ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ, НАУКИ И МОЛОДЕЖНОЙ ПОЛИТИКИ ВОРОНЕЖСКОЙ ОБЛАСТИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ ВОРОНЕЖСКОЙ ОБЛАСТИ «ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ ПРОМЫШЛЕННО-ГУМАНИТАРНЫЙ КОЛЛЕДЖ»

ОСНОВЫ ПРОГРАММИРОВАНИЯ И БАЗ ДАННЫХ

Методические рекомендации и контрольные задания

Для студентов с инвалидностью по специальности 09.02.02 «Компьютерные сети» заочной формы обучения

Составители: Е.Н. Рысцова, А. Е. Овсянникова, А. А. Руднева

Воронеж

В результате освоения учебной дисциплины **ОП.05 «Основы программирования и баз данных»** обучающий с инва-

лидностью должен обладать предусмотренными ФГОС по специальности СПО **09.02.02_ «Компьютерные сети» (базовая подготовка)** следующими умениями, знаниями, которые формируют профессиональную компетенцию, и общими компетенциями:

уметь:

- использовать языки программирования высокого уровня;
- строить логически правильные и эффективные программы;
- использовать язык SQL для программного извлечения сведений из баз данных.

знать:

- общие принципы построения алгоритмов;
- основные алгоритмические конструкции;
- системы программирования;
- технологии структурного и объектно – ориентированного программирования;
- основы теории баз данных;
- модели баз данных;
- основы реляционной алгебры;
- принципы проектирования баз данных;
- средства проектирования структур баз данных;
- язык запросов SQL.

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество.

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития.

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации.

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности.

ПК 2.2. Администрировать сетевые ресурсы в информационных системах.

ПК 2.3. Обеспечивать сбор данных для анализа использования и функционирования программно-технических средств компьютерных сетей.

ПК 3.1. Устанавливать, настраивать, эксплуатировать и обслуживать технические и программно-аппаратные средства компьютерных сетей.

ПРАВИЛА ВЫПОЛНЕНИЯ И ОФОРМЛЕНИЯ ПРАКТИЧЕСКИХ РАБОТ

1. Практические работы (ПР) выполняются студентами 3-го курса в период изучения дисциплины «Основы программирования и баз данных» , а также может быть использована для студентов заочной формы обучения при изучении дисциплины «Основы алгоритмизации и программирования».
2. ПР выполняется в следующем порядке
   - выполняется пп.1 практической части и сдается преподавателю;
   - выполняется пп.2 практической части и сдается на проверку преподавателю в печатном виде в форме отчета, выполненного в формате А4.
3. Срок выполнения и сдачи определяется преподавателем.
4. Вариант ПР назначается соответственно порядковому номеру студента в журнале посещаемости.
5. Каждый вариант ПР содержит задачу, контрольные вопросы по теме.
6. В соответствии с пп.1 каждой задачи должна быть составлена программа на языке программирования (шестой пункт

оформления) и выполнена на компьютере, при компилировании программа не должна выдавать ошибок.

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ К ИЗУЧЕНИЮ КУРСА

При наличии хронических соматических заболеваний основным патопсихологическим механизмом формирования соматогенного астенического симптомокомплекса является изменение биосоциального статуса личности в результате соматического заболевания как фактора, независящего от субъективно-волевой сферы инвалида. У больных этой группы невротические жалобы появляются незаметно, постепенно, спустя некоторое время после диагностирования соматического заболевания, без значимых побочных психотравмирующих влияний.

Частота и выраженность соматогенного астенического симптомокомплекса растёт с увеличением давности соматического заболевания и степени его тяжести. Астеническая симптоматика усиливается в периоды обострения соматического заболевания и, наоборот, смягчается, а в ряде случаев даже исчезает при улучшении соматического состояния больных.

Первые признаками невротических нарушений: физическая и психическая астения, общая слабость, повышенная утомляемость, ощущение обессиливания, снижение работоспособности и концентрации внимания, ухудшение памяти, невнимательность, постсомнические нарушения.

Постепенно появляются расстройства эмоционально-волевой сферы: болезненная раздражительность, вспыльчивость, колебание настроения от мрачно-пессимистического до обычного, ровного.

Впоследствии в эмоциональной сфере превалирующими становятся негативные эмоции, усиливается подавленность, снижение настроения, формируется ощущение тоски, внутреннего дискомфорта с элементами тревожности. Организация учебного пространства и рабочего места должна жестко соответствовать всем санитарно-гигиеническим требованиям, предъявляемым к

домашнему рабочему месту учащегося, оборудованному компьютером.

При наличии отдельных заболеваний требуется специальная организация помещения и рабочего места. Так, помещения для инвалидов с заболеванием туберкулезом желательно ориентировать на солнечную сторону. В них следует обеспечить повышенную кратность воздухообмена, при этом рециркуляция воздуха не допускается. В помещениях для инвалидов вследствие туберкулезных заболеваний отделочные материалы пола и стен следует выбирать с учетом обеспечения влажной уборки и дезинфекции.

Помещения для инвалидов вследствие заболеваний сердечно-сосудистой системы желательно ориентировать на теневую сторону, при невозможности соблюдения этого требования необходимо применение солнцезащитных устройств. Рабочие места инвалидов данной группы при их расположении в непосредственной близости от окон должны быть защищены от перегрева в летнее время солнцезащитными устройствами. Временной режим обучения - щадящий.

Требуется чередование умственной и физической нагрузки для того, чтобы избежать чрезмерного переутомления обучающегося. Технические средства обеспечения комфортного доступа к образованию - те же, что и при организации дистанционных курсов для обучающихся без ограничений в здоровье. Учебные материалы (учебники, рабочие тетради и дидактические материалы) - те же, что у обучающихся без ограничений в здоровье. Формирование у педагогических работников готовности к обучению лиц с ограниченными возможностями здоровья Решение задачи расширения доступности качественного профессионального образования лиц с ОВЗ будет успешным только при условии специальной подготовки к их обучению педагогических работников организации.

Такая подготовка предполагает формирование у преподавателей общеобразовательных и специальных дисциплин, мастеров производственного обучения реабилитационной направлен-

ности профессионально-педагогической деятельности. Реабилитационная направленность является необходимой составляющей профессиональной компетентности современного педагога, обучающего лиц с ОВЗ, и представляет собой сплав определенных психолого-педагогических установок с междисциплинарными знаниями, умениями и опытом реализации задач выявления, профилактики и преодоления барьеров и затруднений, возникающих в процессе обучения данной социальной группы.

Необходимый и достаточный уровень сформированности реабилитационной направленности помогает педагогическим работникам организации компетентно решать задачи, связанные с:

-распознаванием затруднений в учебной и учебно-профессиональной деятельности лиц с ОВЗ, установлением их причин;

-проектированием индивидуального образовательного маршрута для обучающихся с ОВЗ в рамках учебной дисциплины (ее отдельного раздела, темы) или направления профессиональной подготовки;

-оптимальным выбором методов и приемов организации учебной и учебно-производственной деятельности обучающихся с ОВЗ;

-объективным анализом текущих и этапных результатов усвоения учебных программ лицами с ОВЗ;

-созданием условий для их социализации и социально-трудовой интеграции.

ЯЗЫКИ И СИСТЕМЫ ПРОГРАММИРОВАНИЯ

Название «алгоритм» произошло от латинской формы имени среднеазиатского математика аль-Хорезми — Algorithmi.

Алгоритм — одно из основных понятий информатики и математики.

Язык программирования Pascal был разработан в 1968—1971 гг. Никлаусом Виртом в Цюрихском Институте информатики (Швейцария). Первоначальная цель разработки языка диктова-

лась необходимостью инструмента "для обучения программированию как систематической дисциплине". Однако очень скоро обнаружилась чрезвычайная эффективность языка Pascal в самых разнообразных приложениях, от решения небольших задач численного характера до разработки сложных программных систем — компиляторов, баз данных, операционных систем и т.п. К настоящему времени Pascal принадлежит к группе наиболее распространенных и популярных в мире языков программирования. Существуют многочисленные реализации языка практически для всех машинных архитектур; разработаны десятки диалектов и проблемно-ориентированных расширений языка Pascal; обучение программированию и научно-технические публикации в значительной степени базируются на этом языке.

Характеристика и особенности языка. Существует ряд объективных причин, обусловивших выдающийся успех языка Pascal. Среди них в первую очередь необходимо указать следующие:

- Язык в естественной и элегантной форме отразил важнейшие современные концепции технологии разработки программ: развитая система типов, ориентация на принципы структурного программирования, поддержка процесса пошаговой разработки.
- Благодаря своей компактности, концептуальной целостности и ортогональности понятий, а также удачному первоначальному описанию, предложенному автором языка, Pascal оказался весьма легок для изучения и освоения. В противоположность громоздким многотомным описаниям таких языков, как PL/I, Cobol, FORTRAN, достаточно полное описание языка Pascal занимает около 30 страниц текста, а его синтаксические правила можно разместить на одной странице.
- Несмотря на относительную простоту языка, он оказался пригоден для весьма широкого спектра приложений, в том числе для разработки очень больших и сложных программ, например операционных систем.

Pascal весьма технологичен для реализации практически для всех, в том числе и нетрадиционных, машинных архитектур. Утверждается, что разработка Pascal-транслятора "почти не превышает

по трудоемкости хорошую дипломную работу выпускника вуза". Благодаря этому для многих ЭВМ существует несколько различных реализации языка, отражающих те или иные практические потребности программистов.

Язык Pascal стандартизован во многих странах. В 1983 году был принят международный стандарт (ISO 7185:1983).

Рассмотрим основные особенности языка Pascal.

1. Pascal является традиционным алгоритмическим языком программирования, продолжающим линию Algol-60. Это означает, что программа на языке Pascal представляет собой специально организованную последовательность шагов по преобразованию данных, приводящую к решению некоторой задачи. Это отличает Pascal от так называемых непроцедурных языков типа Prolog, по существу представляющих собой формализмы для записи начальных условий некоторой задачи и синтезирующих решение посредством встроенных механизмов логического вывода.

2. Язык Pascal содержит удобные средства для представления данных. Развитая система типов позволяет адекватно описывать данные, подлежащие обработке, и конструировать структуры данных произвольной сложности. Pascal является типизированным языком, что означает фиксацию типов переменных при их описании, а также строгий контроль преобразований типов и контроль доступа к данным в соответствии с их типом (как на этапе компиляции, так и при исполнении программ).

3. Набор операторов языка Pascal отражает принципы структурного программирования и позволяет записывать достаточно сложные алгоритмы в компактной и элегантной форме.

Pascal является процедурным языком с традиционной блочной структурой и статически определенными областями действия имен. Процедурный механизм сочетает в себе простоту реализации и использования и гибкие средства параметризации.

4. Синтаксис языка достаточно несложен. Программы записываются в свободном формате, что позволяет сделать их наглядными и удобными для изучения.

СТРУКТУРА ПРОГРАММЫ. ОСНОВНЫЕ КОМПОНЕНТЫ ЯЗЫКА PASCAL

Программы на языке Паскаль имеют блочную структуру:

1. Блок типа PROGRAM – имеет имя, состоящее только из латинских букв и цифр. Его присутствие не обязательно, но рекомендуется записывать для быстрого распознавания нужной программы среди других листингов.

2. Программный блок, состоящий в общем случае из 7 разделов:

•раздел описания модулей (uses);
•раздел описания меток (label);
•раздел описания констант (const);
•раздел описания типов данных (type);
•раздел описания переменных (var);
•раздел описания процедур и функций;
•раздел описания операторов.

Общая структура программы на языке Паскаль следующая:

```
Program ИМЯ..; {заголовок программы}
Uses ...; {раздел описания модулей}
Var ..; {раздел объявления переменных}
...
Begin {начало исполнительной части программы}
... {последовательность
... операторов}
End. {конец программы}
```

Правила языка

- Использовать буквы латинского алфавита
- Выделять в операторные скобки несколько операторов для одновременного их выполнения
- В конце оператора ставить точку с запятой
- Смысловые части выделять одинаковым отступлением от

начала строки

Основные понятия языка Паскаль

Оператор	- представляет собой законченную фразу языка и определяет некоторый вполне законченный этап обработки данных
Основной оператор	- в своем составе не содержит других операторов (присвоение, оператор перехода, пустой оператор)
Производные операторы	- в состав которых ходят другие операторы (составной оператор, выбирающий, оператор цикла, присоединения)

Данные

Константы	Переменные
Не изменяются в процессе выполнения программы	Значения возникают и могут изменяться в процессе выполнения программы

Простые типы переменных

Integer	Целочисленные данные, во внутреннем представлении занимают 2 байта
Real	Вещественные данные, занимают 6 байтов
Char	Символ, занимает 1 байт
String	Строка символов, занимает MAX+1байт, где MAX – максимальное число символов в строке

Boolean	Логический тип, занимает 1 байт имеет два значения: false (ложь) и true (истина)

Оператор ввода

Для ввода данных предназначен оператор:

- **read** (a1, a2, a3, … , an);

a1…an – переменные, которые последовательно принимают значения, вводимые с клавиатуры.

- **readln** (a1, a2, a3, … , an);

a1…an – переменные, которые последовательно принимают значения, вводимые с клавиатуры и после этого происходит переход на новую строку.

- **readln;**

переход на новую строку

Оператор вывода

Для вывода данных предназначен оператор:

- **write** (b1, b2, b3, … , bn);

(b1, b2, b3, … , bn) – выводятся на экран значения переменных

- **writeln** (b1, b2, b3, … m bn);

выводятся на экран значения переменных и после этого переход на новую строку

- **writeln;** осуществляется переход на новую строку.

Оператор присваивания

В операторе присваивания используется знак присваивания **:=**

МАССИВЫ

Определение. Массив – это совокупность объектов, состоящая из фиксированного упорядоченного числа элементов, имеющих один и тот же тип.

Массивы могут быть одномерными и многомерными (двух-,

трехмерными и т. д.). Примером одномерных массивов может быть список фамилий учеников класса, многомерных - таблица умножения, классный журнал, аттестат зрелости.

Элементы, образующие массив, упорядочены таким образом, что каждому элементу соответствует номер (индекс), определяющий его местоположение в общей последовательности. Доступ к каждому элементу осуществляется путём индексирования.

Для описания массива используется словосочетание array of (массив из) и имеет вид:

array [тип индекса] of <тип>

Тип индекса – любой порядковый номер, определяющий границы изменения значений индекса.

Описание массива задается следующим образом:

<имя типа> = array [тип индекса] of <тип данных>;

Например,

```
Program Name;
Const
  m=50;
Type
  mas=array [1..m] of integer; {массив из m целых чисел}
  digit = array [0 .. 9] of char; {массив десяти символов, имеющих порядковые номера от 0 до 9}
  matrix = array [byte] of string; {массив 256 строк, пронумерованных с 0 до 255}
Var
  massiv: mas;
  m: matrix;
  d: digit;
  a: array [1..n] of real; {явное описание переменной типа массив}
```

В качестве индексных типов можно использовать любые порядковые типы, кроме Longint и типов-диапазонов с базовым типом Longint.

Размер двумерного массива задается парой чисел: M*N, где

M – число строк, а N – число столбцов в таблице.

Пусть задан двумерный массив Matr, имеющий размер 10*20. Этот массив на языке Паскаль может быть описан следующим образом:

```
Var
Matr : array [1..10,1..20] of integer;
```

тогда Matr[5,7] – элемент, расположенный в 5-ой строке и в 7-ом столбце.

Любая константа, переменная, значение функции или выражения в Турбо Паскале характеризуется своим типом. Тип любого из этих объектов определяет множество допустимых значений, которые может иметь объект, а также множество допустимых операций, которые применимы к объекту. Кроме того, тип определяет и формат внутреннего представления значения объекта.

Имя, которое программист присваивает своему определяемому типу, – произвольный идентификатор. Объявление типа должно быть сделано в разделе объявлений, и ему должно предшествовать кодовое слово Type.

Отличительной особенностью массивов является то обстоятельство, что все их компоненты суть данные одного типа (возможно, структурированного); эти компоненты можно легко упорядочить и обеспечить доступ к любому из них простым указанием его порядкового номера, например:

```
Type
Digit = array [0..9] of integer;
Matrix = array [1..100, 0..9] of real;
Var
m : Matrix;
d : Digit;
i : integer;
```

Описание типа массива задается следующим образом:

<имя типа> = array [<сп. инд. типов>] of <тип>;

где:

<имя типа> – правильный идентификатор,

<сп. инд. типов> – список из одного или нескольких индексных типов, разделенных запятыми,

<тип> – любой тип Турбо Паскаля.

Комбинированный тип данных. Запись.

Довольно часто вполне оправданным является представление некоторых элементов в качестве составных частей другой, более крупной логической единицы. представляется естественным сгруппировать информацию о номере дома, названии улицы и городе в единое целое и назвать адресом, а объединенную информацию о дне, месяце и годе рождения – датой. В языке Паскаль для представления совокупности разнородных данных служит комбинированный тип запись.

Запись и массив схожи в том, что обе эти структуры составлены из ряда отдельных компонент. В то же время, если компоненты массива должны быть одного типа, записи могут содержать компоненты разных типов.

Приведем пример описания переменной, имеющей структуру записи:

```
Var
Address : Record
HouseNumber : Integer;
StreetName : String[20];
CityName : String[20];
PeopleName : String;
End;
```

Отметим, что поля StreetName и CityName имеют одинаковый тип: String[20]. Поскольку в описании эти поля могут располагаться в любом порядке, то можно сократить описание записи с полями одинакового типа.

Сокращенное описание записи Address выглядит следующим образом:

```
Var
Address : Record
HouseNumber : Integer;
StreetName, CityName : String[20];
PeopleName : String;
End;
```

Каждая компонента записи называется полем. В переменной записи Address поле с именем HouseNumber само является переменной типа Integer, поле StreetName – двадцатисимвольной строкой и т.д.

Для того чтобы обратиться к некоторому полю записи, следует написать имя переменной и имя поля. Эти два идентификатора должны разделяться точкой.

Оператор, который присваивает полю HouseNumber значение 45, выглядит так:

```
Address.HouseNumber := 45;
```

Таким же образом присваиваются значения другим полям записи Address :

```
Address.StreetName := 'Профсоюзная';
Address.CityName := 'Сургут';
Address.PeopleName := 'Петрова Алла Ивановна'.
```

Каждое поле записи Address можно рассматривать как обычную переменную, которую можно напечатать или использовать в расчетах. Вместе с тем запись может использоваться как единое целое. В этом случае надо ввести тип записи.

Понятие подпрограммы. Процедуры и функции.

Определение. Подпрограмма – это отдельная функционально независимая часть программы. Любая подпрограмма обладает той же структурой, которой обладает и вся программа.

Подпрограммы решают три важные задачи:

- избавляют от необходимости многократно повторять в тексте программы аналогичные фрагменты;
- улучшают структуру программы, облегчая ее понимание;

- повышают устойчивость к ошибкам программирования и непредвидимым последствиям при модификациях программы.

Очень важно понимать, что в подпрограммы выделяется любой законченный фрагмент программы. В качестве ориентиров просмотрите следующие рекомендации.

- Когда Вы несколько раз перепишите в программе одни и те же последовательности команд, необходимость введения подпрограммы приобретает характер острой внутренней потребности.
- Иногда слишком много мелочей закрывают главное. Полезно убрать в подпрограмму подробности, заслоняющие смысл основной программы.
- Полезно разбить длинную программу на составные части – просто как книгу разбивают на главы. При этом основная программа становится похожей на оглавление.
- Бывают сложные частные алгоритмы. Полезно отладить их отдельно в небольших тестирующих программах. Включение программ с отлаженными алгоритмами в основную программу будет легким, если они оформлены как подпрограммы.
- Все, что Вы сделали хорошо в одной программе, Вам захочется перенести в новые. Для повторного использования таких частей лучше сразу выделять в программе полезные алгоритмы в отдельные подпрограммы.

Подпрограммы могут быть стандартными, т.е. определенными системой, и собственными, т.е. определенными программистом.

Стандартная подпрограмма (процедура или функция) - подпрограмма, включенная в библиотеку программ ЭВМ, доступ к которой обеспечивается средствами языка программирования. Вызывается она по имени с заданием фактических параметров с типом описанным при описании данной процедуры в библиотечке процедур и функций.

Из набора стандартных процедур и функций по обработке одного типа информации составляются модули. Каждый модуль

имеет своё имя (мы уже хорошо знакомы с модулями Crt, Graph). Доступ к процедурам и функциям модуля осуществляется при подключении этого модуля (Uses Crt, Graph).

Любая подпрограмма должна быть описана до того, как она будет вызвана в программе или в другой подпрограмме. Все переменные, которые использует подпрограмма, могут быть либо глобальные либо локальные.

Определение. Глобальными называются переменные, объявленные в основной программе и доступные как программе, так и всем ее подпрограммам.

Определение. Локальными называются переменные, объявленные внутри подпрограммы и доступные только ей самой.

Обмен информацией между основной программой и подпрограммой может осуществляться только с помощью глобальных переменных.

В языке Паскаль выделяют два вида подпрограмм: процедуры (Procedure) и функции (Function). Любая программа может содержать несколько процедур и функций. Структура любой подпрограммы аналогична структуре всей программы. Подпрограмма должна быть описана до того, как будет использована в программе или другой подпрограмме.

Процедуры и функции объявляются в разделе описания вслед за разделом переменных.

Тогда общая структура программы выглядит так:

```
Program hh;
Label; {описание меток}
Const; {описание констант}
Type; {описание типов}
Var; {описание переменных}
Procedure; {описание процедур}
Function; {описание функций}
Begin
. . .
. . .
end.
```

Выполнение программы начинается с операторов основной программы. При необходимости вызывается подпрограмма и начинают действовать её операторы. Затем управление передаётся в основную программу, которая продолжает выполняться.

Обращение к подпрограмме - переход к выполнению подпрограммы с заданием информации, необходимой для ее выполнения и возврата. Подпрограмма вызывается по своему имени с указанием необходимых параметров.

Файлы. Виды файлов. Работа с файлами.

До сих пор мы рассматривали задачи, в которых во время выполнения программы данные поступают с клавиатуры, а результаты выводятся на экран дисплея. Поэтому ни исходные данные, ни результаты не сохраняются. Всякий раз при выполнении одной и той же программы, особенно во время отладки, приходится заново вводить данные. А если их очень много? В языке Паскаль есть возможность записать их на диск. для этого необходимо оформить исходные данные и результаты в виде файлов, которые хранятся на диске точно так же как и программы.

Понятие файла – это фундаментальное понятие информатики, вспомним же его определение.

Определение. Файлом называется область памяти жесткого диска, имеющая свое имя.

Язык Турбо Паскаль предлагает три вида такого представления:

- типизированные файлы,
- текстовые файлы,
- нетипизированные файлы.

Определение. Типизированный файл – последовательность элементов одного типа.

Описание файлового типа имеет синтаксис:

file of < тип элементов>;

По сути любой физический файл, Вы можете представить как последовательность блоков памяти описанного типа. Все

компоненты файла имеют общее имя, а каждый еще и имеет свой номер. Начальный элемент имеет нулевой номер.

С каждым файлом можно связать понятие текущий указатель. Это неявно описанная переменная, которая указывает на конкретный элемент файла. Действия с файлами производятся поэлементно, причем в них участвует тот элемент, на который "смотрит" текущий указатель, перемещающийся в результате выполнения действия на следующий элемент.

А самое основное, что Вы должны уметь делать над файлом это – записать информацию из программы в файл и считать нужную информацию в выделенную переменную для обработки программой

До начала работы с файлами устанавливается связь файловой переменной MyFile с именем дискового файла.

После того, как файловая переменная с помощью процедуры Assign связана с конкретным дисковым файлом, можно выполнить любую допустимую операцию с ним.

Все файлы, открытые в результате работы программы, должны быть закрыты при завершении программы процедурой

```
close (MyFile);
```

При выполнении этого оператора закрывается физический файл на диске и фиксируются изменения, связанные с использованием данного файла.

Открытие нового файла для записи производится процедурой, единственный аргумент которой – переменная файлового типа, например:

```
rewrite (MyFile);
```

После выполнения процедуры rewrite файл доступен как для записи, так и для чтения.

Подготовку существующего файла для чтения и записи выполняет процедура

```
reset (MyFile);
```

Запись в файл производится процедурой

```
write (MyFile, var1, var2, ...., varN);
```

Первый аргумент этой процедуры – переменная файлового

типа, далее следует список записываемых переменных, которые должны соответствовать обьявленному типу файла. При выполнении этой операции текущий указатель файла смещается на число позиций, равное числу переменных.

Чтение из файла производится аналогичной процедурой:

```
read (MyFile, var1, var2, ...., varN);
```

Положение элементов в файле нумеруется, начиная с номера 0 для первого элемента. После последнего элемента файла автоматически записывается признак конца файла.

Шаблон программы для записи данных в файл:

```
Program Writing;
Var
   FileName : string; {строка, содержащая имя файла}
   FVar : file of byte; {переменная файлового типа}
   Index : byte;
Begin
   write ('Введите имя файла ');{предложение ввести имя файла}
   readln (FileName);{ввод имени файла}
   assign (FVar, FileName);{связь имени файла и переменной}
   rewrite (FVar);{открытие файла для записи}
   for Index := 0 to 99 do {цикл для расчетов и вывода данных в файл}
      write (FVar,Index);{запись в файл FVar величины Index}
   close (FVar); {закрытие файла}
End.
```

ОБРАЗЦЫ ЛИСТИНГОВ РЕШЕНИЯ ЗАДАЧИ

– Линейный алгоритм

```
program zadacha (input, output);
{вычисление расстояния между двумя точками}
var
```

```
x1, y1, x2, y2, c: real;
bwgin
writeln (’введите значения x1, y1, x2, y2’);
readln (x1, y1, x2, y2);
c:=sqrt ((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
writeln (’расстояние между точками=’,c)
end.
```

– Циклический алгоритм вычисления суммы натуральных чисел

Первоначальное значение S:=0
Подсчёт в цикле:
S:=S+{текущее значение}, необходимо вычислить S=1+2+3+…N

```
Program zadacha (input, output);
{вычисление суммы натуральных чисел)
Var
N,s,i : Interger;
Begin
Writeln (‘те натуральное число N=’);
Readln (n) ;
S:=0
For I:=1 to n do s:=s+1 ;
Writeln (‘сумма=’,s)
End.
```

– Массивы

Пример 1. Заменить отрицательные элементы на противоположные по знаку.
Для этого опишем процедуру. Ей будем передавать один параметр – массив, который будет результатом ее выполнения, так как некоторые элементы могут быть заменены.

```
Procedure Zamena (Var m : MyArray; n:integer);
Var
i : integer;
Begin
for i := 1 to n do
  if m[i] < 0   then  m[i] := -1*m[i];
End;
```

Пример 2. Дана таблица действительных чисел. Сосчитайте сумму всех чисел в таблице.

```
Procedure Summa(A : MyArray; n, m: integer; Var S: real);
Var
i, j : integer;
Var
i, j , Summa : integer;
Begin
S:= 0;
for i := 1 to n do
for j := 1 to m do
S := S+A[i,j];
End.
```

– ЗАПИСИ

В массиве хранятся данные об учениках класса: школа, фамилия, класс. Вывести список учеников, которые учатся в восьмом классе.

```
Program LipovsevM;
Uses
Crt;
Type
Uchenik=record
Shkola : integer;
```

```
Fam : string[15];
Klass : integer;
end;
Var
I,n,a,j : integer;
Massiv : array[1..100] of Uchenik;
Procedure Poisk;
Begin
for i:=1 to n do
if massiv[i].klass=8
then
with massiv[i] do
writeln(Shkola:4,' ',Fam:15,' ',klass);
End;
Begin
ClrScr;
writeln('Введите число учеников ');
write('->');
read(n);
for i:=1 to n do
begin
writeln('Введите через пробел номер школы и фамилию
ученика ');
write('->');
with massiv[i] do
begin
readln(Shkola,Fam);
write('Введите класс ученика (только число) ->');
read(Klass);
end;
end;
writeln('Ученики 8-ых классов:');
writeln('Школа Фамилия Класс');
writeln('--------------------------------');
Poisk;
```

```
ReadKey;
End.
```

– Подпрограммы

Найти среднее арифметическое двух чисел.

```
Program Fadeev;
Uses
Crt;
Var
A, B : integer;
Rez :real;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
Function SredArif(A, B:integer):real;
Begin
SredArif:=(A+B)/2;
End;
{- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -}
Begin
ClrScr;
write('Введите два числа ');
readln(A,B);
Rez:=SredArif(A,B);
write('Среднее арифметическое этих чисел равно ',Rez:5:3);
readln;
End.
```

ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ БАЗ ДАННЫХ

База данных — это один или несколько файлов данных, предназначенных для хранения, изменения и обработки больших объемов взаимосвязанной информации.

Примерами баз данных могут быть телефонная книга, каталог

товаров, штатное расписание предприятия и т. д. Конкретная база данных содержит информацию об очень малой части окружающего мира. В базу данных помещаются сведения о нескольких однотипных объектах окружающего мира.

Предметной областью принято называть часть реального мира, подлежащую изучению с целью организации управления в этой сфере и последующей автоматизации процесса управления.

Объектом называется элемент информационной системы, сведения о котором хранятся в базе данных. Объект — это нечто существующее и различимое, обладающее набором свойств. Иногда объект также называют сущностью (от англ, entity). Классом объектов называют их совокупность, обладающую одинаковым набором свойств. Отличие одного объекта от другого определяется конкретными значениями свойств. Объекты бывают материальные и идеальные. К материальным объектам относятся предметы материального мира — автомобили, здания, предметы мебели и т. д.

К идеальным (абстрактным) объектам относятся такие, которые существуют только в представлении (памяти) человека, но обладающие реальными свойствами. К идеальным объектам можно отнести спектакль, сюжет книги и т. д.

Сущность — отображение объекта в памяти человека или компьютера.

Сущность всегда меньше объекта по составу свойств (характеристик). Поэтому основной задачей при создании базы данных является определение необходимого и достаточного количества свойств (характеристик) сущности, которые будут храниться в базе данных.

Следует различать два близких понятия — сущность и экземпляр сущности.

Сущность — более широкое понятие и употребляется в разных вариантах. Под сущностью будем понимать коллекцию экземпляров сущности, которую впоследствии будем представлять в виде таблицы.

Экземпляр сущности — набор сведений о единице сущности,

который впоследствии будем представлять в виде записи.

Атрибут - это информационное отображение свойств объекта. Каждый объект характеризуется некоторым набором атрибутов.

Атрибут — конкретное значение свойства сущности. Атрибут и сущность тесно связаны между собой. То, что в одной базе данных выступает как атрибут, в другой базе данных может быть сущностью.

Параметр — конкретное значение свойства объекта.

Как правило, отображение совокупности сущностей в базе данных выполняется в виде таблицы. С целью эффективного использования памяти компьютера база данных содержит несколько таблиц.

Поэтому некоторые сущности будут сложными (разделенными на несколько таблиц), т. е. состоящими из одной или нескольких подсущностей. Это обстоятельство и вызвало необходимость устанавливать связи между таблицами.

Связь — это функциональная зависимость между сущностями.

Первичный ключ - это атрибут (или группа атрибутов), который уникальным образом идентифицируют каждый экземпляр объекта (запись).

Вторичным ключом называется атрибут (или группа атрибутов), значение которого может повторяться для нескольких записей (экземпляров объекта).

Ключевым элементом данных (ключом) называется такой атрибут, по значению которого можно определить значения других неключевых атрибутов.

Запись данных (англ, эквивалент record) - это совокупность значений связанных элементов данных.

Прежде всего, вторичные ключи используются в операциях поиска записей.

Программное обеспечение, осуществляющее операции над базами данных, получило название СУБД - система управления базами данных. Очевидно, что его работа должна быть организована таким образом, чтобы выполнялись перечисленные принци-

пы.

Систе́ма управле́ния ба́зами да́нных (СУБД) — совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Классификация БД

По технологии обработки данных БД подразделяются на локальные и распределенные.

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

Распределенная база данных состоит из нескольких, возможно, пересекающихся или даже дублирующих друг друга частей, которые хранятся в различных ЭВМ вычислительной сети. Работа с такой базой осуществляется с помощью системы управления распределенной базой данных (СУРБД). Основная задача систем управления распределенными базами данных состоит в обеспечении средства интеграции локальных баз данных.

Возможны однородные и неоднородные распределенные базы данных.

В однородном случае каждая локальная база данных управляется одной и той же СУБД. В неоднородной системе локальные базы данных могут относиться даже к разным моделям данных. Сетевая интеграция неоднородных баз данных - очень сложная проблема. Многие решения известны на теоретическом уровне, но пока не удается справиться с главной проблемой: недостаточной эффективностью интегрированных систем. Более успешно решается промежуточная задача - интеграция неоднородных SQL-ориентированных систем. Этому в большой степени способствует стандартизация языка SQL.

По типу данных БД

Документальные БД. Служат для хранения слабо структури-

руемых данных, представляющих собой оригиналы, копии документов или их адреса. Документальные модели данных соответствуют представлению о слабоструктурированной информации, ориентированной в основном на свободные форматы документов, текстов на естественном языке. В документальных БД единицей хранения является какой либо документ и пользователю в ответ на запрос выдается либо сам документ либо ссылка на него. БД документального типа могут быть организованы по разному: без хранения и с хранением самого исходного документа на машинных носителях. К системам первого типа можно отнести реферативные и библиографические, а также БД – указатели.

Фактографические БД. Используются для хранения жестко структурируемых данных, представляющих собой факты, сведения, содержащиеся в явном виде в документах, для представления данных и манипулирования ими используются соответственно языке описания данных и язык манипулирования данными. В фактографических БД содержатся краткие сведения об описываемых объектах, представленные в строго определенном формате.

Современные информационные технологии постепенно стирают границу между фактографическими и документальными БД. Существуют средства, позволяющие легко подключать любой документ (текстовый, графический, звуковой) к фактографической базе данных. Фактографические модели данных соответствуют представлению информации в виде определенных структур данных (дерево, сеть, таблица). В системах фактографического типа в БД хранится информация об интересующих пользователя объектах предметной области в виде фактов; в ответ на запрос пользователя выдается требуемая информация об интересующем его объекте или сообщение о том, что информация отсутствует.

По модели данных:

• *Иерархические*; (Иерархические базы данных могут быть представлены как дерево, состоящее из объектов различных уровней. Верхний уровень занимает один объект, второй — объекты второго уровня и т. д.)

• *Сетевые;* (К основным понятиям сетевой модели базы данных относятся: уровень, элемент (узел), связь.)

• *Реляционные;* (Реляционная база данных — база данных, основанная на реляционной модели данных. Слово «реляционный» происходит от англ. relation (отношение).

• *Объектно-ориентированные*; (Объектно-ориентированная база данных (ООБД) — база данных, в которой данные оформлены в виде моделей объектов, включающих прикладные программы, которые управляются внешними событиями.)

• *Объектно-реляционные*; (Объектно-реляционные СУБД объединяют в себе черты реляционной и объектной моделей. Их возникновение объясняется тем, что реляционные базы данных хорошо работают со встроенными типами данных и гораздо хуже — с пользовательскими, нестандартными.

Классификация **БД по содержимому:**

• *географические;* • *исторические;* • *научные;* • *мультимедийные.*

КЛАССИФИКАЦИЯ СУБД

СУБД — это система программного обеспечения, обеспечивающая ввод, хранение и доступ к данным многих пользователей, а также хранящая описание структуры данных. СУБД с помощью встроенных в нее механизмов обеспечивает правильность, полноту и непротиворечивость данных, а также простой и понятный интерфейс.

Главным преимуществом использования СУБД является то, что конечный пользователь может иметь любой уровень квалификации — от самого низкого до самого высокого. СУБД берет на себя многочисленные функции: хранение информации, обеспечение целостности и достоверности данных, одновременная работа многочисленных пользователей, поддерживая секретность и доступность данных на основе паролей и прав доступа, физическое размещение данных в памяти персонального компьютера и правила их описания, простые и унифицированные механизмы

манипулирования данными и т. д.

По мощности СУБД делятся на настольные и корпоративные.

Для настольных характерны невысокие требования к техническим средствам, ориентация на конечного пользователя, низкая стоимость. Корпоративные – обеспечивают работу в распределенной среде, высокую производительность, поддержку коллективной работы при проектировании систем, имеют развитые средства администрирования и более широкие возможности поддержания целостности. Оба типа систем интенсивно развиваются.

По способу доступа к БД

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

· База данных в виде набора файлов находится на жестком диске компьютера.

· На том же компьютере установлены СУБД и приложение для работы с БД.

Пользователь запускает приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку/обновление информации.

· Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД.

· СУБД инициирует обращения к данным, обеспечивая выполнение запросов пользователя (осуществляя необходимые операции над данными).

· Результат СУБД возвращает в приложение.

· Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

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

Многопользовательская технология работы обеспечивалась либо режимом мультипрограммирования (одновременно могли

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

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

Работа построена следующим образом:

· База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (файлового сервера).

· Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлены СУБД и приложение для работы с БД.

· На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к БД на выборку/обновление информации.

· Все обращения к БД идут через СУБД, которая инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на файловом сервере.

· СУБД инициирует обращения к данным, находящимся на файловом сервере, в результате которых часть файлов БД копи-

руется на клиентский компьютер и обрабатывается, что обеспечивает выполнение запросов пользователя (осуществляются необходимые операции над данными).

· При необходимости (в случае изменения данных) данные отправляются назад на файловый сервер с целью обновления БД.

· Результат СУБД возвращает в приложение.

· Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

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

Основные недостатки данной архитектуры:

· При одновременном обращении множества пользователей к одним и тем же данным производительность работы резко падает, т.к. необходимо дождаться пока пользователь, работающий с данными, завершит свою работу. В противном случае возможно затирание исправлений, сделанных одними пользователями, изменениями других пользователей.

· Вся тяжесть вычислительной нагрузки при доступе к БД ложится на приложение клиента, так как при выдаче запроса на выборку информации из таблицы вся таблица БД копируется на клиентскую машину и выборка осуществляется на клиенте. Таким образом, неоптимально расходуются ресурсы клиентского компьютера и сети. В результате возрастает сетевой трафик и увеличиваются требования к аппаратным мощностям пользовательского компьютера.

· Недостаточно развитый аппарат транзакций служит потенциальным источником ошибок в плане нарушения смысловой и ссылочной целостности информации при одновременном внесении изменений в одну и ту же запись. Преимуществом этой архитектуры является низкая нагрузка на ЦП сервера, а недостатком - высокая загрузка локальной сети. На данный момент файл-серверные СУБД считаются устаревшими.

Клиент-серверные. Такие СУБД состоят из клиентской части (которая входит в состав прикладной программы) и сервера. Кли-

ент-серверные СУБД, в отличие от файл-серверных, обеспечивают разграничение доступа между пользователями и мало загружают сеть и клиентские машины. Недостаток клиент-серверных СУБД в самом факте существования сервера (что плохо для локальных программ — в них удобнее встраиваемые СУБД) и больших вычислительных ресурсах, потребляемых сервером. В этой архитектуре на выделенном сервере, работающем под управлением серверной операционной системы, устанавливается специальная программа, управляющая удаленной базой данных - SQL-сервер, например, Microsoft®SQL Server™или Oracle. Основа работы сервера БД - использование языка запросов SQL (Structured Query Language). SQL-сервер обеспечивает интерпретацию запроса, его выполнение в базе данных, формирование результата выполнения запроса и выдачу его приложению-клиенту. При этом ресурсы клиентского компьютера не участвуют в физическом выполнении запроса; клиентский компьютер лишь отсылает запрос к серверной БД и получает результат, после чего интерпретирует его необходимым образом и представляет пользователю. Извлеченные данные транспортируются по сети от сервера к клиенту. Тем самым, количество передаваемой по сети информации уменьшается во много раз. Примеры: Firebird, Interbase, MS SQL Server, Sybase, Oracle, PostgreSQL, MySQL.

Итак, в результате работа построена следующим образом:

· База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети).

· СУБД располагается также на сервере сети.

· Существует локальная сеть, состоящая из клиентских компьютеров, на каждом из которых установлено клиентское приложение для работы с БД.

· На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к СУБД, расположенной на сервере, на выборку/обновление информации. Для общения используется специальный язык запросов SQL, т.е. по сети от клиента к серверу пе-

редается лишь текст запроса.

· СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере.

· СУБД инициирует обращения к данным, находящимся на сервере, в результате которых на сервере осуществляется вся обработка данных и лишь результат выполнения запроса копируется на клиентский компьютер. Таким образом СУБД возвращает результат в приложение.

· Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

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

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

Трехуровневая (трехзвенная) ("тонкий клиент" - сервер приложений - сервер базы данных). Трехзвенная (в некоторых случаях многозвенная) архитектура представляет собой дальнейшее совершенствование технологии "клиент – сервер". Трехуровневая архитектура функционирует в Интранет- и Интернет-сетях. Клиентская часть ("тонкий клиент"), взаимодействующая с пользователем, представляет собой HTML-страницу в Web-браузере либо Windows-приложение, взаимодействующее с Web-сервисами. Вся программная логика вынесена на сервер приложений, который обеспечивает формирование запросов к базе данных, передавае-

мых на выполнение серверу баз данных. Сервер приложений может быть Web-сервером или специализированной программой (например, Oracle Forms Server)

Рассмотрев архитектуру "клиент – сервер", можно заключить, что она является 2-звенной: первое звено – клиентское приложение, второе звено – сервер БД + сама БД. В трехзвенной архитектуре вся бизнес-логика (деловая логика), ранее входившая в клиентские приложения, выделяется в отдельное звено, называемое сервером приложений. При этом клиентским приложениям остается лишь пользовательский интерфейс. Так, в качестве клиентского приложения в описанном выше примере выступает Web-браузер.

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

Итак, в результате работа построена следующим образом:

- База данных в виде набора файлов находится на жестком диске специально выделенного компьютера (сервера сети).
- СУБД располагается также на сервере сети.
- Существует специально выделенный сервер приложений, на котором располагается программное обеспечение (ПО) делового анализа (бизнес-логика).
- Существует множество клиентских компьютеров, на каждом из которых установлен так называемый "тонкий клиент" – клиентское приложение, реализующее интерфейс пользователя.
- На каждом из клиентских компьютеров пользователи имеют возможность запустить приложение – тонкий клиент. Используя предоставляемый приложением пользовательский интерфейс, он инициирует обращение к ПО делового анализа, расположенному на сервере приложений.
- Сервер приложений анализирует требования пользователя и формирует запросы к БД. Для общения используется

специальный язык запросов SQL, т.е. по сети от сервера приложений к серверу БД передается лишь текст запроса.

- СУБД инкапсулирует внутри себя все сведения о физической структуре БД, расположенной на сервере.
- СУБД инициирует обращения к данным, находящимся на сервере, в результате которых результат выполнения запроса копируется на сервер приложений.
- Сервер приложений возвращает результат в клиентское приложение (пользователю).
- Приложение, используя пользовательский интерфейс, отображает результат выполнения запросов.

Модель данных

Модель данных — это абстрактное, самодостаточное, логическое определение объектов, операторов и прочих элементов, в совокупности составляющих абстрактную машину доступа к данным, с которой взаимодействует пользователь. Эти объекты позволяют моделировать структуру данных, а операторы — поведение данных.

Каждая БД и СУБД строится на основе некоторой явной или неявной модели данных. Все СУБД, построенные на одной и той же модели данных, относят к одному типу. Например, основой реляционных СУБД является реляционная модель данных, сетевых СУБД — сетевая модель данных, иерархических СУБД — иерархическая модель данных и т.д.

СУБД используют различные модели данных:

Иерархическая модель. В иерархической модели элементы организованы в структуры, связанные между собой иерархическими или древовидными связями. Родительский элемент может иметь несколько дочерних элементов. Но у дочернего элемента может быть только один предок.

Иерархическая модель организует данные в форме дерева с иерархией родительских и дочерних сегментов. Такая модель подразумевает возможность существования одинаковых (преимущественно дочерних) элементов. Данные здесь хранятся в се-

рии записей с прикреплёнными к ним полями значений. Модель собирает вместе все экземпляры определённой записи в виде «типов записей» — они эквивалентны таблицам в реляционной модели, а отдельные записи — столбцам таблицы. Для создания связей между типами записей иерархическая модель использует отношения типа «родитель-потомок» вида 1:N. Это достигается путём использования древовидной структуры — она «позаимствована» из математики, как и теория множеств, используемая в реляционной модели.

Иерархические БД были популярны, начиная с конца 1960-х годов, когда компания IBM представила свою СУБД «Система управления информацией. Иерархическая схема состоит из типов записей и типов «родитель-потомок»:

- Запись — это набор значений полей.
- Записи одного типа группируются в типы записей.
- Отношения «родитель-потомок» — это отношения вида 1:N между двумя типами записей.
- Схема иерархической базы данных состоит из нескольких иерархических схем.

Сетевая модель. В сетевой модели данных у родительского элемента может быть несколько потомков, а у дочернего элемента — несколько предков. Записи в такой модели связаны списками с указателями. Иерархическая модель структурирует данные в виде древа записей, где есть один родительский элемент и несколько дочерних. Сетевая модель позволяет иметь несколько предков и потомков, формирующих решётчатую структуру.

Сетевая модель позволяет более естественно моделировать отношения между элементами. И хотя эта модель широко применялась на практике, она так и не стала доминантной по двум основным причинам. Во-первых, компания IBM решила не отказываться от иерархической модели в расширениях для своих продуктов, таких как IMS и DL/I. Во-вторых, через некоторое время её сменила реляционная модель, предлагавшая более высокоуровневый, декларативный интерфейс.

Популярность сетевой модели совпала с популярностью

иерархической модели. Некоторые данные намного естественнее моделировать с несколькими предками для одного дочернего элемента. Сетевая модель как раз и позволяла моделировать отношения «многие ко многим».

Основной элемент сетевой модели данных — набор, который состоит из типа «запись-владелец», имени набора и типа «запись-член». Запись подчинённого уровня («запись-член») может выполнять свою роль в нескольких наборах. Соответственно, поддерживается концепция нескольких родительских элементов.

Запись старшего уровня («запись-владелец») также может быть «членом» или «владельцем» в других наборах. Модель данных — это простая сеть, связи, типы пересечения записей. А также наборы, которые могут их объединять. Таким образом, полная сеть представлена несколькими парными наборами.

В каждом из них один тип записи является «владельцем» (от него отходит «стрелка» связи), и один или более типов записи являются «членами» (на них указывает «стрелка»). Обычно в наборе существует отношение 1:M, но разрешено и отношение 1:1. Сетевая модель данных CODASYL основана на математической теории множеств.

Известные сетевые базы данных: TurboIMAGE; IDMS; Встроенная RDM; Серверная RDM.

Реляционная модель. В реляционной модели, в отличие от иерархической или сетевой, не существует физических отношений. Вся информация хранится в виде таблиц (отношений), состоящих из рядов и столбцов. А данные двух таблиц связаны общими столбцами, а не физическими ссылками или указателями. Для манипуляций с рядами данных существуют специальные операторы. Распространённые реляционные СУБД: Oracle, Sybase, DB2, Ingres, Informix и MS-SQL Server.

Реляционные таблицы обладают следующими свойствами:

- Все значения атомарны.
- Каждый ряд уникален.
- Порядок столбцов не важен.
- Порядок рядов не важен.

- У каждого столбца есть своё уникальное имя.

Некоторые поля могут быть определены как ключевые. Это значит, что для ускорения поиска конкретных значений будет использоваться индексация. Когда поля двух различных таблиц получают данные из одного набора, можно использовать оператор JOIN для выбора связанных записей двух таблиц, сопоставив значения полей.

Часто у полей будет одно и то же имя в обеих таблицах. Например, таблица «Заказы» может содержать пары «ID-покупателя» и «код-товара». А в таблице «Товар» могут быть пары «код-товара» и «цена». Поэтому чтобы рассчитать чек для определённого покупателя, необходимо суммировать цену всех купленных им товаров, использовав JOIN в полях «код-товара» этих двух таблиц. Такие действия можно расширить до объединения нескольких полей в нескольких таблицах.

Объект в реляционной модели определяется как позиция информации, хранимой в базе данных. Объект может быть осязаемым или неосязаемым. Примером осязаемого объекта может быть сотрудник организации, а примером неосязаемой сущности — учётная запись покупателя. Объекты определяются атрибутами — информационным отображением свойств объекта. Эти атрибуты также известны как столбцы, а группа столбцов — как ряд. Ряд также можно определить как экземпляр объекта.

Объекты связываются отношениями, основные типы которых можно определить следующим образом:

1. «Один к одному» - означает, что одной записи в главной таблице соответствует одна запись в подчиненной таблице.
2. «Один ко многим» - означает, что одной записи в главной таблице может соответствовать несколько записей в подчиненной таблице.
3. «Много к одному» - отличается от «один ко многим» направлением.
4. «Много ко многим» - означает, что одной записи главной таблицы может соответствовать несколько записей подчиненной таблицы, и одновременно одной записи подчиненной таб-

лицы – несколько записей главной таблицы.

В реляционной модели объекты и их отношения представлены двухмерным массивом или таблицей.

Каждая таблица представляет объект.

Каждая таблица состоит из рядов и столбцов.

Отношения между объектами представлены столбцами.

Каждый столбец представляет атрибут объекта.

Значения столбцов выбираются из области или набора всех возможных значений.

Столбцы, которые используются для связи объектов, называются ключевыми. Есть два типа ключей — первичные и внешние.

Первичные служат для однозначного определения объекта. Внешний ключ — это первичный ключ одного объекта, существующий как атрибут в другой таблице.

Преимущества реляционной модели данных:

1. Простота использования.
2. Гибкость.
3. Независимость данных.
4. Безопасность.
5. Простота практического применения.
6. Слияние данных.
7. Целостность данных.

Недостатки:

1. Избыточность данных.
2. Низкая производительность.

ПРОЕКТИРОВАНИЕ РЕЛЯЦИОННЫХ БАЗ ДАННЫХ

Этапы проектирования РБД.

Проектирование баз данных — процесс создания схемы базы данных и определения необходимых ограничений целостности

Основные задачи

- Обеспечение хранения в БД всей необходимой информации.
- Обеспечение возможности получения данных по всем необ-

ходимым запросам.

- Сокращение избыточности и дублирования данных.
- Обеспечение правильности их содержания и т.д.

На этапе проектирования базы данных разработчик должен определить, из каких таблиц должна состоять база данных, какие данные нужно поместить в каждую таблицу и как связать таблицы.

Следовательно, в результате проектирования определяются логическая структура базы данных, т. е. состав реляционных таблиц, их структура и межтабличные связи. Для создания базы данных необходимо располагать описанием выбранной предметной области, охватывающим реальные объекты и процессы, а также определить все необходимые источники информации для удовлетворения предполагаемых запросов. Структура данных предметной области может отображаться ИЛМ, на основе которой легко создается реляционная база данных.

Этапы проектирования и создания базы данных:

• построение информационно-логической модели данных предметной области;

• определение логической структуры реляционной базы данных;

• конструирование таблиц базы данных;

• создание схемы данных;

• ввод данных в таблицы (создание записей);

• разработка необходимых форм, запросов, макросов, модулей, отчетов;

• разработка пользовательского интерфейса.

Выделение информационных объектов

В процессе разработки модели данных необходимо выделить информационные объекты, соответствующие требованиям нормализации данных, и определить связи между ними. При разработке модели данных используются два подхода.

1.Функциональный подход. Сначала определяются основные задачи, для решения которых строится база, выявляются потреб-

ности задач в данных и соответственно определяются состав и структура информационных объектов.

2.Предметный подход: ИО выбираются на основе общих представлений о ПО. Сразу устанавливаются типовые объекты предметной области. При этом подходе легко выявить информационные объекты, соответствующие реальным объектам, однако получаемая при этом ИЛМ, как правило, требует дальнейших преобразований, в частности преобразования связей N:M между объектами.

Наиболее рационально сочетание этих подходов, так как на начальном этапе, как правило, нет исчерпывающих сведений обо всех задачах.

Рассмотрим формальные правила выделения информационных объектов:

- на основе описания предметной области выявить документы и их атрибуты, подлежащие хранению в базе данных;
- определить функциональные зависимости между атрибутами;
- выбрать все зависимые атрибуты и указать для каждого все его ключевые атрибуты, т. е. атрибуты, от которых он зависит;
- сгруппировать атрибуты, одинаково зависимые от ключевых атрибутов.

(Полученные группы зависимых атрибутов вместе с их ключевыми атрибутами образуют информационные объекты.)

Определение связей между объектами, создание ИЛМ.

Связь устанавливается между двумя информационными объектами. Наличие связи, как правило, определяется природой реальных объектов, процессов или явлений, отображаемых этими информационными объектами.

Многозначные связи в этом случае осуществляются через объект, выполняющий роль объекта-связки. Примером много - многозначных связей является пара вида: товар-договор, товар накладная, если один поставщик поставляет разные наименования товаров, а товар одного наименования может поставляться

несколькими поставщиками.

ИЛМ рассматриваемой предметной области построена в соответствии с выявленными информационными объектами и связями между ними. Объекты в ИЛМ размещены по уровням. На нулевом уровне размещаются объекты, не подчиненные никаким другим объектам. Уровень остальных объектов определяется наиболее длинным путем к объекту от нулевого уровня. Такое размещение объектов дает представление об их иерархической подчиненности, делает модель более наглядной и облегчает понимание одно-многозначных отношений между объектами.

В среде СУБД может быть создана схема данных, наглядно отображающая логическую структуру базы данных. Схема данных практически совпадает с топологией информационно-логической модели.

ЯЗЫК ЗАПРОСОВ SQL

SQL (Structured Query Language) — представляет из себя структурированный язык запросов (перевод с английского). Язык ориентирован на работу с реляционными (табличными) базами данных. Язык прост и, по сути, состоит из команд (интерпретируемый), посредством которых можно работать с большими массивами данных (базами данных), удаляя, добавляя, изменяя информацию в них и осуществляя удобный поиск.

Для работы с SQL кодом необходима система управления базами данных (СУБД), которая предоставляет функционал для работы с базами данных.

Система управления базами данных (СУБД) — совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.

Язык SQL состоит из следующих составных частей:

1. Язык манипулирования данными (Data Manipulation Language, DML);**состоит из 4 главных команд:**

- выборка данных из БД — SELECT
- вставка данных в таблицу БД — INSERT

- обновление (изменение) данных в таблицах БД — UPDATE
- удаление данных из БД — DELETE

2. Язык определения данных (Data Definition Language, DDL);используется для создания и изменения структуры базы данных и ее составных частей — таблиц, индексов, представлений (виртуальных таблиц), а также триггеров и сохраненных процедур. Ими являются:

- создание базы данных — CREATE DATABASE
- создание таблицы — CREATE TABLE
- изменение таблицы (структуры) — ALTER TABLE
- удаление таблицы — DROP TABLE

3. Язык управления данными (Data Control Language, DCL).

GRAND – назначение привилегий

REVOKE – удаление привилегий.

Язык SQL предоставляет ряд функций, используемых в выражениях, из которых наиболее часто применяются следующие:

Статистические функции:

AVG() – среднее значение

MAX() – максимальное значение

MIN() – минимальное значение

SUM() – сумма значений

COUNT() – количество значение

COUNT(*) – количество ненулевых значений

Строковые функции:

UPPER(Str) – преобразование символов строки Str в символы верхнего регистра

LOWER(Str) – преобразование символов строки Str в символы нижнего регистра

TRIM(Str) – удаление пробелов в начале и конце строки Str

Характеристика оператора SELECT.

Оператор SELECT – важнейший оператор языка SQL. Он используется для отбора записей, удовлетворяющих сложным кри-

териям поиска. Этот оператор имеет формат:

SELECT [DISTINCT] Список полей или *
FROM список таблиц
[WHERE условие отбора]
[ORDER BY список полей для сортировки]
[GROUP BY список полей для группировки]
[HAVING условие группирования]
[UNION вложенный оператор SELECT]

При выполнении оператора SELECT результат SQL-запроса – это выборка записей, удовлетворяющих заданному критерию.

Описатель DISTINCT определяет, разрешены или запрещены повторяющиеся записи (имеющие одинаковые значения всех полей). Если описатель отсутствует, то в НД могут входить записи, имеющие одинаковые значения всех полей.

Список полей определяет состав полей результирующего НД. Если в НД требуется включить все поля таблицы, то вместо перечисления имен полей можно указать символ «*».

В операнде **FROM** перечисляются имена таблиц, из которых отбираются записи. Операнд **WHERE** задает критерий, которым должны удовлетворять записи в результирующем НД. Выражение, описывающее условие отбора, является логическим. Операнд **GROUP BY** позволяет выделить группы записей в результирующем НД. Группой являются записи с одинаковыми значениями в полях, перечисленных после операнда GROUP BY. Операнд **HAVING** используется совместно с операндом GROUP BY для набора записей внутри групп. Операнд **ORDER BY** содержит список полей, определяющих порядок сортировки записей результирующего НД. По умолчанию сортировка по каждому полю выполняется по возрастанию. Если необходимо отсортировать по убыванию, то после имени этого поля указывается описатель **DESC**. Операторы SELECT могут быть вложенными друг в друга. Для объединения операторов используется операнд UNION, в котором располагается вложенный оператор SELECT, называемый подзапросом. Результирующий НД представляет записи, удовлетворяющие условиям отбора, заданным операндами

WHERE обоих операторов.

Пример:

```
SELECT Personnel.name, Personnel.I_Code
        FROM Personnel
```

Критерии отбора записей. Для ограничения отбираемой из базы данных информации оператор SELECT позволяет использовать условие, которое задается с помощью предложения WHERE. В случае реализации условной выборки оператор SELECT имеет следующий вид:

```
SELECT {* | ALL | DISTINCT fieldl . field2 fieldN}
FROM tablel {.. table2 tableN}
WHERE условие
```

Специальные операторы языка SQL, применяемые для задания условия, можно разделить на следующие группы:

- операторы сравнения;
- логические операторы;
- операторы объединения;
- операторы отрицания.

Результатом выполнения каждого из этих операторов является логическое значение (true или false). Если для некоторой записи оператор возвращает значение true, то запись включается в результат выборки, если false — не включается.

Операторы сравнения используются в запросах SQL для наложения ограничений на информацию, возвращаемую в результате выполнения запроса. Это типичные операторы, существующие во всех алгоритмических языках: <, >, <=, >=, <>, != (не равно), !> (не больше), !< (не меньше).

В примера представлен запрос, выбирающий из таблицы «Товары» только те записи, категория товаров в которых равна 2:

```
SELECT *
FROM Товары
WHERE Категория=2
```

Логические операторы - это операторы, в которых для за-

дания ограничений на отбор данных используются специальные ключевые слова. В SQL определены следующие логические операторы:

IS NULL, BETWEEN…AND, IN, LIKE, EXISTS, UNIQUE, ALL, ANY.

Оператор IS NULL предназначен для сравнения текущего значения поля со значением NULL. Он используется для отбора записей, в некоторое поле которых не занесено никакое значение.

Для иллюстрации использования этого оператора воспользуемся таблицей «Клиенты». С помощью приведенного ниже запроса произведем выборку из нее записей клиентов, у которых не указано название предприятия, которое они представляют:

```
SELECT Фамилия, Имя, Отчество, Телефон, Город, Адрес
FROM Клиенты
WHERE Предприятие IS NULL
```

Оператор BETWEEN...AND применяется для отбора записей, в которых значения поля находятся внутри заданного диапазона. Границы диапазона включаются в условие отбора.

Чтобы продемонстрировать работу этого оператора, вернемся к таблице «Товары» и выберем в ней товары, цена которых находится в диапазоне от 200 до 2000. Для этого сформируем следующий запрос:

```
SELECT *
FROM Товары
WHERE Цена BETWEEN 200 AND 2000
```

Оператор IN используется для выборки записей, в которых значение некоторого поля соответствует хотя бы одному из значений заданного списка. Выберем из таблицы «Клиенты» список клиентов, которые живут в Беларуси, Украине или Казахстане:

```
SELECT Фамилия, Имя, Отчество, Страна
FROM Клиенты
WHERE Страна IN ('Беларусь','Украина','Казахстан')
```

Оператор LIKE применяется для сравнения значения поля со значением, заданным при помощи шаблонов. Для задания шаблонов используются два символа:

- знак процента *«%»* — заменяет последовательность символов любой (в том числе и нулевой) длины;
- символ подчеркивания «_» — заменяет любой единичный символ.

Найдем в таблице «Клиенты» записи, в которых фамилия клиента начинается с буквы «М»:

SELECT Фамилия, Имя, Отчество, Телефон
FROM Клиенты
WHERE Фамилия LIKE 'М%'

А теперь найдем в этой же таблице записи, для которых номер телефона начинается на цифры (816)025-61, а две последние цифры неизвестны:

SELECT Фамилия, Имя, Отчество, Телефон
FROM Клиенты
WHERE Телефон LIKE ' (816)025-61__ '

Операторы объединения. Часто при написании запроса на выборку данных требуется задать сложное условие, для которого недостаточно использовать только один оператор. В этом случае используется объединение нескольких условий с помощью специальных операторов.

В SQL определены два таких оператора:

- Оператор AND используется в тех случаях, когда необходимо отобрать записи, соответствующие нескольким условиям. Причем для каждой записи, включаемой в результат выборки, должны выполняться *все* заданные ограничения. Оператор AND объединяет несколько условий путем выполнения операции логического умножения результатов всех заданных ограничений. Результат true, соответственно, будет получен только в том случае, если все объединяемые условия принимают значение true.
- Оператор OR выполняет операцию логического сложения результатов всех заданных условий. При использовании данного

оператора запись включается в результирующую выборку в случае выполнения *хотя бы одного* из заданных ограничений.

При использовании операторов объединения каждое логическое выражение следует заключать в круглые скобки. Для примера произведем выборку данных о товарах, цена которых больше 50, но меньше 1000:

```
SELECT *
FROM Товары
WHERE (Цена>50) AND (Цена<1000)
```

Синтаксические правила использования оператора 0R такие же, как и для оператора AND.

Следующий запрос:

```
SELECT *
FROM Товары
WHERE (Цена>50) OR (Цена<1000)
```

возвратит список товаров, цена которых меньше 50 или больше 1000.

Операторы отрицания. Для каждого из рассматриваемых операторов может быть выполнена операция отрицания, меняющая результат выполнения оператора на противоположный. Для реализации этой используется оператор NOT. Ниже приведены примеры использования этого оператора с логическими операторами:

```
IS NOT NULL
NOT BETWEEN
NOT IN
NOT LIKE
NOT EXISTS
NOT UNIQUE
```

Упорядочение данных (сортировка). Для упорядочения данных в выборке, полученной в результате выполнения запроса, используется предложение **ORDER BY**. Синтаксис оператора

SELECT в этом случае будет следующим:

SELECT {* | ALL | DISTINCT fieldl. .field2 fieldN}
FROM tablel {. table2 tableN}
WHERE условие
ORDER BY field {ASC | DESC}

После ключевых слов ORDER BY указывается имя поля (полей), по которому производится сортировка, а затем указывается режим сортировки:

• **ASC** — режим, используемый по умолчанию. При этом информация располагается в порядке возрастания значения указанного поля (для текстовых полей — в алфавитном порядке).

• **DESC** — используется для вывода информации в порядке убывания значений указанного поля (для текстовых полей — в порядке, обратном алфавитному).

Например, чтобы отсортировать список товаров по алфавиту, следует использовать следующий запрос:

SELECT Категория, Наименование, Цена
FROM Товары
ORDER BY Наименование

Использование вычисляемых полей. Язык SQL позволяет создавать вычисляемые поля в тексте запроса. Для реализации этой функции в запросе просто приводится выражение, в котором используются арифметические и математические операторы, а также имена полей в качестве переменных. В результате выполнения запроса с вычисляемыми полями выборка будет содержать не только ту информацию, которая содержится в таблицах базы данных, но и дополнительную информацию, полученную в результате вычисления заданного выражения.

При создании вычисляемого поля можно использовать следующие арифметические операторы:

- оператор сложения «+»;
- оператор вычитания «-»;
- оператор умножения «*»;
- оператор деления «/».

Приоритет перечисленных операторов соответствует общепринятому: умножение и деление, затем сложение и вычитание. Порядком выполнения операторов можно управлять с помощью круглых скобок.

Рассмотрим пример использования вычисляемых полей. Для этого на основании данных таблицы «Продажи» вычислим для каждого товара сумму денег, полученных за проданный товар (произведение цены на количество проданного товара), и сумму, на которую заказано товаров (произведение цены на количество заказанного товара), а также разность между ними:

```
SELECT [Код товара], Цена, Заказано, Продано,
Цена*Продано, Цена*Заказано,
Цена*Заказано-Цена*Продано
FROM Продажи
```

Псевдонимы полей. В запросах SQL можно изменять имена полей. Задаваемые при этом новые имена называются ***псевдонимами*** **(aliases**). Их удобно применять при задании в запросе вычисляемых полей. С помощью псевдонимов этим полям можно присваивать осмысленные имена. Псевдоним помещается после имени поля или после вычисляемого выражения через ключевое слово **AS.**

В качестве примера воспользуемся предыдущим запросом, задав в нем псевдонимы для вычисляемых полей:

```
SELECT [Код товара], Цена, Заказано, Продано,
Цена*Продано AS [Сумма продажи],
Цена*Заказано AS [Сумма заказа],
Цена*Заказано-Цена*Продано AS [Разность]
FROM Продажи
```

Группировка данных. ***Группировка данных*** — это объединение записей в соответствии со значениями некоторого заданного поля. Для группировки результатов выборки совместно с оператором SELECT используется предложение **GROUP BY**. Данное предложение должно следовать после предложения WHERE,

но перед предложением ORDER BY. После ключевых слов GROUP BY указывается список полей, включенных в выборку с помощью оператора SELECT. Причем нужно обязательно указывать *все* отбираемые поля (за исключением полей, относящихся к агрегирующим функциям), хотя порядок их перечисления после предложения GROUP BY может не соответствовать порядку списка после слова SELECT.

Синтаксис оператора SELECT с предложением GROUP BY следующий:

SELECT fieldl, field2,…, fieldN
FROM tablel {… table2, tableN}
WHERE условие
GROUP BY fieldl, field2,…, fieldN
ORDER BY fieldl {ASC | DESC}

Применение предложения GROUP BY без дополнительных функций дает такой же результат, как и применение предложения упорядочения ORDER BY.

Например, если выбрать из таблицы «Товары» два поля — «Наименование» и «Категория», а затем сгруппировать их с помощью запроса:

SELECT Наименование. Категория
FROM Товары
GROUP BY Категория, Наименование.

то результат выборки будет упорядочен по значению первого поля, указанного в предложении GROUP BY.

Если в запросе выбрать только одно поле и выполнить для него группировку, то результирующая выборка не будет содержать дублирующих друг друга записей.

Например, если выполнить запрос, аналогичный предыдущему, но выбрать только поле «Категория»:

SELECT Категория
FROM Товары
GROUP BY Категория.

Соединение таблиц. Виды соединений.

Выборка данных из нескольких таблиц. Как правило, информация, хранящаяся в базе данных, содержится в нескольких связанных между собой таблицах. Язык SQL позволяет создавать запросы, извлекающие данные из нескольких таблиц. При этом выполняется операция *соединения,* состоящая в объединении нескольких таблиц с целью поиска в них запрошенных данных.

Существует несколько способов соединения таблиц. Наиболее часто встречаются следующие:

- соединение равенства;
- соединение неравенства;
- внешние соединения.

Для задания вида соединения используется предложение WHERE, в котором вид соединения указывается с помощью операторов сравнения или логических операторов.

Соединение равенства. Данное соединение является наиболее часто используемым. Соединение равенства обычно производится по общему для нескольких таблиц полю (которое, как правило, является первичным ключом).

Синтаксис оператора выборки для этого способа соединения таблиц будет следующим:

SELECT tablel.fieldl, table2.field2 { tableN.fieldN}
FROM tablel, table2 { tableN}
WHERE tablel.common_fie1dl = table2.common_fieldl
{AND tablel.common_fie1d2 = table2.common_field2}

При формировании запроса на выборку из нескольких таблиц в списке полей после слова SELECT перед именем поля обычно указывается имя таблицы, к которой это поле относится. Такое действие называется *квалификацией* полей запроса. Квалификация обязательна только для полей, имеющих одинаковые имена в разных таблицах, из которых производится выборка.

Рассмотрим пример выборки из двух таблиц с использованием соединения равенства. Выберем из таблицы «Клиенты» поля,

содержащие сведения об именах клиентов, а из таблицы «Продажи» — поля, в которых содержатся сведения о покупках, сделанных клиентами. Для связывания таблиц воспользуемся общим для обеих таблиц полем «Код клиента»:

SELECT Клиенты.Фамилия, Клиенты.Имя,
Клиенты.Отчество, Продажи.Продано
FROM Клиенты, Продажи
WHERE Клиенты.[Код клиента]=Продажи.[Код клиента]

При связывании таблиц можно использовать предложение группировки. Изменим рассмотренный выше запрос таким образом, чтобы результаты были сгруппированы по полям «Фамилия», «Имя», «Отчество» и для каждого клиента выводилось суммарное количество покупок:

SELECT Клиенты.Фамилия, Клиенты.Имя, Клиенты.Отчество, SUM(Продажи.Продано) AS [Количество покупок]
FROM Клиенты. Продажи
WHERE Клиенты.[Код клиента]=Продажи.[Код клиента]
GROUP BY Клиенты.Фамилия, Клиенты.Имя, Клиенты.Отчество

Соединение неравенства. В случае применения соединения неравенства информация из двух таблиц объединяется таким образом, чтобы значения в заданном поле одной таблицы не совпадали со значениями соответствующего ему поля в другой таблице.

Синтаксис запроса при соединении неравенства аналогичен предыдущему случаю, только вместо оператора «=» в предложении WHERE используются операторы **«<>», «<», «>»** и т. п.

SELECT tablel.fieldl, table2.field2 { tableN.fieldN}
FROM tablel, table2 {tableN}
WHERE tablel.common_fieldl <> table2.coramon_fieldl
{AND tablel.common_field2 > table2.common_field2}

Соединения неравенства используются довольно редко. В частности, для базы данных, используемой нами в качестве прак-

тической модели, довольно трудно привести осмысленный пример такого соединения.

Внешние соединения. При использовании внешнего соединения результат запроса будет содержать все записи одной из таблиц, даже в том случае, если в связанной с ней таблице отсутствуют совпадающие значения. Этот тип соединения реализуется с помощью оператора OUTER JOIN.

Внешние соединения подразделяются на три группы:

• левое внешнее соединение, **LEFT OUTER JOIN** — выборка будет содержать все записи таблицы, имя которой указано слева от оператора OUTER JOIN;

• правое внешнее соединение, **RIGHT OUTER JOIN** — выборка будет содержать все записи таблицы, имя которой указано справа от оператора OUTER JOIN;

• полное внешнее соединение, **FULL OUTER JOIN** — в выборку включаются все записи из правой и левой таблицы.

Для внешнего соединения условие соединения указывается не с помощью предложения WHERE, а входит в оператор OUTER JOIN после ключевого слова ON:

SELECT tablel.fieldl, table2.field2 { tableN.fieldN}
FROM tablel
LEFT | RIGHT | FULL {OUTER} JOIN table2
ON условие
(LEFT | RIGHT | FULL {OUTER} JOIN table3
ON условие}

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ

Контрольная работа выполняется по 10 вариантам. Каждый ВАРИАНТ СОДЕРЖИТ ПЯТЬ ВОПРОСОВ.

Вариант контрольной работы определяется по номеру зачетной книжки и соответствует последней цифре. Если последней

цифрой является ноль, то выбирается десятый вариант.

При выполнении работы следует соблюдать научную терминологию и обозначения.

Контрольную работу следует оформлять в печатном виде в соответствии со следующими требованиями:

- Печать на бумаге формата А4(210Х297 мм).
- Поля документа:
  - верхнее: 2 см;
  - левое: 3 см;
  - нижнее: 2 см;
  - правое: 1,5 см.
- Шрифт: Times New Roman, 12 пт., полуторный междустрочный интервал.
- Абзац: выравнивание по ширине, отступ первой строки 1,25 см.
- Нумерация страниц: внизу страницы, от центра, первая страница не нумеруется.

Контрольная работа должна быть оформлена в следующем порядке:

- Титульный лист установленного образца (указывают учебный шифр, наименование дисциплины, курс, отделение, индекс учебной группы, фамилию, имя и отчество исполнителя, точный почтовый адрес).
- Задания выполненные в соответствии с вариантом.
- Список использованной литературы.
- Приложения (при необходимости).

Список использованной литературы должен включать учебно-методические пособия, учебники, по которым выполнена работа. На последнем листе ставится подпись исполнителя и оставляется место для рецензии.

Контрольные вопросы и условия заданий должны предшествовать ответам. Содержание ответов должно быть четким и

кратким. Все описания выполнения определенных последовательностей действий с программами должны сопровождаться комментариями относительно их содержания и предназначения.

В установленные учебным графиком сроки студент направляет выполненную работу для проверки на заочное отделение. Предельный срок сдачи за две недели до начала сессии.

Таблица вариантов контрольной работы

Номер варианта / Номер задания	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	11	12	13	14	15	16	17	18	19	20
3	21	22	23	24	25	26	27	28	29	30
4	31	32	33	34	35	36	37	38	39	40
5	41	42	43	44	45	46	47	48	49	50

КОНТРОЛЬНЫЕ ЗАДАНИЯ

Задание для контрольной работы

Контрольная работа состоит из ответа на теоретические вопросы и выполнения практического задания согласно установленному варианту(ПР)

1. Понятие алгоритма, свойства, способы описания.
2. Виды алгоритмов.
3. Алгоритмизация, как базовая составляющая технологического процесса.

4. От простого кодирования до языков высокого уровня. Виды языков их классификация, понятие систем программирования.
5. Этапы технологического процесса создания программного продукта.
6. Компиляция программы. Интерпретатор. Транслятор.
7. Структура программы: заголовок, описательная часть, исполнительная часть.
8. Алфавит языка. Разделители. Ключевые слова. Стандартные идентификаторы. Идентификаторы пользователя. Назначения меток в программе.
9. Понятие простых типов данных. Функции для порядковых типов данных.
10. Целочисленные и литерные типы данных. Примеры описания.
11. Вещественные и логические типы данных. Примеры описания.
12. Перечисляемые типы данных. Примеры описания.
13. Интервальные типы данных. Примеры описания.
14. Структурные типы данных: строки.
15. Структурные типы данных: массивы. Формат описания массива. Обращение к элементу массива.
16. Статические и динамические массивы.
17. Структурные типы данных: множества. Формат описания множества. Операции над множествами.
18. Структурные типы данных: записи.
19. Структурные типы данных: файлы.
20. Структурные типы данных: вариантные.
21. Арифметические и логические выражения.
22. Операторы ввода и вывод информации. Оператор присваивания.
23. Условный оператор. Принцип работы оператора. Пример использования.
24. Оператор выбора. Принцип работы оператора. Пример использования.

25. Операторы цикла: оператор цикла с параметром. Принцип работы оператора. Пример использования.
26. Операторы цикла: оператор цикла с постусловием. Принцип работы оператора. Пример использования.
27. Операторы цикла: оператор цикла с предусловием. Принцип работы оператора. Пример использования.
28. Подпрограммы-функции: описание, вызов, рекурсия.
29. Подпрограммы – процедуры: описание, вызов. Процедуры без параметров, процедуры с параметрами,
30. Файлы и работа с ними. Способы описания файлов. Доступ к файлам. Процедуры и функции для работы с файлами.
31. Основы теории баз данных. Классификация БД и СУБД.
32. Модели данных. Реляционная модель данных.
33. Понятие реляционной алгебры. Операции реляционной алгебры: объединение, пересечение, разность, произведение, выборка, проекция, соединение, деление.
34. Логическое проектирование БД. Физическое проектирование БД.
35. Объекты БД Access. Графический интерфейс Access.
36. Типы данных. Создание схемы базы данных. Ввод данных в таблицы.
37. Создание пользовательского интерфейса с помощью форм, создание отчетов
38. Создание запросов. Общий вид оператора SELECT. Устранение дублирующих строк (предикат DISTINCT) Выборка по условию. Предложение WHERE. Операции сравнения, логические операторы. Оператор IN, Оператор BETWEEN, Оператор LIKE.
39. Агрегирование данных. Предложение GROUP BY, предложение HAVING/ Упорядочивание значений полей. Использование псевдонимов в запросах. Вложенные запросы.
40. Объединение таблиц (внутреннее объединение, внешнее объединение, объединение более, чем двух таблиц).
41. Составить программу для работы с матрицей, используя операторы цикла. Даны матрицы A и B размером 2x3. По-

лучить матрицу C=A+B.

42. Составить программу для работы с матрицей, используя операторы цикла. Даны матрицы A и B размером 3x4. Получить матрицу C=A*B.
43. Составить программу для работы с матрицей, используя операторы цикла. Найти максимальный элемент матрицы A размером 3x3 и его индексы.
44. Составить программу для работы с матрицей, используя операторы цикла. Найти минимальный элемент матрицы B размером 3x4 и его индексы.
45. Составить программу для работы с матрицей, используя операторы цикла. Найти суммы строк матрицы A размером 3x4.
46. Составить программу для работы с матрицей, используя операторы цикла. Найти суммы столбцов матрицы B размером 3x5.
47. Составить программу для работы с матрицей, используя операторы цикла. Найти сумму элементов главной диагонали матрицы C размером 4x4.
48. Составить программу для работы с матрицей, используя операторы цикла. Найти произведение элементов главной диагонали матрицы B размером3x3.
49. Составить программу для работы с матрицей, используя операторы цикла. Найти среднее арифметическое элементов матрицы A размером 3x3.
50. Составить программу для работы с матрицей, используя операторы цикла. Найти сумму элементов по периметру матрицы C размером 4x4.