **Фильтр1 - Фильтр4.**

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Дать определение понятию система управления базами данных (СУБД)?

2. Перечислите состав программы Microsoft Access?

3. Какие языки программирования используются в Microsoft Access?

4. Как происходит сохранение БД в Microsoft Access?

## Практическое занятие № 16

## Тема: Построение баз данных с использованием СУБД MS SQL-Server

Цель: научится разрабатывать базы данных с помощью системы управления базами данных MS SQL Server.

Пояснения.

Microsoft SQL Server — система управления реляционными базами данных (СУБД), разработанная корпорацией Microsoft. Основной используемый язык запросов — Transact-SQL, создан совместно Microsoft и Sybase. Transact-SQL является реализацией стандарта ANSI/ISO по структурированному языку запросов (SQL) с расширениями. Используется для работы с базами данных размером от персональных до крупных баз данных масштаба предприятия; конкурирует с другими СУБД в этом сегменте рынка.

Сервер баз данных Microsoft SQL Server в качестве языка запросов использует версию языка SQL, получившую название Transact-SQL (сокращённо T-SQL). Язык T-SQL является реализацией SQL-92 (стандарт ISO для языка SQL) с множественными расширениями. T-SQL позволяет использовать

дополнительный синтаксис для хранимых процедур и обеспечивает поддержку транзакций (взаимодействие базы данных с управляющим приложением).

При взаимодействии с сетью Microsoft SQL Server и Sybase ASE используют протокол уровня приложения под названием Tabular Data Stream (TDS, протокол передачи табличных данных). Протокол TDS также был реализован в проекте FreeTDS с целью обеспечить различным приложениям возможность взаимодействия с базами данных Microsoft SQL Server и Sybase.

Для обеспечения доступа к данным Microsoft SQL Server поддерживает Open Database Connectivity (ODBC) — интерфейс взаимодействия приложений с СУБД. Версия SQL Server 2005 обеспечивает возможность подключения пользователей через веб-сервисы, использующие протокол SOAP. Это позволяет клиентским программам, не предназначенным для Windows, кроссплатформенно соединяться с SQL Server. Компания Microsoft также выпустила сертифицированный драйвер JDBC, позволяющий приложениям под управлением Java (таким как BEA и IBM WebSphere) соединяться с Microsoft SQL Server 2000 и 2005.

Также SQL Server поддерживает зеркалирование и кластеризацию баз данных. Кластер сервера SQL — это совокупность одинаково конфигурированных серверов; такая схема помогает распределить рабочую нагрузку между несколькими серверами. Все сервера имеют одно виртуальное имя, и данные распределяются по IP-адресам машин кластера в течение рабочего цикла. Также в случае отказа или сбоя на одном из серверов кластера доступен автоматический перенос нагрузки на другой сервер.

SQL Server поддерживает избыточное дублирование данных по трем сценариям:

Снимок: Производится «снимок» базы данных, который сервер отправляет получателям.

История изменений: Все изменения базы данных непрерывно передаются пользователям.

Синхронизация с другими серверами: Базы данных нескольких серверов синхронизируются между собой. Изменения всех баз данных происходят независимо друг от друга на каждом сервере, а при синхронизации происходит сверка данных. Данный тип дублирования предусматривает возможность разрешения противоречий между БД.

В SQL Server 2005 встроена поддержка .NET Framework. Благодаря этому хранимые процедуры БД могут быть написаны на любом языке платформы .NET, используя полный набор библиотек, доступных для .NET Framework, включая Common Type System (система обращения с типами данных в Microsoft .NET Framework). Однако, в отличие от других процессов, .NET Framework, будучи базисной системой для SQL Server 2005, выделяет дополнительную

память и выстраивает средства управления SQL Server вместо того, чтобы использовать встроенные средства Windows. Это повышает производительность в сравнении с общими алгоритмами Windows, так как алгоритмы распределения ресурсов специально настроены для использования в структурах SQL Server.

Для построения базы данных с использованием MS SQL Server необходимо использовать команду и задать ряд параметров :

• PRIMARY — файл определяется как первичное устройство.
• NAME — логическое имя; по умолчанию совпадает с именем файла.
• FILENAME — полное имя файла на диске.
• SIZE — исходный размер файла. Минимальный размер файла журнала равен 512 Кбайт.
• MAXSIZE — максимальный размер файла.
• UNLIMITED — размер файла не ограничивается.
• FILEGROWTH — приращение размера в мегабайтах (MB), килобайтах (KB) или процентах (%). По умолчанию приращение равно 10%.
• FOR LOAD — обеспечивает обратную совместимость со сценариями SQL, написанными для предыдущих версий SQL Server.
• FOR ATTACH — указывает, что файлы базы данных уже существуют.

Рассмотрим построение БД в MS SQL на примере.

Построить базу данных для бухгалтерской программы.

Первый способ. Создание БД по умолчанию.

```
CREATE DATABASE test1

/* Данные — 2 Мбайт, файл журнала — по умолчанию */
```

Второй способ. Создание БД с характеристиками: Данные — 2 Мбайт, файл журнала — по умолчанию.

```
CREATE DATABASE test2

ON (FILENAME = 'c:\d1.mdf', SIZE = 2, NAME = 'd1')
```

Третий способ. Создание БД с характеристиками: Первичный файл — 10 Мбайт, одна группа файлов g1 и журнал размером 10 Мбайт.

```
CREATE DATABASE test3

ON PRIMARY (FILENAME ='c:\test3.mdf',

SIZE = 10 , NAME = 'd1'),
```

FILEGROUP g1 (FILENAME — 'c:\g1.mdf',

SIZE = 10 , NAME = 'g1')

LOG ON (FILENAME ='c:\test3.ldf',

SIZE = 10, NAME = 'log1')

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

Построить БД для работы складской программы подбирая оптимальные характеристики.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Какой способ построения БД по Вашему мнению самый быстрый и почему?

2. Какой способ построения БД по Вашему мнению самый эффективный и почему?

3. Какой способ построения БД по Вашему мнению самый самый оптимальный и почему?

**Практическое занятие № 17**

**Тема: Использование принципов нормализации при проектировании базы данных**

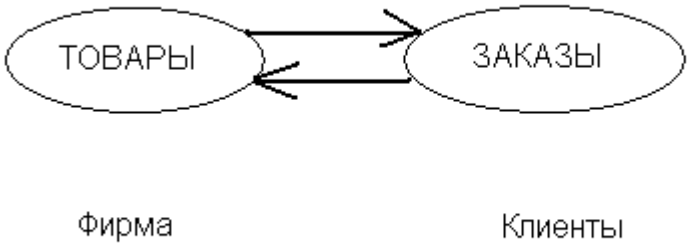
**Цель:** научиться проектировать БД с использованием принципов нормализации.

Пояснения.

Проектирование БД начинается с ответов на вопросы:

1. Какие товары продает фирма, какова их цена, описание, и т.д.
2. По выбранным товарам оформляются заказы. Кто (покупатели), когда и совершили заказ и какова общая сумма заказа?

Таким образом, схему организации фирмы можно представить в виде двух отношений:




Начальная схема отношений.

Далее определяется схема отношений:

ТОВАРЫ-ЗАКАЗЫ

ТОВАРЫ {товарНом, товарНазв, товарОпис, товарРис, товарЦена, колНаСкладе, типНом, типНазв}

ЗАКАЗЫ {заказНом, заказДата, ПокупНом, ФИОПокуп, ЭлпочтаПокуп, адресПокуп, товарНом, количТовЗаказа, видДост, ценаДост, налогНДС, общСумма, заказСост}

В атрибутах «типНом» и «типНазв» содержится информация о типах товаров, например, название типа — Процессоры (типНом = 1), к которому относятся конкретные товары - Intel 386, Intel 486, AMD, и т.д. Атрибут «колНаСкладе» содержит общее количество товаров с данным номером на складе.

В одном заказе может заказываться несколько товаров с разными номерами. Атрибут «количТовЗаказа» содержит количество товаров данного типа, заказанных в данном заказе.

В атрибутах «ПокупНом», «ФИОПокуп», «ЭлпочтаПокуп», «адресПокуп» содержится информация о покупателе — номер, ФИО, Электронная почта и адрес.

В атрибутах «видДост», «ценаДост» хранится информация о доставке — номер, вид и цена доставки. Атрибут «заказСост» содержит состояние заказа, который может быть иметь значение «Сделка совершена» либо «Сделка выполнена неполно».

Для вычисления общей суммы заказа применяем формулу

общСумма = ТоварЦена*количТов + налогНДС + ценаДост.

На 1-м шаге проектирования форма 1NF приводится к виду 2NF.

На 2-м шаге проектирования форма 2NF приводится к виду 3NF.

## Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

## Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

## Задание.

Спроектировать БД с использованием принципов нормализации для работы предприятия (фирмы), занимающейся продажей компьютерной техники используя:

- первую нормальную форму (1 NF);
- вторую нормальную форму (2 NF);
- третью нормальную форму (3 NF);

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Напишите последовательность нормальных форм.

2. Напишите основные свойства нормальных форм.

3. Укажите, когда можно применять нормальную форму Бойса-Кодда.

## Практическое занятие № 18

## Тема: Использование семантических моделей при проектировании базы данных

**Цель:** научиться применять семантическое моделирование при проектировании БД.

Пояснения.

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

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

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

1.Построение ER-диаграммы.

1.1 Выберите из описания предметной области все существительные. Продумайте, какие из них будут соответствовать сущностям, а какие атрибутам сущностей. Зарисуйте в отчет все сущности с их атрибутами согласно обозначениям, принятым в ER-диаграммах.

1.2 На рисунке подчеркиванием атрибутов обозначьте для каждой сущности уникальный идентификатор (Ключ). При необходимости добавьте сущностям атрибуты, которые помогут однозначно отличить каждый экземпляр сущности.

1.3 Определите и включите в схему связи сущностей. Подпишите названия связей и пронумеруйте связи. Для первой связи укажите тип и модальность. Для всех связей запишите их прочтение слева направо и справа налево.

1.4 Если в схеме присутствуют связи типа «много-со-многими» уберите их путем ввода дополнительной сущности. Измененную схему зарисуйте в отчет.

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

По описанию предметной области построить семантическую модель БД методом ER-диаграмм, на основании которой построить набор таблиц БД.

Описание предметной области (Грузоперевозки).

АТП имеет грузовые автомобили с гос. номерами и организует перевозки для своих заказчиков. Стоимость перевозки зависит от расстояния и грузоподъемности автомобиля, который ее выполняет. Каждый заказчик может сделать заказ нескольких перевозок. Одну перевозку выполняет один грузовик.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;

5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Каковы задачи, решаемые на этапе семантического проектирования?

2. Что включает в себя семантическая модель?

3. В чём состоит идея модели «Сущность - связь»?

## Практическое занятие № 19

## Тема: Использование SQL запросов при формировании выборки БД

**Цель:** научится использовать язык запросов SQL для формирования выборки данных из БД.

Пояснения.

Назначение SQL команды SELECT – это выборка и отображение данных из одной или нескольких таблиц базы данных.

Синтаксис команды SELECT:

SELECT [DISTINCT | ALL] {* | [*выражение_столбца* [AS *новое_имя*]] [, …]}
FROM *имя_таблицы* [*псевдоним*] [, …] [WHERE *условие*] [GROUP BY *список_столбцов*] [HAVING *условие*] [ORDER BY *список_столбцов*]

Где: *Выражение_столбца* – это имя столбца или выражение из нескольких имен *Имя_таблицы* – это имя таблицы или представления, из которой нужно

выбрать данные *Псевдоним* – это сокращенное имя таблицы DISTINCT – ключевое слово, для исключения повторяющихся строк из результата (по умолчанию подразумевается ALL).

Порядок конструкций в операторе SELECT не может быть изменен. Обязательными являются конструкции SELECT и FROM. Остальные конструкции могут быть опущены.

***Порядок обработки элементов оператора SELECT:***

1. FROM -Определяются имена используемой таблицы или нескольких таблиц.
2. WHERE – накладывается условие отбора данных.

3. GROUP BY – образуются группы строк, имеющие одинаковые значения в указанном столбце.
4. HAVING – накладывается условие на отбор сгруппированных строк.
5. SELECT – определяются столбцы, которые нужно отобразить в результате.
6. ORDER BY – отобранные данные сортируются по указанным столбцам.

***Конструкция WHERE***

Конструкция WHERE предназначена для ограничения попадания строк в выходные данные, по результату проверки на соответствие определенному условию.

Существуют пять основных условий поиска или предикатов:

| Условие поиска | Ключевые слова, специальные символы |
|---|---|
| Сравнение | >,<,=,>=,<=,<>,!= |
| Диапазон | BETWEEN/NOT BETWEEN |
| Принадлежность к множеству | IN/NOT IN |
| Соответствие шаблону | LIKE/NOT LIKE |
| Значение NULL | IS NULL/IS NOT NULL |

***Конструкция GROUP BY***

Конструкция GROUP BY предназначена для группировки строк. Строки группируются по одинаковым значениям в указанном столбце. В результате для каждой группы строк определяется одна итоговая строка. При использовании конструкции GROUP BY каждый элемент списка SELECT должен иметь единственное значение для всех строк группы. Таким образом элементами списка SELECT при использовании оператора GROUP BY могут быть:

- Имена столбцов (только если они используются так же в конструкции GROUP BY).
- Агрегатные (групповые, агрегирующие) функции (COUNT – количество значений в столбце, SUM – сумма значений в столбце, AVG – усредненное значение в столбце, MIN – минимальное значение в столбце, MAX – максимальное значение в столбце).
- Константы.
- Выражения из перечисленных выше элементов.

***Конструкция HAVING***

Конструкция HAVING предназначена для ограничения групп, помещенных в результирующую таблицу запроса, путем проверки на соответствие определенным условиям. Используется совместно с конструкцией GROUP BY. Имена столбцов, используемые в конструкции HAVING должны присутствовать в конструкции GROUP BY или применяться в агрегатных функциях.

***Примеры простых запросов***

Для примеров используется таблица shop (Магазины) (см. пункт «Задание»).

1. Выбрать всю информацию по всем магазинам:

сервер: ps-345=> пользователь: SELECT * FROM shop;

2. Выбрать всю информацию по магазинам города Челябинска:

сервер: ps-345=> пользователь: SELECT * FROM shop пользователь: WHERE town LIKE 'Челябинск';

3. Выбрать информацию, включающую в себя название магазина, адрес магазина, торговую площадь для всех магазинов города Челябинска. Озаглавить столбцы:

сервер: ps-345=> пользователь: SELECT name as «название магазина», пользователь: adres as «адрес», area as «площадь» пользователь: FROM shop пользователь: WHERE town LIKE 'Челябинск';

4. Определить максимальную торговую площадь в каждом городе. Отсортировать по убыванию торговой площади:

сервер: ps-345=> пользователь: SELECT town as «город»,MAX(area) пользователь: as «максимальная торговая площадь» пользователь: FROM shop пользователь: GROUP BY town пользователь: ORDER BY MAX(area) DESC;

5. Определить количество магазинов в каждом городе. Из результирующего списка исключить города , в которых меньше 2 магазинов. Отсортировать список городов по алфавиту:

сервер: ps-345=> пользователь: SELECT town as «город»,COUNT(shopid)
пользователь: as «количество магазинов» пользователь: FROM shop
пользователь: GROUP BY town

пользователь: HAVING COUNT(shopid)>=2

пользователь: ORDER BY town;

***Подзапросы***

Подзапросы это операторы SELECT, внедренные в тело другого оператора SELECT. Подзапросы могут использоваться в конструкциях WHERE и HAVING. Подзапросы бывают трех видов:

- скалярные – возвращают значение;
- строковые – возвращают строку;
- табличные – возвращают таблицу. Одновременно с подзапросами могут использоваться ключевые слова ALL,

ANY, SOME.

ALL – условие должно соблюдаться для всех строк подзапроса.

ANY, SOME – условие должно соблюдаться хотя бы для одной строки подзапроса.

Так же с подзапросами могут использоваться ключевые слова EXISTS и NOT EXISTS. Эти ключевые слова проверяют наличие строк в результирующей таблице подзапроса. Результат использования этих ключевых слов может быть TRUE или FALSE. Для ключевого слова EXISTS результат равен TRUE, если подзапрос, с которым это ключевое слово использовано возвращает хотя бы одну строку.

Для проверки нахождения некоторой строки в подзапросе используют предикат IN.

***Примеры запросов с подзапросами***

Для примеров используется таблица shop (Магазины) и таблица seller (Продавцы)

Определить магазины, в которых работают сотрудники с именем Татьяна. Указать название магазина, адрес:

сервер: ps-345=>

пользователь: SELECT name as «магазин», adres as «адрес»

пользователь: FROM shop

пользователь: WHERE shopid IN

пользователь: (SELECT shopid

пользователь: FROM seller

пользователь: WHERE fio LIKE '%Татьяна%');

Определить магазины с максимальной торговой площадью:

сервер: ps-345=>

пользователь: SELECT name as «магазин», adres as «адрес»

пользователь: area as «площадь»

пользователь: FROM shop

пользователь: WHERE area=(SELECT MAX(area) FROM shop);

Определить всех сотрудников, работающих в городе Челябинске:

сервер: ps-345=>

пользователь:

SELECT fio as «ФИО»

пользователь:

FROM seller sl

пользователь:

WHERE EXISTS (SELECT * FROM shop s

пользователь:

WHERE sl.shopid=s.shopid

пользователь:

AND town='Челябинск');

*Многотабличные запросы*

Многотабличные запросы - это запросы, которые используют данные нескольких таблиц.

Для того, чтобы объединить в результирующей таблице столбцы нескольких таблиц, используют соединение таблиц и ключевые слова JOIN ON. В результате операции соединения двух таблиц создается таблица, состоящая из пар связанных строк, выбранных из каждой таблицы. Каким образом связываются пары строк определяет условие, которое задается после ключевого слова ON.

SELECT [DISTINCT | ALL] {* | *список столбцов*} FROM *имя_таблицы1* JOIN *имя_таблицы2* ON *условие* Аналогичное действие можно выполнить и таким образом: SELECT [DISTINCT | ALL] {* | *список столбцов*} FROM *имя_таблицы1, имя_таблицы2* WHERE *условие*

Соединение таблиц, это частный случай декартова произведения таблиц. Декартово произведение двух таблиц – это таблица, состоящая из всех возможных пар строк, входящих в состав обеих таблиц. Для задания декартова произведения используются ключевые слова CROSS JOIN.

SELECT [DISTINCT | ALL] {* | *список столбцов*} FROM *имя_таблицы1* CROSS JOIN *имя_таблицы2*

Аналогичное действие можно выполнить и таким образом:

SELECT [DISTINCT | ALL] {* | *список столбцов*} FROM *имя_таблицы1, имя_таблицы2*

Для того чтобы объединить в результирующей таблице строки нескольких таблиц используют объединение таблиц и ключевое слово UNION. Таблицы должны быть совместимы по соединению – т.е. иметь одинаковую структуру (одинаковое количество столбцов, с одинаковыми типами данных).

SELECT [DISTINCT | ALL] {* | *список столбцов*} FROM *имя_таблицы1* WHERE *условие* UNION SELECT [DISTINCT | ALL] {* | *список столбцов*} FROM *имя_таблицы2* WHERE *условие*

***Примеры многотабличных запросов***

Определить всех сотрудников, работающих в городе Челябинске:

сервер: ps-345=> пользователь: SELECT fio as «ФИО» пользователь: FROM seller sl JOIN shop s пользователь: ON sl.shopid=s.shopid пользователь: WHERE town='Челябинск';

Определить количество продавцов в каждом магазине. В итоговой информации указать название магазина, адрес, торговую площадь, количество продавцов.

сервер: ps-345=> пользователь: SELECT name as «магазин», adres as «адрес», пользователь: area as «площадь», c as «количество продавцов» пользователь: FROM shop s JOIN пользователь: (SELECT shopid, COUNT(*) as c пользователь: FROM seller пользователь: GROUP BY shopid) n пользователь: ON s.shopid=n.shopid;

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

Используемая база данных «Сеть магазинов по продажам сумок» Состоит из 7 таблиц:

1. seller (Продавцы)

Атрибут

Ключ

Семантика

Тип данных

Ограничение целостности

1sellerid PK

Уникальный код продавца serial

PRIMARY KEY

2 fio Фамилия, имя, отчество продавца

Varchar (40)

NOT NULL

3 shopid FK (shop.shop.id)

Уникальный код магазина serial

NOT NULL, FOREIGN KEY

2. shop (Магазины)

Атрибут

Ключ

Семантика

Тип данных

Ограничение целостности

1shopid PK

Уникальный код магазина serial

PRIMARY KEY

2 name Название магазина varchar(40)

NOT NULL

3town Город, в котором находится магазин

varchar(20)

NOT NULL, FOREIGN KEY

4 adress

Адрес магазина

Varchar (40)

NOT NULL

5 area Торговая площадь

Numeric (4.1)

NOT NULL, >0

3. sale (Продажи)

1  saleid  PK

Уникальный код продажи  serial

PRIMARY KEY

2 dateprod

Дата продажи  date

DEFAULT NULL

3 goodsid  FK (goods. goodsid)

Уникальный код товара serial

NOT NULL, FOREIGN KEY

4 sellerid

FK (seller. sellerid)

Уникальный код продавца serial

FOREIGN KEY

5 proceeds

Выручка за товар  Numeric(6.2)

NOT NULL  4. supplier (Поставщики)

1 supid PK

Уникальный код поставщика serial

PRIMARY KEY

2 supname

Название поставщика Varchar(40)

NOT NULL

3 suptown

Город поставщика Varchar(20)

NOT NULL

4 supadres

Адрес поставщика Varchar(40)

NOT NULL

5 rating

Рейтинг поставщика Integer

DEFAULT 0

6 note

примечание text

DEFAULT NULL

5. goods (Товары)

1 goodsid PK

Уникальный код товара serial

PRIMARY KEY

2 model FK (model. model)

Название модели Varchar(40)

NOT NULL, FOREIGN KEY

3 cost

Цена закупки

Numeric(6.2)

NOT NULL, >0

4 number

Поставлено всего integer

NOT NULL, >0

5 remain

Количество в наличии integer

>0

5 customid FK (custom. customid)

Уникальный код заказа serial

NOT NULL, FOREIGN KEY

6 shopid FK (shop.shopid)

Уникальный код магазина serial

FOREIGN KEY

7 costplus

Цена для покупателя

Numeric(6.2) NOT NULL, CostPlus>Cost

6. Model (Модели)

1 model PK

Название модели

Varchar(40)

PRIMARY KEY

2 descripti on

Описание text

7. Custom (Заказ)

1 customid

PK Уникальный код заказа serial

PRIMARY KEY

2 nom

Номер заказа

integer

NOT NULL, >0

3 datecust

Дата заказа

date

NOT NULL

4 supid

FK (supplier. supid)

Уникальный код поставщика

serial

NOT NULL, FOREIGN KEY

5 period

Время выполнения заказа (дни)

integer

NOT NULL, >=0

4 datedel

Дата поставки date

DEFAULT NULL

Простые запросы.

1. Выбрать всю информацию по поставщикам. Отсортировать по названию.

2. Выбрать информацию по поставщикам, включающую название, адрес, рейтинг. Отсортировать по убыванию рейтинга.

3. Выбрать всех поставщиков из города Челябинск

4. Выбрать всех поставщиков из города Челябинск и Москва. Информация о поставщике должна включать название поставщика, рейтинг, город.

5. Выбрать всех поставщиков с рейтингом менее 0. Отсортировать по убыванию рейтинга.

## Агрегатные функции

1. Определить среднюю торговую площадь в каждом городе.

2. Определить максимальную торговую площадь в каждом городе.

3. Определить минимальную торговую площадь в каждом городе.

4. Определить суммарную торговую площадь во всех городах, кроме Челябинска.

5. Определить количество магазинов в каждом городе.

## Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

## Контрольные вопросы.

1. Какие основные конструкции входят в оператор SELECT?

2. Можно ли менять порядок конструкций в операторе SELECT?

3. Укажите ключевое слово, предназначенное для исключения повторяющихся строк из результата запроса?

4. В чем отличие конструкций WHERE и HAVING?

5. С помощью какой конструкции выполняется сортировка результата запроса?

6. Какое ключевое слово используется для выполнения сортировки по убыванию?

7. Какие конструкции являются обязательными для оператора SELECT?

## Практическое занятие № 20

## Тема: Использование SQL запросов при сортировке данных в БД

**Цель:** научится использовать язык запросов SQL для формирования сортировки данных в БД.

Пояснения.

Ключевое слово GROUP BY

Ключевое слово GROUP BY используется в операторе SELECT для того, чтобы объединять повторяющиеся значения в группы. Ключевое слово GROUP BY должно следовать за выражением WHERE и предшествовать ключевому слову ORDER BY.

Вот какая должна быть последовательность ключевых слов в операторе, выполняющем запрос:

SELECT FROM WHERE GROUP BY ORDER BY

Ключевое слово GROUP BY должно следовать за условиями в выражении ключевого слова WHERE и предшествовать ключевому слову ORDER BY, если последнее имеется.

SELECT столбец1, столбец2

FROM таблица1, таблица2 WHERE условия

GROUP BY столбец1, столбец2

ORDER BY столбец1, столбец2

Группирование выбранных данных

Группировать данные просто. В выражении ключевого слова GROUP BY могут использоваться только выбранные столбцы (т. е. столбцы из списка ключевого слова SELECT в операторе запроса). Если имя столбца не указано в списке ключевого слова SELECT, то имя этого столбца в выражении ключевого слова GROUP BY использовать нельзя. Это логично — как группировать в отчете данные, которых в нем нет?

Но если столбец выбран, то его имя должно быть включено в выражение ключевого слова GROUP BY. Имя столбца можно представить и его номером, о чем мы поговорим немного позже. При группировании данных порядок группирования столбцов не обязан совпадать с порядком, заданным в выражении ключевого слова SELECT.

К функциям группирования — функциям, используемым в выражении ключевого слова GROUP BY для объединения данных в группы, — относятся AVG, MAX, MIN, зим и COUNT.

Создание групп и использование итоговых функций

При использовании в операторе SELECT ключевого слова GROUP BY должны соблюдаться некоторые правила. В частности, имена выбранных для отображения столбцов должны присутствовать и в выражении ключевого слова GROUP BY, за исключением тех, к которым применены итоговые функции. Столбцы в выражении ключевого слова GROUP BY не обязательно должны быть представлены в том же порядке, что и в выражении ключевого слова SELECT. Но если имя столбца указано в выражении ключевого слова SELECT, имя этого столбца должно присутствовать и в выражении ключевого слова GROUP BY. Вот несколько примеров использования оператора SELECT с ключевым словом GROUP BY.

Пример

SELECT EMP_ID, CITY

FROM EMPLOYEEJTBL

GROUP BY CITY, EMP_ID;

В этом операторе SQL из таблицы EMPLOYEE_TBL выбираются столбцы EMP_ID и CITY, а данные последних выводятся сгруппированными сначала по CITY, а затем по EMP_ID.

Обратите внимание на порядок выбора столбцов и на порядок столбцов в выражении ключевого слова GROUP BY

Пример

SELECT EMP_ID, SUM(SALARY)

FROM EMPLOYEE_PAY_TBL

GROUP BY SALARY, EMP_ID;

Этот оператор SQL возвращает данные столбца EMP_ID и сумму по группам зарплат, созданным по величине зарплаты (SALARY) и табельному номеру (EMP_ID).

Пример

SELECT SUM(SALAPY)

FROM EMPLOYEE_PAY_TBL;

Здесь оператор SQL возвращает сумму всех выплат по зарплате из таблицы

EM PLOYEE_PAY_TBL.

Пример

SELECT SUM(SALARY)

FROM EMPLOYEE_PAY_TBL

GROUP BY SALARY;

Здесь оператор SQL возвращает суммы по группам, созданным по всем уровням зарплаты.

Вот несколько примеров с использованием реальных данных. Сначала убедимся, что в таблице EMPLOYEE_TBL представлены три города.

Ввод:

SELECT CITY

FROM EMPLOYEEJTBL ;

Вывод:

CITY

GREENWOOD

INDIANAPOLIS

WHITELAND

INDIANAPOLIS

INDIANAPOLIS

INDIANAPOLIS

В следующем примере подсчитывается число записей по каждому городу. Именно из-за того, что используется ключевое слово GROUP BY, вы здесь видите результаты по каждому из городов в отдельности.

Ввод:

```
SELECT CITY, COUNT(*)

FROM EMPLOYEE_TBL

GROUP BY CITY;
```

Вывод:

```
CITY COUNT(*)

GREENWOOD 1

INDIANAPOLIS 4

WHITELAND 1
```

Следующий запрос осуществляет выборку из временной таблицы, созданной на основе таблиц EMPLOYEE_TBL и EMPLOYEE_PAY_TBL. О том как объединить две таблицы в одном запросе, мы поговорим чуть позже.

Ввод:

```
SELECT *

FROM EMP_PAY_TMP;
```

Вывод:

```
CITY LAST_NAME FIRST_NAME PAY_RATE SALARY

GREENWOOD STEPHENS TINA 30000

INDIANAPOLIS PLEW LINDA 14.75

WHITELAND GLASS BRANDON 40000

INDIANAPOLIS GLASS JACOB 20000

INDIANAPOLIS WALLACE MARIAH 11

INDIANAPOLIS SPURGEON TIFFANY 15
```

В следующем примере с помощью функции AVG извлекаются средние значения для почасовой оплаты и зарплаты по каждому городу в отдельности. Для городов Гринвуд и Уайтленд среднего значения почасовой оплаты нет,

поскольку работа ни одного из представленных в таблице служащих из этих городов не оплачивается почасово.

Ввод:

```
SELECT CITY, AVG(PAY_RATE), AVG(SALARY)

FROM EMP_PAY_TMP

GROUP BY CITY;
```

Вывод:

```
CITY AVG(PAY_RATE) AVG(SALARY}

GREENWOOD 30000

INDIANAPOLIS 13.5833333 20000

WHITELAND 40000
```

В следующем примере для группирования данных комбинируется использование нескольких компонентов запроса. Необходимо получить опять же средние значения для почасовой оплаты и зарплаты, но только для городов Индианаполис и Уайтленд. Для этого данные группируются по полю CITY — другого выбора здесь нет, поскольку иначе из выбранных столбцов используется итоговая функция. Наконец, отчет упорядочивается сначала по столбцу 2, а затем по столбцу 3, т. е. по средней почасовой оплате и средней зарплате. Попытайтесь до конца разобраться в показанном ниже операторе и выведенных данных.

Ввод:

```
SELECT CITY, AVG(PAY_RATE), AVG(SALARY)

FROM EMP_PAY_TMP

WHERE CITY IN ('INDIANAPOLIS','WHITELAND')

GROUP BY CITY

ORDER BY 2,3;
```

Вывод:

```
CITY AVG(PAY_RATE) AVG(SALARY)
```

INDIANAPOLIS 13.5833333 20000

WHITELAND 40000

Значения сортируются так, что значения NULL оказываются в конце. Поэтому запись для города Индианаполис представлена первой. Город Гринвуд не был выбран, но если бы был, то соответствующая ему запись была бы представлена перед записью для Уайтленда, поскольку для Гринвуда средняя зарплата (SALARY) равна 30000, а средняя зарплата является вторым параметром сортировки в выражении ключевого слова ORDER BY.

В завершение раздела рассмотрим использование в выражении ключевого слова ORDER BY итоговых функций MAX и MIN.

Ввод:

SELECT CITY, MAX(PAY_RATE), MIN(SALARY)

FROM EMP_PAY_TMP

GROUP BY CITY;

Вывод:

CITY MAX(PAY_RATE) MIN(SALARY)

GREENWOOD 30000

INDIANAPOLIS 15 20000

WHITELAND 40000

Представление имен столбцов числами

В отличие от выражения ключевого слова ORDER BY, в выражении ключевого слова GROUP BY указать порядок столбцов с помощью их номеров нельзя — за исключением того случая, когда используется ключевое слово UNION и имена всех столбцов разные. Вот пример использования номеров вместо имен столбцов.

SELECT EMP_ID, SUM(SALARY)

FROM EMPLOYEE_PAY_TBL

UNION

SELECT EMP_ID, SUM(PAY_RATE)

```
FROM EMPLOYEE_PAY_TBL

GROUP BY 2, 1;
```

Этот оператор SQL возвращает табельный номер служащего (EMP_ID) и группирует суммы по значениям зарплаты. При использовании ключевого слова UNION результаты двух операторов SELECT объединяются. Группирование выполняется сначала по столбцу 2Б, представляющем зарплату (SALARY), а затем по столбцу 1, представляющем табельный номер служащего (EMP_ID).

GROUP BY И ORDER BY

Обратите внимание на то, что GROUP BY и ORDER BY работают одинаково в том смысле, что оба эти ключевые слова задают сортировку данных. В выражении ключевого слова ORDER BY задается сортировка данных запроса, а в выражении ключевого слова GROUP BY — сортировка этих данных по группам. Поэтому ключевое слово GROUP BY можно использовать для сортировки точно так же, как и ORDER BY.

Вот несколько особенностей использования ключевого слова GROUP BY для сортировки.

• Все выбранные столбцы, к которым не применяются итоговые функции, должны быть указаны в списке ключевого слова GROUP BY.

• В отличие от выражения ключевого слова ORDER BY, в выражении ключевого слова GROUP BY имена столбцов нельзя заменить числами.

• Использовать ключевое слово GROUP BY вообще нет необходимости, если не используются итоговые функции.

Вот пример использования для сортировки данных ключевого слова GROUP BY вместо ключевого слова ORDER BY:

Ввод:

```
SELECT LAST_NAME, FIRST_NAME, CITY

FROM EMPLOYEE_TBL

GROUP BY LAST_NAME;
```

Вывод:

```
SELECT LAST_NAME, FIRST_NAME, CITY

*
```

ERROR at line 1:

ORA-00979: not a GROUP BY expression

В этом примере сервер базы данных сообщает об ошибке из-за того, что имя столбца FIRST_NAME не указано в выражении ключевого слова GROUP BY. Помните о том, что все столбцы из списка ключевого слова SELECT должны быть указаны в выражении ключевого слова GROUP BY, за исключением тех столбцов, к которым применяются итоговые функции.

В следующем примере проблема предыдущего оператора решена путем добавления в список ключевого слова GROUP BY недостающих имен из списка ключевого слова SELECT.

Ввод:

SELECT IAST_NAME, FIRST_NAME, CITY

FROM EMPLOYEE_TBL

GROUP BY LAST_NAME, FIRST_NAME, CITY;

Вывод:

LAST_NAME FIRST_NAME CITY

GLASS BRANDON WHITELAND

GLASS JACOB INDIANAPOLIS

PLEW LINDA INDIANAPOLIS

SPURGEON TIFFANY INDIANAPOLIS

STEPHENS TINA GREENWOOD

WALLACE MARIAH INDIANAPOLIS

В этом примере выбираются те же данные из той же таблицы, но уже все столбцы перечислены в выражении ключевого слова GROUP BY в том же порядке, в каком они указаны в списке ключевого слова SELECT. В результате данные показаны отсортированными сначала по столбцу LAST_NAME, затем по столбцу FIRST_NAME и наконец, по столбцу CITY. С помощью ключевого слова ORDER BY то же самое получить легче, но для правильного использования ключевого слова GROUP BY, наверное, будет полезно разобраться, как с его помощью сортируются данные, перед тем, как выполнить группирование результатов.

В следующем примере получается выборка из таблицы EMPLOYEE_TBL и используется ключевое слово GROUP BY для упорядочения данных по значениям столбца CITY.

Ввод:

```
SELECT CITY, LAST_HAME

FROM EMPLOYEEJTBL

GROUP BY CITY, LAST_NAME;
```

Вывод:

```
CITY LAST_NAME

GREENWOOD STEPHENS

INDIANAPOLIS GLASS

INDIANAPOLIS PLEW

INDIANAPOLIS SPURGEON

INDIANAPOLIS WALLACE

WHITELAND GLASS
```

Сравните порядок представленных здесь данных с порядком в предыдущих примерах. Здесь учтены все записи таблицы EMPLOYEE_TBL, результаты группируются по значениям столбца CITY, но упорядочены сначала по числу записей для каждого города.

Ввод:

```
SELECT CITY, COUNT(*)

FROM EMPLOYEEJTBL

GROUP BY CITY

ORDER BY 2,1;
```

Вывод:

CITY COUNT(*)

GREENWOOD 1

WHITELAND 1

INDIANAPOLIS 4

Обратите внимание на порядок представления полученных данных. Они сначала отсортированы по числу записей для каждого города и только потом по названиям городов. Для первых двух городов число записей равно 1. Поскольку числа совпадают, порядок представления записей определяется вторым параметром сортировки, а именно, названием города. Поэтому Гринвуд идет перед Уайтлендом.

Хотя ключевые слова GROUP BY и ORDER BY и функционируют подобным образом, между ними имеется существенное отличие. Ключевое слово GROUP BY предназначено для группирования одинаковых значений, а задачей ORDER BY является представление данных просто в определенном порядке. Ключевые слова GROUP BY и ORDER BY можно использовать в одном операторе SELECT, но каждое из них должно выполнять свою задачу. В одном операторе SELECT ключевое слово GROUP BY должно предшествовать ключевому слову ORDER BY.

Ключевое слово GROUP BY можно использовать для сортировки данных в операторе CREATE VIEW, а вот ключевое слово ORDER BY использовать в операторе CREATE VIEW нельзя.

## Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

## Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

## Задание.

1. Открыть БД «Сеть магазинов по продаже сумок».

2. Выполнить сортировку данных по ФИО продавца.

3. Выполнить сортировку данных по названию магазина.

4. Выполнить сортировку по выручке за товар.

5. Выполнить сортировку по рейтингу поставщика.

6. Выполнить сортировку по количеству товара в магазине.

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Какие способы сортировки данных существуют?

2. Для чего используется ключевое слово Group By.

3. С какими итоговыми функциями SQL можно использовать ключевое слово Group By.

4. В чём отличия и сходства между ключевыми словами Group By и Order By?

5. Для чего используется ключевое слово Union?

## Практическое занятие № 21

## Тема: Использование триггеров в БД.

**Цель:** научится использовать триггеры для формирования БД.

Пояснения.

Триггер – это хранимая процедура особого типа, вызываемая на выполнение в ответ на определённые события. Триггеры подразделяются на два основных типа: триггеры языка определения данных ( DDL) и триггеры языка манипулирования данными ( DML)

Триггеры DDL активизируются в ответ на внесение каких-либо изменений в структуру б.д пользователями ( создание, удаление, изменение структуры таблиц,).

Триггеры DML представляют собой фрагменты кода, которые закрепляются за конкретной таблицей.

В отличие от хранимых процедур, при использовании которых необходимо явно вызывать на выполнение определённый код, триггеры вызываются на выполнение автоматически при обнаружении события, связанного с ним. Триггеры не получают параметров и не возвращают значений.

Конструкция FOR ( или AFTER) позволяет указать какое действие приводит к запуску триггера.

Триггер Insert этот триггер вызывается на выполнение каждый раз, когда вставляется новая строка в таблицу, за которой закреплён триггер. Для каждой новой строки создаётся её копия и вставляется в таблицу INSERTED. Эта таблица сохраняется до тех пор пока действует триггер ( с момента запуска до его завершения).

Триггер Delete вызывается на выполнение каждый раз, при удалении записи. Копия удаляемой записи помещается во временную таблицу Deleted.

Триггер Update вызывается на выполнение при обновлении записи таблицы, для которой он создан. операция модификации строки трактуется как удаление старой версии строки ( помещение её в Deleted) и добавление новой строки ( помещение её в INSERTED). При этом триггеры на удаление и добавление не запускаются.

*Условный оператор для триггера в языке SQL-Transact*

Один оператор

Несколько операторов

IF условие

Оператор1

Else

Оператор2

**IF** условие

*begin*

Оператор1

Оператор2

*end*

**Else**

*begin*

Оператор3

Оператор4

*end*

Пример триггера:

//Определение триггера

CREATE TRIGGER Add_copies BEFORE

INSERT ORDER 1

ON Stepanov.Books

REFERENCING NEW AS New_books

FOR EACH ROW

/* Предваряющий строчный триггер, активизирующийся при наступлении события "Добавление новой записи" в таблицу Stepanov.Books. В этой таблице имеются три поля, для двух из которых определены значения по умолчанию. Поэтому в операторе INSERT достаточно ввести значения только одного поля - Code_book */

BEGIN

DECLARE Kol SMALLINT;

SET Kol=New_books.number;

/* MESSAGE 'Code_book=',

New_books.Code_book; */

```
WHILE Kol>0 LOOP

INSERT INTO Stepanov.Copies(

Code_book)

VALUES(

New_books.Code_book);

SET Kol=Kol-1

END LOOP

END
```

Необходимые принадлежности

1. Персональный компьютер с программным обеспечением.

2. Справочная литература

Работа в аудитории.

1. Разработать БД согласно заданию.

2. Составить письменно отчёт и продемонстрировать преподавателю.

Задание.

Создайте триггер на изменение зарплаты сотрудника, проверяющий, находится ли она в заданных должностных рамках. Рамки зарплаты берутся из таблицы JOBS. Если назначенная зарплата меньше минимально возможной, то установить сотруднику минимально возможную зарплату (min_salary). И наоборот, если назначенная зарплата больше максимально возможной, то установить сотруднику максимально возможную зарплату (max_salary).

Содержание отчета

1. Номер и название работы;
2. Цель работы;
3. Задание с исходными данными;
4. Необходимые принадлежности;
5. Проект БД;
6. Заключение

Контрольные вопросы.

1. Дать определение понятию триггер.

2. Какие виды триггеров существуют?

3. Чем триггеры отличаются от процедур?

4. Перечислите часто используемые триггеры.