ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ОРЕНБУРГСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ОБУЧАЮЩИХСЯ ПО ОСВОЕНИЮ ДИСЦИПЛИНЫ

Б1.В.ДВ.07.02 Базы данных

Специальность 38.05.01 Экономическая безопасность

Специализация Экономико-правовое обеспечение экономической безопасности

Форма обучения очная

1. КОНСПЕКТ ЛЕКЦИЙ

1. 1 Лекция №1, №2 (4 часа).

Тема: «История создания дисциплины»

1.1.1 Вопросы лекции:

1. Краткая характеристика дисциплины
2. История создания дисциплины

1.1.2 Краткое содержание вопросов:

1. Краткая характеристика дисциплины.

Цель преподавания дисциплины - формирование представления о роли и месте знаний по базам данных при практическом использовании их в своей профессиональной деятельности.

Дисциплина «Базы данных» относится к профессиональному циклу, вариативная части.

Компетенции обучающегося, формируемые в результате освоения дисциплины

1. Способность применять математический инструментарий для решения экономических задач (ОК-15).

2. Способность выбирать инструментальные средства для обработки финансовой, бухгалтерской и иной экономической информации и обосновывать свой выбор (ПК-32).

В результате освоения дисциплины обучающийся должен:

Знать:

- базы данных и системы управления базами данных для информационных систем различного назначения.
- основные положений концепции баз данных и принципов построения баз данных;
- классификацию и модели данных;
- современные системы управления базами данных и их место в системах обработки данных.

Уметь:

- разрабатывать инфологические и датологические схемы баз данных;
- создавать базы данных;
- реализовывать простые информационные технологии в экранном интерфейсе современных систем управления базами данных;
- применять методики проектирования баз данных для конкретных предметных областей;
- эффективно работать индивидуально при разработке баз данных;
- эффективно работать в качестве члена команды по разработке программных средств.

Владеть:

- методами описания схем баз данных;
- навыками работы в качестве члена группы при разработке баз данных;
- способностью брать на себя ответственность за результаты работы по разработке программных средств.

Взаимосвязь планируемых результатов обучения по дисциплине (знания, умения, навыки и (или) опыт деятельности) и планируемых результатов освоения образовательной программы (компетенций обучающегося):

- ОК-15: способность применять математический инструментарий для решения экономических задач:

-Знания:

Этап 1. Классификацию и модели данных.

Этап 2. Базы данных и системы управления базами данных для информационных систем различного назначения.

- Умения:
Этап 1: Применять методики проектирования баз данных для конкретных предметных областей.
Этап 2: Реализовывать простые информационные технологии в экранном интерфейсе современных систем управления базами данных; эффективно работать индивидуально при разработке баз данных.
- Навыки и (или) опыт деятельности:
Этап 1: Навыками самостоятельного овладения новыми знаниями.
Этап 2: Методами описания схем баз данных.

- ПК-32: способность выбирать инструментальные средства для обработки финансовой, бухгалтерской и иной экономической информации и обосновывать свой выбор
-Знания:
Этап 1. Основные положений концепции баз данных и принципов построения баз данных.
Этап 2: Современные системы управления базами данных и их место в системах обработки данных.
- Умения:
Этап 1: Создавать базы данных.
Этап 2: Разрабатывать инфологические и датологические схемы баз данных; эффективно работать в качестве члена команды по разработке программных средств.
- Навыки и (или) опыт деятельности:
Этап 1: Навыками работы в качестве члена группы при разработке баз данных.
Этап 2: Способностью брать на себя ответственность за результаты работы по разработке программных средств

Общая трудоемкость дисциплины «Базы данных» составляет 3 ЗЕ (108 часов.

2. История создания дисциплины

В истории вычислительной техники можно проследить развитие двух основных областей ее использования. Первая область — применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную. Развитие этой области способствовало интенсификации методов численного решения сложных математических задач, появлению языков программирования, ориентированных на удобную запись численных алгоритмов, становлению обратной связи с разработчиками новых архитектур ЭВМ. Характерной особенностью данной области применения вычислительной техники является наличие сложных алгоритмов обработки, которые применяются к простым по структуре данным, объем которых сравнительно невелик.

Вторая область, которая непосредственно относится к нашей теме, — это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. Информационная система представляет собой программно-аппаратный комплекс, обеспечивающий выполнение следующих функций:

· надежное хранение информации в памяти компьютера;

· выполнение специфических для данного приложения преобразований информации и вычислений;

· предоставление пользователям удобного и легко осваиваемого интерфейса.

Обычно такие системы имеют дело с большими объемами информации, имеющей достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, автоматизированные системы управления предприятиями, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

Вторая область использования вычислительной техники возникла несколько позже первой. Это связано с тем, что на заре вычислительной техники возможности компьютеров по хранению информации были очень ограниченными. Говорить о надежном и долговременном

хранении информации можно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная (основная) память компьютеров этим свойством обычно не обладает. В первых компьютерах использовались два вида устройств внешней памяти — магнитные ленты и барабаны. Емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные же барабаны (они ближе всего к современным магнитным дискам с фиксированными головками) давали возможность произвольного доступа к данным, но имели ограниченный объем хранимой информации.

Эти ограничения не являлись слишком существенными для чисто численных расчетов. Даже если программа должна обработать (или произвести) большой объем информации, при программировании можно продумать расположение этой информации во внешней памяти (например, на последовательной магнитной ленте), обеспечивающее эффективное выполнение этой программы. Однако в информационных системах совокупность взаимосвязанных информационных объектов фактически отражает модель объектов реального мира. А потребность пользователей в информации, адекватно отражающей состояние реальных объектов, требует сравнительно быстрой реакции системы на их запросы. И в этом случае наличие сравнительно медленных устройств хранения данных, к которым относятся магнитные ленты и барабаны, было недостаточным.

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

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

1.2 Лекция №3, 4 (4 часа).

Тема: «Система баз данных»

1.2.1 Вопросы лекции:

1. Определение и назначение баз данных
2. Файловые системы хранения данных
3. Области применения баз данных

1.2.2 Краткое содержание вопросов:

1. Определение и назначение баз данных.

С самого начала развития вычислительной техники образовались два основных направления ее использования.

Первое направление — применение вычислительной техники для выполнения численных расчетов, которые слишком долго или вообще невозможно производить вручную.

Второе направление — это использование средств вычислительной техники в автоматических или автоматизированных информационных системах. В самом широком смысле информационная система представляет собой программный комплекс, функции которого состоят в поддержке надежного хранения информации в памяти компьютера,

выполнении специфических для данного приложения преобразований информации и/или вычислений, предоставлении пользователям удобного и легко осваиваемого интерфейса. Обычно объемы информации, с которыми приходится иметь дело таким системам, достаточно велики, а сама информация имеет достаточно сложную структуру. Классическими примерами информационных систем являются банковские системы, системы резервирования авиационных или железнодорожных билетов, мест в гостиницах и т. д.

Второе направление возникло несколько позже первого. Это связано с тем, что на заре вычислительной техники компьютеры обладали ограниченными возможностями. Надежное и долговременное хранение информации возможно только при наличии запоминающих устройств, сохраняющих информацию после выключения электрического питания. Оперативная память этим свойством обычно не обладает. Используемые в ранних ЭВМ два вида устройств внешней памяти, магнитные ленты и барабаны были несовершенными. Емкость магнитных лент была достаточно велика, но по своей физической природе они обеспечивали последовательный доступ к данным. Магнитные барабаны давали возможность произвольного доступа к данным, но были ограниченного размера. Появление соответствующих носителей данных, в первую очередь, жестких дисков, дало толчок к работам по созданию информационных компьютерных систем.

Основу любой информационной системы составляет база данных — это набор данных, которые организованы специальным образом.

В настоящее время действует Закон «О правовой охране программ для электронных вычислительных машин и баз данных» № 3523-1 от 23.09.92. В этом законе дается следующее определение базы данных: «**База данных** — это объективная форма представления и организации совокупности данных (например, статей, расчетов), систематизированных таким образом, чтобы эти данные могли быть найдены и обработаны с помощью ЭВМ».

В основе решения многих задач лежит обработка информации. Для облегчения обработки информации создаются информационные системы (ИС). Автоматизированными называют ИС, в которых применяют технические средства, в частности ЭВМ. Большинство существующих ИС являются автоматизированными, поэтому для краткости просто будем называть их ИС.

В широком понимании под определение ИС подпадает любая система обработки информации. По области применения ИС можно разделить на системы, используемые в производстве, образовании, здравоохранении, науке, военном деле, социальной сфере, торговле и других отраслях. По целевой функции ИС можно условно разделить на следующие основные категории: управляющие, информационно-справочные, поддержки принятия решений.

Заметим, что иногда используется более узкая трактовка понятия ИС как совокупности аппаратно-программных средств, задействованных для решения некоторой прикладной задачи. В организации, например, могут существовать информационные системы, на которых соответственно возложены следующие задачи: учет кадров и материально-технических средств, расчет с поставщиками и заказчиками, бухгалтерский учет и т. п.

Банк данных является разновидностью ИС, в которой реализованы функции централизованного хранения и накопления обрабатываемой информации, организованной в одну или несколько баз данных. Банк данных (БнД) в общем случае состоит из следующих компонентов: базы (нескольких баз) данных, системы управления базами данных, словаря данных, администратора, вычислительной системы и обслуживающего персонала. Вкратце рассмотрим названные компоненты и некоторые связанные с ними важные понятия.

База данных (БД) представляет собой совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Логическую структуру хранимых в базе данных называют моделью представления данных. К основным моделям представления данных (моделям данных) относятся следующие: иерархическая, сетевая, реляционная, постреляционная, многомерная и объектно-ориентированная (см. раздел 2).

Система управления базами данных (СУБД) - это комплекс языковых и программных средств, предназначенный для создания, ведения и совместного использования

БД многими пользователями. Обычно СУБД различают по используемой модели данных. Так, СУБД, основанные на использовании реляционной модели данных, называют реляционными СУБД.

Одними из первых СУБД являются следующие системы: IMS (IBM, 1968 г.), IDMS (Cullinet, 1971 г.), ADABAS (Software AG, 1969 г.) и ИНЭС (ВНИИСИ АН СССР, 1976 г.). Количество современных систем управления базами данных исчисляется тысячами.

Приложение представляет собой программу или комплекс программ, обеспечивающих автоматизацию обработки информации для прикладной задачи. Нами рассматриваются приложения, использующие БД. Приложения могут создаваться в среде или вне среды СУБД - с помощью системы программирования, использующей средства доступа к БД, к примеру, Delphi или C++ Builder. Приложения, разработанные в среде СУБД, часто называют приложениями СУБД, а приложения, разработанные вне СУБД, - внешними приложениями.

Для работы с базой данных зачастую достаточно средств СУБД и не нужно использовать приложения, создание которых требует программирования. Приложения разрабатывают главным образом в случаях, когда требуется обеспечить удобство работы с БД неквалифицированным пользователям или интерфейс СУБД не устраивает пользователей.

Словарь данных (СД) представляет собой подсистему БнД, предназначенную для централизованного хранения информации о структурах данных, взаимосвязях файлов БД друг с другом, типах данных и форматах их представления, принадлежности данных пользователям, кодах защиты и разграничения доступа и т.п. Функционально СД присутствует во всех БнД, но не всегда выполняющий эти функции компонент имеет именно такое название. Чаще всего функции СД выполняются СУБД и вызываются из основного меню системы или реализуются с помощью ее утилит.

Администратор базы данных (АБД) есть лицо или группа лиц, отвечающих за выработку требований к БД, ее проектирование, создание, эффективное использование и сопровождение. В процессе эксплуатации АБД обычно следит за функционированием информационной системы, обеспечивает защиту от несанкционированного доступа, контролирует избыточность, непротиворечивость, сохранность и достоверность хранимой в БД информации. Для однопользовательских информационных систем функции АБД обычно возлагаются на лиц, непосредственно работающих с приложением БД.

Вычислительной сети АБД, как правило, взаимодействует с администратором сети. В обязанности последнего входят контроль за функционированием аппаратно-программных средств сети, реконфигурация сети, восстановление программного обеспечения после сбоев и отказов оборудования, профилактические мероприятия и обеспечение разграничения доступа.

Вычислительная система (ВС) представляет собой совокупность взаимосвязанных и согласованно действующих ЭВМ или процессоров и других устройств, обеспечивающих автоматизацию процессов приема, обработки и выдачи информации потребителям. Поскольку основными функциями БнД являются хранение и обработка данных, то используемая ВС, наряду с приемлемой мощностью центральных процессоров (ЦП) должна иметь достаточный объем оперативной и внешней памяти прямого доступа.

Обслуживающий персонал выполняет функции поддержания технических и программных средств в работоспособном состоянии. Он проводит профилактические, регламентные, восстановительные и другие работы по планам, а также по мере необходимости.

2. Файловые системы хранения данных.

Фа́йловая систе́ма (англ. file system) — порядок, определяющий способ организации, хранения и именования данных на носителях информации в компьютерах, а также в другом электронном оборудовании: цифровых фотоаппаратах, мобильных телефонах и т. п. Файловая система определяет формат содержимого и способ физического хранения информации, которую принято группировать в виде файлов. Конкретная файловая система определяет размер имен файлов и (каталогов), максимальный возможный размер файла и

раздела, набор атрибутов файла. Некоторые файловые системы предоставляют сервисные возможности, например, разграничение доступа или шифрование файлов.

Файловая система связывает носитель информации с одной стороны и API для доступа к файлам — с другой. Когда прикладная программа обращается к файлу, она не имеет никакого представления о том, каким образом расположена информация в конкретном файле, так же, как и на каком физическом типе носителя (CD, жёстком диске, магнитной ленте, блоке флеш-памяти или другом) он записан. Всё, что знает программа — это имя файла, его размер и атрибуты. Эти данные она получает от драйвера файловой системы. Именно файловая система устанавливает, где и как будет записан файл на физическом носителе (например, жёстком диске).

С точки зрения операционной системы (ОС), весь диск представляет собой набор кластеров (как правило, размером 512 байт и больше). Драйверы файловой системы организуют кластеры в файлы и каталоги (реально являющиеся файлами, содержащими список файлов в этом каталоге). Эти же драйверы отслеживают, какие из кластеров в настоящее время используются, какие свободны, какие помечены как неисправные.

Однако файловая система не обязательно напрямую связана с физическим носителем информации. Существуют виртуальные файловые системы, а также сетевые файловые системы, которые являются лишь способом доступа к файлам, находящимся на удалённом компьютере.

Практически всегда файлы на дисках объединяются в каталоги.

В простейшем случае все файлы на данном диске хранятся в одном каталоге. Такая одноуровневая схема использовалась в CP/M и в первой версии MS-DOS 1.0. Иерархическая файловая система со вложенными друг в друга каталогами впервые появилась в Multics, затем в UNIX.

Wiki.txt
Tornado.jpg
Notepad.exe
(Одноуровневая файловая система)

Каталоги на разных дисках могут образовывать несколько отдельных деревьев, как в DOS/Windows, или же объединяться в одно дерево, общее для всех дисков, как в UNIX-подобных системах.

C:
 \Program files
  \CDEx
   \CDEx.exe
   \CDEx.hlp
   \mppenc.exe
 \Мои документы
  \Wiki.txt
  \Tornado.jpg
D:
 \Music
  \ABBA
   \1974 Waterloo
   \1976 Arrival
    \Money, Money, Money.ogg
   \1977 The Album
(Иерархическая файловая система Windows/DOS)

В UNIX существует только один корневой каталог, а все остальные файлы и каталоги вложены в него. Чтобы получить доступ к файлам и каталогам на каком-нибудь диске, необходимо смонтировать этот диск командой mount. Например, чтобы открыть файлы на CD, нужно, говоря простым языком, сказать операционной системе: «возьми файловую систему на этом компакт-диске и покажи её в каталоге /mnt/cdrom». Все файлы и каталоги, находящиеся на CD, появятся в этом каталоге /mnt/cdrom, который называется точкой

монтирования (англ. mount point). В большинстве UNIX-подобных систем съёмные диски (дискеты и CD), флеш-накопители и другие внешние устройства хранения данных монтируют в каталог /mnt, /mount или /media. Unix и UNIX-подобные операционные системы также позволяют автоматически монтировать диски при загрузке операционной системы.

```
/
  /usr
    /bin
      /arch
      /ls
      /raw
    /lib
      /libhistory.so.5.2
      /libgpm.so.1
  /home
    /lost+found
      /host.sh
    /guest
      /Pictures
        /example.png
      /Video
        /matrix.avi
        /news
          /lost_ship.mpeg
```

(Иерархическая файловая система в Unix и UNIX-подобных операционных системах)

Обратите внимание на использование слешей в файловых системах Windows, UNIX и UNIX-подобных операционных системах (В Windows используется обратный слеш «\», а в UNIX и UNIX-подобных операционных системах простой слеш «/»)

Кроме того, следует отметить, что вышеописанная система позволяет монтировать не только файловые системы физических устройств, но и отдельные каталоги (параметр --bind) или, например, образ ISO (опция loop). Такие надстройки, как FUSE, позволяют также монтировать, например, целый каталог на FTP и ещё очень большое количество различных ресурсов.

Ещё более сложная структура применяется в NTFS и HFS. В этих файловых системах каждый файл представляет собой набор атрибутов. Атрибутами считаются не только традиционные только для чтения, системный, но и имя файла, размер и даже содержимое. Таким образом, для NTFS и HFS то, что хранится в файле, — это всего лишь один из его атрибутов.

Если следовать этой логике, один файл может содержать несколько вариантов содержимого. Таким образом, в одном файле можно хранить несколько версий одного документа, а также дополнительные данные (значок файла, связанная с файлом программа). Такая организация типична для HFS на Macintosh.

По предназначению файловые системы можно классифицировать на нижеследующие категории:

- Для носителей с произвольным доступом (например, жёсткий диск): FAT32, HPFS, ext2 и др. Поскольку доступ к дискам в несколько раз медленнее, чем доступ к оперативной памяти, для прироста производительности во многих файловых системах применяется асинхронная запись изменений на диск. Для этого применяется либо журналирование, например в ext3, ReiserFS, JFS, NTFS, XFS, либо механизм soft updates и др. Журналирование широко распространено в Linux, применяется в NTFS. Soft updates — в BSD системах.
- Для носителей с последовательным доступом (например, магнитные ленты): QIC и др.
- Для оптических носителей — CD и DVD: ISO9660, HFS, UDF и др.
- Виртуальные файловые системы: AEFS и др.

- Сетевые файловые системы: NFS, CIFS, SSHFS, GmailFS и др.

- Для флэш-памяти: YAFFS, ExtremeFFS, exFAT.

Немного выпадают из общей классификации специализированные файловые системы: ZFS (собственно файловой системой является только часть ZFS), VMFS (т. н. кластерная файловая система, которая предназначена для хранения других файловых систем) и др.

Основные функции любой файловой системы нацелены на решение следующих задач:

- именование файлов;

- программный интерфейс работы с файлами для приложений;

- отображения логической модели файловой системы на физическую организацию хранилища данных;

- организация устойчивости файловой системы к сбоям питания, ошибкам аппаратных и программных средств;

- содержание параметров файла, необходимых для правильного его взаимодействия с другими объектами системы (ядро, приложения и пр.).

В многопользовательских системах появляется ещё одна задача: защита файлов одного пользователя от несанкционированного доступа другого пользователя, а также обеспечение совместной работы с файлами, к примеру, при открытии файла одним из пользователей, для других этот же файл временно будет доступен в режиме «только чтение».

3. Области применения баз данных.

Автоматизированные информационные системы (АИС), основу которых составляют базы данных, появились в 60-х годах XX века в военной промышленности и бизнесе — там, где были накоплены значительные объемы полезных данных. Первоначально АИС были ориентированы лишь на работу с информацией фактического характера — числовыми или текстовыми характеристиками объектов. Затем по мере развития техники появилась возможность обработки текстовой информации на естественном языке.

Принципы хранения разных видов информации в АИС аналогичны, но алгоритмы ее обработки определяются характером ин формационных ресурсов. Соответственно различают два класса АИС: документальные и фактографические.

Документальные АИС служат для работы с документами на естественном языке. Наиболее распространенный тип документальных АИС — информационно-поисковые системы, предназначенные для накопления и подбора документов, удовлетворяющих заданным критериям. Эти системы могут выполнять просмотр и подборку монографий, публикаций в периодике, сообщений пресс- агентств, текстов законодательных актов и т.д.

Фактографические АИС оперируют фактическими сведениями, представленными в формализованном виде, и используются для решения задач обработки данных.

Обработка данных — специальный класс решаемых на ЭВМ задач, связанных с вводом, хранением, сортировкой, отбором и группировкой записей данных однородной структуры. К задачам этого класса относятся: учет товаров в магазинах и на складах; начисление зарплаты; управление производством, финансами, телекоммуникациями и т. п.

Различают фактографические АИС оперативной обработки данных, подразумевающие быстрое обслуживание относительно простых запросов от большого числа пользователей, и фактографические АИС аналитической обработки, ориентированные на выполнение сложных запросов, требующих проведения статистической обработки исторических (накопленных за некоторый промежуток времени) данных, моделирования процессов предметной области и прогнозирования развития этих процессов.

Таким образом, АИС применяются в следующих областях:

• организация хранилищ данных;

• системы анализа данных;

• системы принятия решений;

• мобильные и персональные базы данных;

• географические базы данных;

• мультимедиа базы данных;

• распределенные информационные системы.

Каждая информационная система в зависимости от назначения имеет дело с той или иной частью конкретного мира, которую принято называть ее предметной областью. Анализ предмет ной области является необходимым начальным этапом разработки любой информационной системы. Именно на этом этапе определяются информационные потребности всей совокупности пользователей будущей системы, которые, в свою очередь, предопределяют содержание ее базы данных. Предметная область конкретной информационной системы рассматривается, прежде все го, как некоторая совокупность реальных объектов, которые представляют интерес для ее пользователей. Примерами объектов предметной области могут служить персональные ЭВМ, программные продукты и их пользователи. Каждый из этих объектов обладает определенным набором свойств (атрибутов). Так, например, компьютер характеризуется названием фирмы-производителя, идёнтификатором модели, типом микропроцессора, объемом оперативной и внешней памяти, типом графической карты и т. д.

Информационный объект это описание некоторой сущности предметной области, т. е. реального объекта, процесса, явления или события. Информационный объект (сущность) образуется совокупностью логически взаимосвязанных атрибутов (свойств), представляющих собой качественные и количественные характеристики объекта (сущности).

Между объектами предметной области могут существовать связи, имеющие различный содержательный смысл. Эти связи могут быть обязательными или факультативными (необязательными).

Если вновь порожденный объект оказывается по необходимости связанным с каким-либо объектом предметной области, то между этими двумя объектами существует обязательная связь. В противном случае связь является факультативной. Например, обязательная связь Замещает существует между двумя объектами СОТРУДНИК и ДОЛЖНОСТЬ в предметной области кадровой информационной системы, т. е. каждый принимаемый в организацию сотрудник зачисляется на какую-либо должность и не может быть сотрудника, не замещающего какой-либо должности. В то же время связь Замещает между типами объектов СОТРУДНИК и ДОЛЖНОСТЬ является факультативной, поскольку могут существовать вакантные должности. Совокупность объектов предметной области и связей между ними характеризует структуру предметной области. Множество объектов предметной области, значения атрибутов объектов и связи между ними могут изменяться во времени. Изменения могут сводиться к появлению новых или исключению из рассмотрения некоторых существующих объектов в предметной области, установлению новых или разрушению существующих связей между ними. Следовательно, с каждым моментом времени можно сопоставить некоторое состояние предметной области.

Информационно-логическая модель (ИЛМ) — это совокупность информационных объектов (сущностей) предметной области и связей между ними. Процесс создания информационной модели начинается с определения концептуальных требований будущих пользователей БД. Требования отдельных пользователей интегрируются в едином обобщенном представлении, которое называют концептуальной моделью данной предметной области.

Такая модель отображает предметную область в виде взаимосвязанных объектов без указания способов их физического хранения.

Концептуальная модель представляет собой интегрированные концептуальные требования всех пользователей к базе данных данной предметной области. При этом усилия разработчика должны быть направлены в основном на структуризацию данных, принадлежащих будущим пользователям БД и выявление взаимосвязей между ними.

Возможно, что отраженные в концептуальной модели взаимосвязи между объектами окажутся впоследствии нереализуемыми средствами выбранной СУБд, что потребует ее изменения. Вер сия концептуальной модели, которая может быть реализована конкретной СУБд, называется логической моделью.

Логическая модель, отражающая логические связи между атрибутами объектов вне зависимости от их содержания и среды хранения, может быть реляционной, иерархической или сете вой. Таким образом, логическая модель отображает логические связи между информационными данными в данной концептуальной модели.

Различным пользователям в информационной модели соответствуют различные подмножества ее логической модели, которые называются внешними моделями пользователей. Таким образом, внешняя модель пользователя представляет собой отображение его концептуальных требований в логической модели и соответствует тем представлениям, которые этот пользователь получает о предметной области на основе логической модели. Следовательно, насколько хорошо спроектирована внешняя модель, настолько полно и точно информационная модель отображает предметную область и настолько полно и точно работает автоматизированная система управления этой предмет ной областью.

Логическая модель отображается в физическую память, которая может быть построена на электронных, магнитных, оптических, биологических или других принципах.

Внутренняя модель предметной области определяет размещение данных, методы доступа к ним и технику индексирования в данной логической модели и иначе называется физической моделью. Информационные данные любого пользователя в БД должны быть независимы от всех других пользователей, т. е. не должны оказывать влияния на существующие внешние модели. Это положение отражает первый уровень независимости данных. С другой стороны, внешние модели пользователей никак не связаны с типом физической памяти, в которой будут храниться данные, и с физическими методами доступа к этим данным. Это положение отражает второй уровень независимости данных.

1.3 Лекция № 5, 6 (4 часа).

Тема: «Топология баз данных»

1.3.1 Вопросы лекции:

1. Основные типы структур данных
2. Информационная модель данных и ее состав
3. Классификация баз данных

1.3.2 Краткое содержание вопросов:

1. Основные типы структур данных.

Необходимым условием хранения информации в памяти компьютера является возможность преобразования этой самой информации в подходящую для компьютера форму. В том случае, если это условие выполняется, следует определить структуру, пригодную именно для наличествующей информации, ту, которая предоставит требующийся набор возможностей работы с ней. Здесь под структурой понимается способ представления информации, посредством которого совокупность отдельно взятых элементов образует нечто единое, обусловленное их взаимосвязью друг с другом. Скомпонованные по каким-либо правилам и логически связанные межу собой, данные могут весьма эффективно обрабатываться, так как общая для них структура предоставляет набор возможностей управления ими – одно из того за счет чего достигаются высокие результаты в решениях тех или иных задач. Но не каждый объект представляем в произвольной форме, а возможно и

вовсе для него имеется лишь один единственный метод интерпретации, следовательно, несомненным плюсом для программиста будет знание всех существующих структур данных. Таким образом, часто приходиться делать выбор между различными методами хранения информации, и от такого выбора зависит работоспособность продукта.

Говоря о не вычислительной технике, можно показать ни один случай, где у информации видна явная структура. Наглядным примером служат книги самого разного содержания. Они разбиты на страницы, параграфы и главы, имеют, как правило, оглавление, то есть интерфейс пользования ими. В широком смысле, структурой обладает всякое живое существо, без нее органика не смогла бы существовать.

Вполне вероятно, читателю приходилось сталкиваться со структурами данных непосредственно в информатике, например, с теми, что встроены в язык программирования. Часто они именуются типами данных. К таковым относятся: массивы, числа, строки, файлы др.

Методы хранения информации, называемые «простыми», т. е. неделимыми на составные части, предпочтительнее изучать вместе с конкретным языком программирования, либо же глубоко углубляться в суть их работы. Поэтому здесь будут рассмотрены лишь «интегрированные» структуры, те которые состоят из простых, а именно: массивы, списки, деревья и графы.

Массивы

Массив – это структура данных с фиксированным и упорядоченным набором однотипных элементов (компонентов). Доступ к какому-либо из элементов массива осуществляется по имени и номеру (индексу) этого элемента. Количество индексов определяет размерность массива. Так, например, чаще всего встречаются одномерные (вектора) и двумерные (матрицы) массивы. Первые имеют один индекс, вторые – два. Пусть одномерный массив называется A, тогда для получения доступа к его i-ому элементу потребуется указать название массива и номер требуемого элемента: A[i]. Когда A – матрица, то она представляема в виде таблицы, доступ к элементам которой осуществляется по имени массива, а также номерам строки и столбца, на пересечении которых расположен элемент: A[i, j], где i – номер строки, j – номер столбца.

В разных языках программирования работа с массивами может в чем-то различаться, но основные принципы, как правило, везде одни. В языке Pascal, обращение к одномерному и двумерному массиву происходит точно так, как это показано выше, а, например, в C++ двумерный массив следует указывать так: A[i][j]. Элементы массива нумеруются поочередно. На то, с какого значения начинается нумерация, влияет язык программирования. Чаще всего этим значением является 0 или 1.

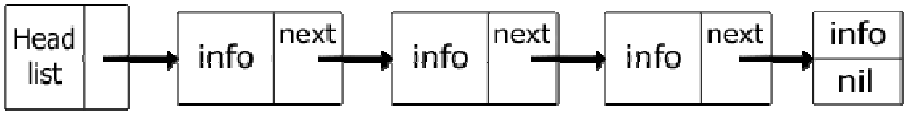
Массивы, описанного типа называются статическими, но существуют также массивы по определенным признакам отличные от них: динамические и гетерогенные. Динамичность первых характеризуется непостоянностью размера, т. е. по мере выполнения программы размер динамического массива может изменяться. Такая функция делает работу с данными более гибкой, но при этом приходится жертвовать быстродействием, да и сам процесс усложняется. Обязательный критерий статического массива, как было сказано, это однородность данных, единовременно хранящихся в нем. Когда же данное условие не выполняется, то массив является гетерогенным. Его использование обусловлено недостатками, которые имеются в предыдущем виде, но оно оправданно во многих случаях.

Таким образом, даже если Вы определились со структурой, и в качестве нее выбрали массив, то этого все же недостаточно. Ведь массив это только общее обозначение, род для некоторого числа возможных реализаций. Поэтому необходимо определиться с конкретным способом представления, с наиболее подходящим массивом.

Списки

Список – абстрактный тип данных, реализующий упорядоченный набор значений. Списки отличаются от массивов тем, что доступ к их элементам осуществляется последовательно, в то время как массивы – структура данных произвольного доступа. Данный абстрактный тип имеет несколько реализаций в виде структур данных. Некоторые из них будут рассмотрены здесь.

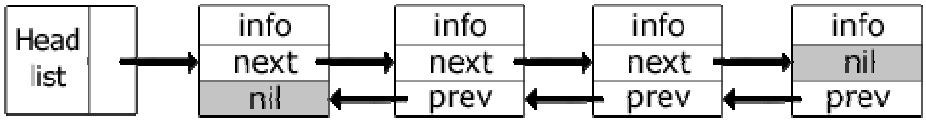
Список (связный список) – это структура данных, представляющая собой конечное множество упорядоченных элементов, связанных друг с другом посредствам указателей. Каждый элемент структуры содержит поле с какой-либо информацией, а также указатель на следующий элемент. В отличие от массива, к элементам списка нет произвольного доступа.



Односвязный список

В односвязном списке, приведенным выше, начальным элементом является Head list (голова списка [произвольное наименование]), а все остальное называется хвостом. Хвост списка составляют элементы, разделенные на две части: информационную (поле info) и указательную (поле next). В последнем элементе вместо указателя, содержится признак конца списка – nil.

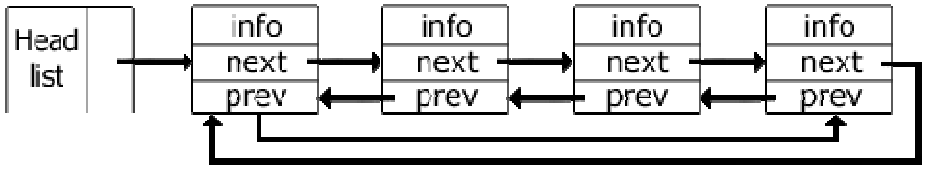
Односвязный список не слишком удобен, т. к. из одной точки есть возможность попасть лишь в следующую точку, двигаясь тем самым в конец. Когда кроме указателя на следующий элемент есть указатель и на предыдущий, то такой список называется двусвязным.



Двусвязный список

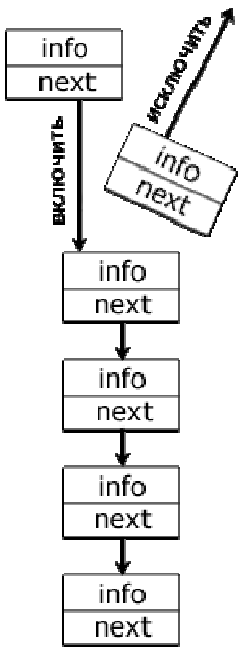
Возможность двигаться как вперед, так и назад полезна для выполнения некоторых операций, но дополнительные указатели требуют задействования большего количества памяти, чем таковой необходимо в эквивалентном односвязном списке.

Для двух видов списков описанных выше существует подвид, называемый кольцевым списком. Сделать из односвязного списка кольцевой можно добавив всего лишь один указатель в последний элемент, так чтобы он ссылался на первый. А для двусвязного потребуется два указателя: на первый и последний элементы.



Кольцевой список

Помимо рассмотренных видов списочных структур есть и другие способы организации данных по типу «список», но они, как правило, во многом схожи с разобранными, поэтому здесь они будут опущены.



Кроме различия по связям, списки делятся по методам работы с данными. О некоторых таких методах сказано далее.

Стек

Стек характерен тем, что получить доступ к его элементом можно лишь с одного конца, называемого вершиной стека, иначе говоря: стек – структура данных, функционирующая по принципу LIFO (last in — first out, «последним пришёл — первым вышел»). Изобразить эту структуру данных лучше в виде вертикального списка, например, стопки каких-либо вещей, где чтобы воспользоваться одной из них нужно поднять все те вещи, что лежат выше нее, а положить предмет можно лишь на вверх стопки.

В показанном односвязном списке операции над элементами происходят строго с одного конца: для включения

нужного элемента в пятую по счету ячейку необходимо исключить тот элемент, который занимает эту позицию. Если бы было, например 6 элементов, а вставить конкретный элемент требовалось также в пятую ячейку, то исключить бы пришлось уже два элемента.

Очередь

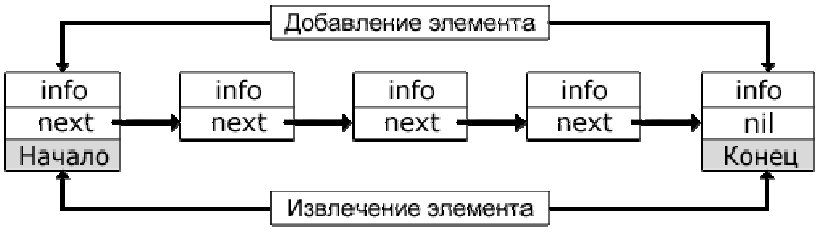
Структура данных «Очередь» использует принцип организации FIFO (First In, First Out — «первым пришёл — первым вышел»). В некотором смысле такой метод более справедлив, чем тот, по которому функционирует стек, ведь простое правило, лежащее в основе привычных очередей в различные магазины, больницы считается вполне справедливым, а именно оно является базисом этой структуры. Пусть данное наблюдение будет примером. Строго говоря, очередь – это список, добавление элементов в который допустимо, лишь в его конец, а их извлечение производиться с другой стороны, называемой началом списка.

Очередь

Эта структура одновременно функционирует по двум способам организации данных: FIFO и LIFO. Поэтому ее допустимо отнести к отдельной программной единице, полученной в результате суммирования двух предыдущих видов списка.

Дек

Дек (deque — double ended queue, «двухсторонняя очередь») – стек с двумя концами. Действительно, несмотря конкретный перевод, дек можно определять не только как двухстороннюю очередь, но и как стек, имеющий два конца. Это означает, что данный вид списка позволяет добавлять элементы в начало и в конец, и то же самое справедливо для операции извлечения.



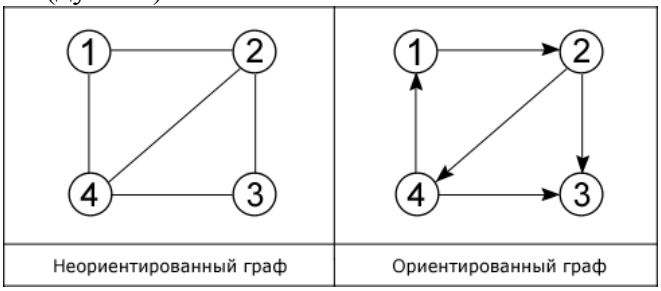
Дек

Эта структура одновременно работает по двум способам организации данных: FIFO и LIFO. Поэтому ее допустимо отнести к отдельной программной единице, полученной в результате суммирования двух предыдущих видов списка.

Графы

Раздел дискретной математики, занимающийся изучением графов, называется теорией графов. В теории графов подробно рассматриваются известные понятия, свойства, способы представления и области применения этих математических объектов. Нас же интересует, лишь те ее аспекты, которые важны в программировании.

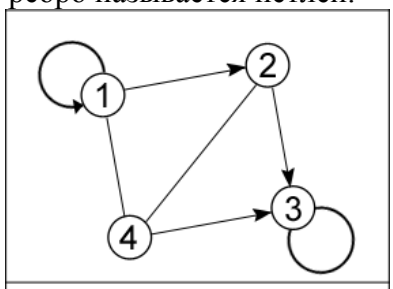
Граф – совокупность точек, соединенных линиями. Точки называются вершинами (узлами), а линии – ребрами (дугами).



Как показано на рисунке различают два основных вида графов: ориентированные и неориентированные. В первых ребра являются направленными, т. е. существует только одно доступное направление между двумя связными вершинами, например из вершины 1 можно пройти в вершину 2, но не наоборот. В неориентированном связном графе из каждой вершины можно пройти в каждую и обратно. Частный случай двух этих видов – смешанный граф. Он характерен наличием как ориентированных, так и неориентированных ребер.

Степень входа вершины – количество входящих в нее ребер, степень выхода – количество исходящих ребер.

Ребра графа необязательно должны быть прямыми, а вершины обозначаться именно цифрами, так как показано на рисунке. К тому же встречаются такие графы, ребрам которых поставлено в соответствие конкретное значение, они именуются взвешенными графами, а это значение – весом ребра. Когда у ребра оба конца совпадают, т. е. ребро выходит из вершины F и входит в нее, то такое ребро называется петлей.

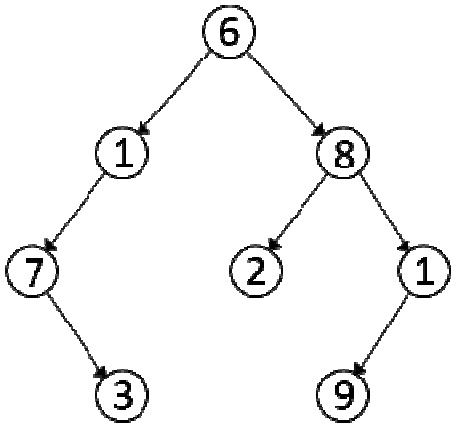


Смешанный граф с наличием петель

Графы широко используются в структурах, созданных человеком, например в компьютерных и транспортных сетях, web-технологиях. Специальные способы представления позволяют использовать граф в информатике (в вычислительных машинах). Самые известные из них: «Матрица смежности», «Матрица инцидентности», «Список смежности», «Список рёбер». Два первых, как понятно из названия, для репрезентации графа используют матрицу, а два последних – список.

Деревья

Дерево – структура данных



Неупорядоченное дерево

Дерево как математический объект это абстракция из соименных единиц, встречающихся в природе. Схожесть структуры естественных деревьев с графами определенного вида говорит о допущении установления аналогии между ними. А именно со связанными и вместе с этим ациклическими (не имеющими циклов) графами. Последние по своему строению действительно напоминают деревья, но в чем то и имеются различия, например, принято изображать математические деревья с корнем расположенным вверху, т. е. все ветви «растут» сверху вниз. Известно же, что в природе это совсем не так.

Поскольку дерево это по своей сути граф, у него с последним многие определения совпадают, либо интуитивно схожи. Так корневой узел (вершина 6) в структуре дерева – это единственная вершина (узел), характерная отсутствием предков, т. е. такая, что на нее не ссылается ни какая другая вершина, а из самого корневого узла можно дойти до любой из имеющихся вершин дерева, что следует из свойства связности данной структуры. Узлы, не ссылающиеся ни на какие другие узлы, иначе говоря, ни имеющие потомков называются листьями (2, 3, 9), либо терминальными узлами. Элементы, расположенные между корневым узлом и листьями – промежуточные узлы (1, 1, 7, 8). Каждый узел дерева имеет только одного предка, или если он корневой, то не имеет ни одного.

Поддерево – часть дерева, включающая некоторый корневой узел и все его узлы-потомки. Так, например, на рисунке одно из поддеревьев включает корень 8 и элементы 2, 1, 9.

С деревом можно выполнять многие операции, например, находить элементы, удалять элементы и поддеревья, вставлять поддеревья, находить корневые узлы для некоторых вершин и др. Одной из важнейших операций является обход дерева. Выделяются несколько методов обхода. Наиболее популярные из них: симметричный, прямой и обратный обход. При прямом обходе узлы-предки посещаются прежде своих потомков, а в обратном обходе, соответственно, обратная ситуация. В симметричном обходе поочередно просматриваются поддеревья главного дерева.

Представление данных в рассмотренной структуре выгодно в случае наличия у информации явной иерархии. Например, работа с данными о биологических родах и видах, служебных должностях, географических объектах и т. п. требует иерархически выраженной структуры, такой как математические деревья.

2. Информационная модель данных и ее состав.

Каждая информационная система в зависимости от ее назначения имеет дело с той или иной частью конкретного мира, которую принято называть предметной областью информационной системы. Анализ предметной области является необходимым начальным этапом разработки любой информационной системы. Именно на этом этапе определяются информационные потребности всей совокупности пользователей будущей системы, которые, в свою очередь, предопределяют содержание ее базы данных. Предметная область данной информационной системы рассматривается прежде всего как некоторая совокупность реальных объектов, которые представляют интерес для ее пользователей. Примерами объектов предметной области могут служить персональные ЭВМ, программные продукты, их пользователи. Каждый из них обладает определенным набором свойств (атрибутов). Так, компьютер характеризуется названием фирмы-производителя, идентификатором модели, типом микропроцессора, объемом оперативной и внешней памяти, типом графической карты и т. д.

Информационный объект — это описание некоторой сущности предметной области — реального объекта, процесса, явления или события. Информационный объект (сущность) образуется совокупностью логически взаимосвязанных атрибутов (свойств), представ-ляющих качественные и количественные характеристики объекта (сущности).

Между объектами предметной области могут существовать связи, имеющие различный содержательный смысл. Эти связи могут быть обязательными или факультативными.

Если вновь порожденный объект оказывается по необходимости связанным с каким-либо объектом предметной области, то между этими двумя объектами существует обязательная связь. В противном случае связь является факультативной (необязательной).

Обязательная связь «ЗАМЕЩАЕТ» существует, например, между двумя объектами СОТРУДНИК и ДОЛЖНОСТЬ в предметной области кадровой информационной системы. Каждый принимаемый в организацию сотрудник зачисляется на какую-либо должность и не может быть сотрудника, не замещающего какой-либо должности. В то же время связь

«ЗАМЕЩАЕТСЯ» между типами объектов СОТРУДНИК и ДОЛЖНОСТЬ является факультативной, поскольку могут существовать вакантные должности.

Совокупность объектов предметной области и связей между ними характеризует (типовую) структуру предметной области.

Множество объектов предметной области, значения атрибутов объектов и связи между ними могут изменяться во времени. Изменения могут сводиться к появлению новых или исключению из рас-смотрения некоторых существующих объектов в предметной области, установлению новых или разрушению существующих связей между ними. Поэтому с каждым моментом времени можно сопоставить некоторое состояние предметной области.

Информационно-логическая модель (ИЛМ) — совокупность ин-формационных объектов (сущностей) предметной области и связей между ними.

3. Классификация баз данных.

В силу многогранности баз данных и СУБД (комплекса технических и программных средств, для хранения, поиска, защиты и использования данных) имеется множество классификационных признаков.

Классификация БД по основным признакам приведена на рис. 1.

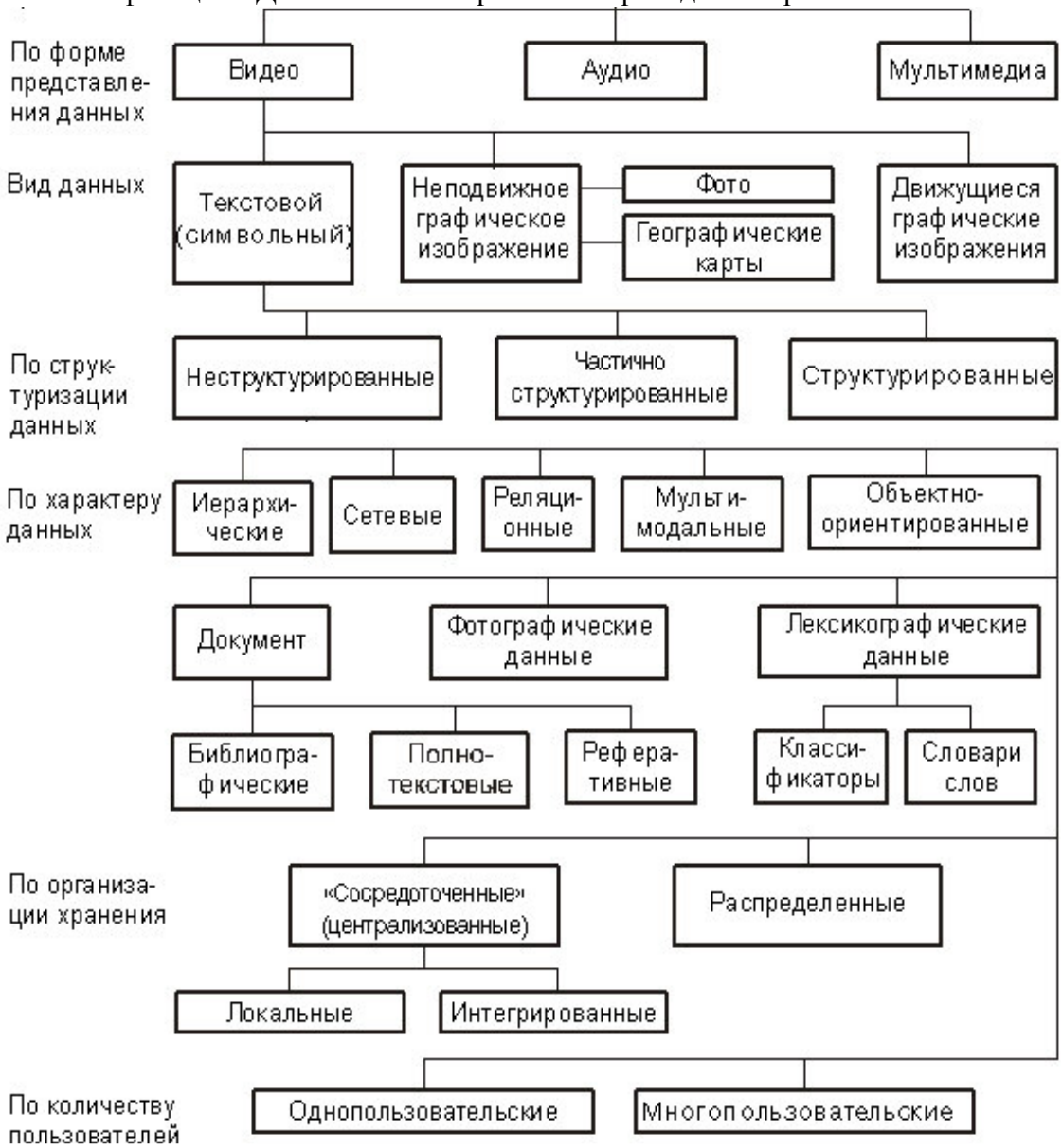


Рис. 1. Классификация баз данных

Базы данных могут классифицироваться и с точки зрения экономической: по условиям предоставления услуг - бесплатные и платные (бесприбыльные, коммерческие); по

форме собственности - государственные, негосударственные; по степени доступности - общедоступные, с ограниченным кругом пользователей.

1.4 Лекция №7, 8 (4 часа).

Тема: «Введение в реляционную модель данных»

1.4.1 Вопросы лекции:

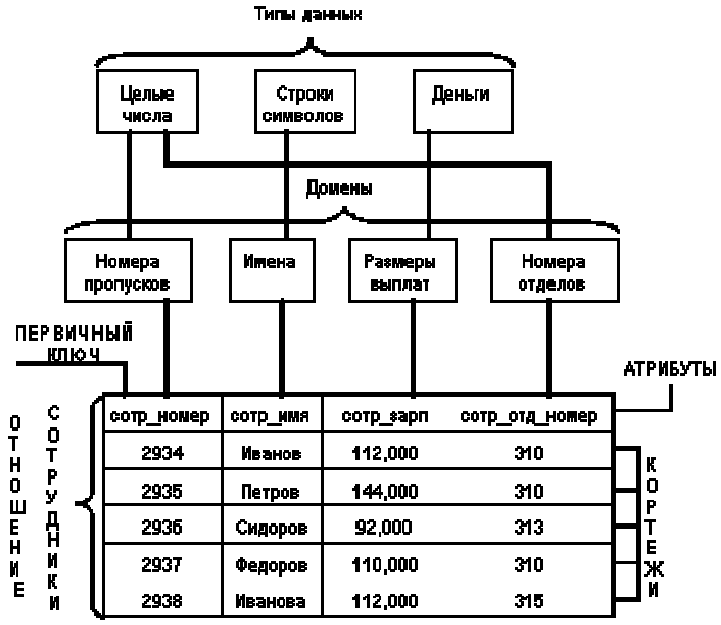
1. Основные понятия реляционных баз данных
2. Фундаментальные свойства отношений
3. Реляционная модель данных

1.4.2 Краткое содержание вопросов:

1. Основные понятия реляционных баз данных.

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

Для начала покажем смысл этих понятий на примере отношения СОТРУДНИКИ, содержащего информацию о сотрудниках некоторой организации:



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 "темпоральных" 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 трех типов: строки символов, целые числа и "деньги".

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. Если вычисление этого логического выражения дает результат "истина", то элемент данных является элементом домена.

Наиболее правильной интуитивной трактовкой понятия домена является понимание домена как допустимого потенциального множества значений данного типа. Например, домен "Имена" в нашем примере определен на базовом типе строк символов, но в число его

значений могут входить только те строки, которые могут изображать имя (в частности, такие строки не могут начинаться с мягкого знака).

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 пропусков" и "Номера групп" относятся к типу целых чисел, но не являются сравнимыми. Заметим, что в большинстве реляционных СУБД понятие домена не используется, хотя в Oracle V.7 оно уже поддерживается.

Схема отношения - это именованное множество пар {имя атрибута, имя домена (или типа, если понятие домена не поддерживается)}. Степень или "арность" схемы отношения - мощность этого множества. Степень отношения СОТРУДНИКИ равна четырем, то есть оно является 4-арным. Если все атрибуты одного отношения определены на разных доменах, осмысленно использовать для именования атрибутов имена соответствующих доменов (не забывая, конечно, о том, что это является всего лишь удобным способом именования и не устраняет различия между понятиями домена и атрибута).

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). Тем самым, степень или "арность" кортежа, т.е. число элементов в нем, совпадает с "арностью" соответствующей схемы отношения. Попросту говоря, кортеж - это набор именованных значений заданного типа.

Отношение - это множество кортежей, соответствующих одной схеме отношения. Иногда, чтобы не путаться, говорят "отношение-схема" и "отношение-экземпляр", иногда схему отношения называют заголовком отношения, а отношение как набор кортежей - телом отношения. На самом деле, понятие схемы отношения ближе всего к понятию структурного типа данных в языках программирования. Было бы вполне логично разрешать отдельно определять схему отношения, а затем одно или несколько отношений с данной схемой.

Однако в реляционных базах данных это не принято. Имя схемы отношения в таких базах данных всегда совпадает с именем соответствующего отношения-экземпляра. В классических реляционных базах данных после определения схемы базы данных изменяются только отношения-экземпляры. В них могут появляться новые и удаляться или модифицироваться существующие кортежи. Однако во многих реализациях допускается и изменение схемы базы данных: определение новых и изменение существующих схем отношения. Это принято называть эволюцией схемы базы данных.

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". Когда мы перейдем к рассмотрению практических вопросов организации реляционных баз данных и средств управления, мы будем использовать эту житейскую терминологию. Этой терминологии придерживаются в большинстве коммерческих реляционных СУБД.

Реляционная база данных - это набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

Как видно, основные структурные понятия реляционной модели данных (если не считать понятия домена) имеют очень простую интуитивную интерпретацию, хотя в теории реляционных БД все они определяются абсолютно формально и точно.

2. Фундаментальные свойства отношений.

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

Отсутствие кортежей-дубликатов

То свойство, что отношения не содержат 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. Во многих практических реализациях РСУБД допускается нарушение свойства уникальности кортежей для промежуточных отношений, порождаемых неявно при выполнении запросов. Такие отношения являются не множествами, а мультимножествами, что в ряде случаев позволяет добиться определенных преимуществ, но иногда приводит к серьезным проблемам.

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.

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.

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). Принято говорить, что в реляционных базах данных допускаются только нормализованные отношения или отношения, представленные в первой нормальной форме. Потенциальным примером ненормализованного отношения является следующее:

НОМЕР_ОТДЕЛА	ОТДЕЛ		
	СОТР_НОМЕР	СОТР_ИМЯ	СОТР_ЗАРП
310	2934	Иванов	112,000
	2935	Петров	112,500
313	2937	Федоров	110,000
315	2938	Иванова	112,000

Можно сказать, что здесь мы имеем бинарное отношение, значениями атрибута ОТДЕЛЫ которого являются отношения. Заметим, что исходное отношение СОТРУДНИКИ является нормализованным вариантом отношения ОТДЕЛЫ:

СОТР_НОМЕР	СОТР_ИМЯ	СОТР_ЗАРП	СОТР_ОТД_НОМЕР
2934	Иванов	112,000	310
2935	Петров	144,000	310
2936	Сидоров	92,000	313
2937	Федоров	110,000	310
2938	Иванова	112,000	315

Нормализованные отношения составляют основу классического реляционного подхода к организации баз данных. Они обладают некоторыми ограничениями (не любую информацию удобно представлять в виде плоских таблиц), но существенно упрощают манипулирование данными. Рассмотрим, например, два идентичных оператора занесения кортежа:

Зачислить сотрудника Кузнецова (пропуск номер 3000, зарплата 115,000) в отдел номер 320 и

Зачислить сотрудника Кузнецова (пропуск номер 3000, зарплата 115,000) в отдел номер 310.

Если информация о сотрудниках представлена в виде отношения СОТРУДНИКИ, оба оператора будут выполняться одинаково (вставить кортеж в отношение СОТРУДНИКИ). Если же работать с ненормализованным отношением ОТДЕЛЫ, то первый оператор выразится в занесение кортежа, а второй - в добавление информации о Кузнецове в множественное значение атрибута ОТДЕЛ кортежа с первичным ключом 310.

3. Реляционная модель данных.

Когда в предыдущих разделах мы говорили об основных понятиях реляционных баз данных, мы не опирались на какую-либо конкретную реализацию. Эти рассуждения в равной степени относились к любой системе, при построении которой использовался реляционный подход.Другими словами, мы использовали понятия так называемой реляционной модели данных. Модель данных описывает некоторый набор родовых понятий и признаков, которыми должны обладать все конкретные СУБД и управляемые ими базы данных, если они основываются на этой модели. Наличие модели данных позволяет сравнивать конкретные реализации, используя один общий язык. Хотя понятие модели данных является общим, и можно говорить о иерархической, сетевой, некоторой семантической и т.д. моделях данных, нужно отметить, что это понятие было введено в обиход применительно к реляционным системам и наиболее эффективно используется именно в этом контексте. Попытки прямолинейного применения аналогичных моделей к дореляционным организациям показывают, что реляционная модель слишком "велика" для них, а для постреляционных организаций она оказывается "мала".

Наиболее распространенная трактовка реляционной модели данных, по-видимому, принадлежит Дейту, который воспроизводит ее (с различными уточнениями) практически во всех своих книгах. Согласно Дейту реляционная модель состоит из трех частей, описывающих разные аспекты реляционного подхода: структурной части, манипуляционной части и целостной части. В структурной части модели фиксируется, что единственной структурой данных, используемой в реляционных БД, является нормализованное n-арное отношение. По сути дела, в предыдущих двух разделах этой лекции мы рассматривали именно понятия и свойства структурной составляющей реляционной модели. В манипуляционной части модели утверждаются два фундаментальных механизма манипулирования реляционными БД - реляционная алгебра и реляционное исчисление. Первый механизм базируется в основном на классической теории множеств (с некоторыми уточнениями), а второй - на классическом логическом аппарате исчисления предикатов первого порядка. Мы рассмотрим эти механизмы более подробно на следующей лекции, а пока лишь заметим, что основной функцией манипуляционной части реляционной модели является обеспечение меры реляционности любого конкретного языка реляционных БД: язык называется

реляционным, если он обладает не меньшей выразительностью и мощностью, чем реляционная алгебра или реляционное исчисление.

Наконец, в целостной части реляционной модели данных фиксируются два базовых требования целостности, которые должны поддерживаться в любой реляционной СУБД. Первое требование называется требованием целостности сущностей. Объекту или сущности реального мира в реляционных БД соответствуют кортежи отношений. Конкретно требование состоит в том, что любой кортеж любого отношения отличим от любого другого кортежа этого отношения, т.е. другими словами, любое отношение должно обладать первичным ключом. Как мы видели в предыдущем разделе, это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Второе требование называется требованием целостности по ссылкам и является несколько более сложным. Очевидно, что при соблюдении нормализованности отношений сложные сущности реального мира представляются в реляционной БД в виде нескольких кортежей нескольких отношений. Например, представим, что нам требуется представить в реляционной базе данных сущность ОТДЕЛ с атрибутами ОТД_НОМЕР (номер отдела), ОТД_КОЛ (количество сотрудников) и ОТД_СОТР (набор сотрудников отдела). Для каждого сотрудника нужно хранить СОТР_НОМЕР (номер сотрудника), СОТР_ИМЯ (имя сотрудника) и СОТР_ЗАРП (заработная плата сотрудника). Как мы вскоре увидим, при правильном проектировании соответствующей БД в ней появятся два отношения: ОТДЕЛЫ (ОТД_НОМЕР, ОТД_КОЛ) (первичный ключ - ОТД_НОМЕР) и СОТРУДНИКИ (СОТР_НОМЕР, СОТР_ИМЯ, СОТР_ЗАРП, СОТР_ОТД_НОМ) (первичный ключ - СОТР_НОМЕР).

Как видно, атрибут СОТР_ОТД_НОМ появляется в отношении СОТРУДНИКИ не потому, что номер отдела является собственным свойством сотрудника, а лишь для того, чтобы иметь возможность восстановить при необходимости полную сущность ОТДЕЛ. Значение атрибута СОТР_ОТД_НОМ в любом кортеже отношения СОТРУДНИКИ должно соответствовать значению атрибута ОТД_НОМ в некотором кортеже отношения ОТДЕЛЫ. Атрибут такого рода называется внешним ключом, поскольку его значения однозначно характеризуют сущности, представленные кортежами некоторого другого отношения (т.е. задают значения их первичного ключа). Говорят, что отношение, в котором определен внешний ключ, ссылается на соответствующее отношение, в котором такой же атрибут является первичным ключом. Требование целостности по ссылкам, или требование внешнего ключа состоит в том, что для каждого значения внешнего ключа, появляющегося в ссылающемся отношении, в отношении, на которое ведет ссылка, должен найтись кортеж с таким же значением первичного ключа, либо значение внешнего ключа должно быть неопределенным (т.е. ни на что не указывать). Для нашего примера это означает, что если для сотрудника указан номер отдела, то этот отдел должен существовать.

Ограничения целостности сущности и по ссылкам должны поддерживаться СУБД. Для соблюдения целостности сущности достаточно гарантировать отсутствие в любом отношении кортежей с одним и тем же значением первичного ключа. С целостностью по ссылкам дела обстоят несколько более сложно. Понятно, что при обновлении ссылающегося отношения (вставке новых кортежей или модификации значения внешнего ключа в существующих кортежах) достаточно следить за тем, чтобы не появлялись некорректные значения внешнего ключа. Но как быть при удалении кортежа из отношения, на которое ведет ссылка? Здесь существуют три подхода, каждый из которых поддерживает целостность по ссылкам. Первый подход заключается в том, что запрещается производить удаление кортежа, на который существуют ссылки (т.е. сначала нужно либо удалить ссылающиеся кортежи, либо соответствующим образом изменить значения их внешнего ключа). При втором подходе при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным. Наконец, третий подход (каскадное удаление) состоит в том, что при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

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

1.5 Лекция №9, 10 (4 часа).

Тема: «Базисные средства манипулирования реляционными данными»

1.5.1 Вопросы лекции:

1. Реляционная алгебра
2. Обзор реляционной алгебры Кодда
3. Обзор реляционной алгебры А Дейта и Дарвена

1.5.2 Краткое содержание вопросов:

1. Реляционная алгебра.

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

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

-объединения отношений;

-пересечения отношений;

-взятия разности отношений;

-прямого произведения отношений.

Специальные реляционные операции включают:

-ограничение отношения;

-проекцию отношения;

-соединение отношений;

-деление отношений.

Кроме того, в состав алгебры включается операция присваивания, позволяющая сохранить в базе данных результаты вычисления алгебраических выражений, и операция переименования атрибутов, дающая возможность корректно сформировать заголовок (схему) результирующего отношения.

2. Обзор реляционной алгебры Кодда.

При выполнении операции объединения (UNION) двух отношений с одинаковыми заголовками производится отношение, включающее все кортежи, которые входят хотя бы в одно из отношений-операндов.

Операция пересечения (INTERSECT) двух отношений с одинаковыми заголовками производит отношение, включающее все кортежи, которые входят в оба отношения-операнда.

Отношение, являющееся разностью (MINUS) двух отношений с одинаковыми заголовками, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, которое является вторым операндом.

При выполнении декартова произведения (TIMES) двух отношений, пересечение заголовков которых пусто, производится отношение, кортежи которого производятся путем объединения кортежей первого и второго операндов.

Результатом ограничения (WHERE) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.

При выполнении проекции (PROJECT) отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого являются соответствующими подмножествами кортежей отношения-операнда.

При соединении (JOIN) двух отношений по некоторому условию образуется результирующее отношение, кортежи которого производятся путем объединения кортежей первого и второго отношений и удовлетворяют этому условию.

У операции реляционного деления (DIVIDE BY) два операнда – бинарное и унарное отношения. Результирующее отношение состоит из унарных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда.

Операция переименования (RENAME) производит отношение, тело которого совпадает с телом операнда, но имена атрибутов изменены.

Операция присваивания (:=) позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.

Поскольку результатом любой реляционной операции (кроме операции присваивания, которая не вырабатывает значения) является некое отношение, можно образовывать реляционные выражения, в которых вместо отношения-операнда некоторой реляционной операции находится вложенное реляционное выражение. В построении реляционного выражения могут участвовать все реляционные операции, кроме операции присваивания. Вычислительная интерпретация реляционного выражения диктуется установленными приоритетами операций:

RENAME WHERE = PROJECT TIMES = JOIN = INTERSECT = DIVIDE BY UNION = MINUS

В другой форме приоритеты операций показаны на рис. 1. Вычисление выражения производится слева направо с учетом приоритетов операций и скобок.

Операция	Приоритет
RENAME	4
WHERE	3
PROJECT	3
TIMES	2
JOIN	2
INTERSECT	2
DIVIDE BY	2
UNION	1
MINUS	1

Рис. 1. Таблица приоритетов операций традиционной реляционной алгебры

Замкнутость реляционной алгебры и операция переименования

Каждое значение-отношение характеризуется заголовком (или схемой) и телом (или множеством кортежей). Поэтому, если нам действительно нужна алгебра, операции которой замкнуты относительно понятия отношения, то каждая операция должна производить отношение в полном смысле, т. е. оно должно обладать и телом, и заголовком. Только в этом случае можно будет строить вложенные выражения.

Заголовок отношения представляет собой множество пар <имя-атрибута, имя-домена>. Если посмотреть на общий обзор реляционных операций, приведенный в предыдущем подразделе, то видно, что домены атрибутов результирующего отношения однозначно определяются доменами отношений-операндов. Однако с именами атрибутов результата не всегда все так просто.

Например, представим себе, что у отношений-операндов операции декартова произведения имеются одноименные атрибуты с одинаковыми доменами. Каким был бы заголовок результирующего отношения? Поскольку это множество, в нем не должны содержаться одинаковые элементы. Но и потерять атрибут в результате недопустимо. А это значит, что в таком случае вообще невозможно корректно выполнить операцию декартова произведения.

Аналогичные проблемы могут возникать и в случаях других двуместных операций. Для разрешения проблем в число операций реляционной алгебры вводится операция переименования. Ее следует применять в том случае, когда возникает конфликт именования атрибутов в отношениях-операндах одной реляционной операции. Тогда к одному из операндов сначала применяется операция переименования, а затем основная операция выполняется уже без всяких проблем. Более строго мы определим операцию переименования в следующей лекции, а пока лишь заметим, что

результатом этой операции является отношение, совпадающее во всем с отношением-операндом, кроме того, что имя указанного атрибута изменено на заданное имя.

В дальнейшем изложении мы будем предполагать применение операции переименования во всех конфликтных ситуациях. Заметим, кстати, что невозможность применения некоторых операций к произвольным парам значений отношений без предварительного переименования атрибутов отношений операндов означает, что «алгебра» Кодда не является алгеброй отношений в математическом смысле. Описываемая в следующей главе Алгебра А такими недостатками не обладает: результотатом применения любой операции к любым отношениям является некоторое отношение.

Особенности теоретико-множественных операций реляционной алгебры

Хотя в основе теоретико-множественной части реляционной алгебры Кодда лежит классическая теория множеств, соответствующие операции реляционной алгебры обладают некоторыми особенностями.

Операции объединения, пересечения, взятия разности. Совместимость по объединению

Начнем с операции объединения отношений (все, что будет сказано по поводу объединения, верно и для операций пересечения и взятия разности отношений). Смысл операции объединения в реляционной алгебре в целом остается теоретико-множественным. В теории множеств:

-результатом объединения двух множеств A{a} и B{b} является такое множество C{c}, что для каждого c либо существует такой элемент a, принадлежащий множеству A, что c=a, либо существует такой элемент b, принадлежащий множеству B, что c=b;

-пересечением множеств A и B является такое множество C{c}, что для любого c существуют такие элементы a, принадлежащий множеству A, и b, принадлежащий множеству B, что c=a=b;

-разностью множеств A и B является такое множество C{c}, что для любого c существует такой элемент a, принадлежащий множеству A, что c=a, и не существует такой элемент b, принадлежащий B, что c=b.

Но если в теории множеств операция объединения осмысленна для любых двух множеств-операндов, то в случае реляционной алгебры результатом операции объединения должно являться отношение. Если в реляционной алгебре допустить возможность теоретико-множественного объединения двух произвольных отношений (с разными заголовками), то, конечно, результатом операции будет множество, но множество разнотипных кортежей, т. е. не отношение. Если исходить из требования замкнутости реляционной алгебры относительно понятия отношения, то такая операция объединения является бессмысленной.

Эти соображения подводят к понятию совместимости отношений по объединению: два отношения совместимы по объединению в том и только в том случае, когда обладают одинаковыми заголовками. В развернутой форме это означает, что в заголовках обоих отношений содержится один и тот же набор имен атрибутов, и одноименные атрибуты определены на одном и том же домене (эта развернутая формулировка, вообще говоря, является излишней, но она пригодится нам в следующем абзаце).

Если два отношения совместимы по объединению, то при обычном выполнении над ними операций объединения, пересечения и взятия разности результатом операции является отношение с корректно определенным заголовком, совпадающим с заголовком каждого из отношений-операндов. Напомним, что если два отношения «почти» совместимы по объединению, т. е. совместимы во всем, кроме имен атрибутов, то до выполнения операции типа объединения эти отношения можно сделать полностью совместимыми по объединению путем применения операции переименования.

Для иллюстрации операций объединения, пересечения и взятия разности предположим, что в базе данных имеются два отношения СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 с одинаковыми схемами {СЛУ_НОМЕР, СЛУ_ИМЯ, СЛУ_ЗАРП, СЛУ_ОТД_НОМЕР} (имена доменов опущены по причине очевидности). Каждое из отношений содержит данные о служащих,

участвующих в соответствующем проекте. На рис. 2 показано примерное наполнение каждого из двух отношений (некоторые служащие участвуют в обоих проектах).

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2939	Сидоренко	18000.00	313
2940	Федоренко	20000.00	310
2941	Иваненко	22000.00	315

Рис. 2. Примерное наполнение отношений СЛУЖАЩИЕ _В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ _В_ПРОЕКТЕ_2

Тогда выполнение операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 UNION СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 позволит получить информацию обо всех служащих, участвующих в обоих проектах. Выполнение операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 INTERSECT СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 позволит получить данные о служащих, которые одновременно участвуют в двух проектах. Наконец, операция СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 MINUS СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 выработает отношение, содержащее кортежи служащих, которые участвуют только в первом проекте. Результаты этих операций показаны на рис. 3.

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 UNION **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2939	Сидоренко	18000.00	313
2940	Федоренко	20000.00	310
2941	Иваненко	22000.00	315
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 INTERSECT **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 MINUS **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

Рис. 3. Результаты выполнения операций UNION, INTERSECT и MINUS

Заметим, что включение в состав операций реляционной алгебры трех операций объединения, пересечения и взятия разности является, очевидно, избыточным, поскольку, например, операция пересечения выражается через операцию взятия разности14). Тем не менее Кодд в свое

время решил включить все три операции, исходя из интуитивных потребностей далекого от математики потенциального пользователя системы реляционных БД.

Операция расширенного декартова произведения и совместимость отношений относительно этой операции

Другие проблемы связаны с операцией взятия декартова произведения двух отношений. В теории множеств декартово произведение может быть получено для любых двух множеств, и элементами результирующего множества являются пары, составленные из элементов первого и второго множеств. Если говорить более точно, декартовым произведением множеств A{a} и B{b} является такое множество пар C{<c1, 2>}, что для каждого элемента <c1, c2> множества C существуют такой элемент a множества A, что c1=a, и такой элемент b множества B, что c2=b.

Поскольку отношения являются множествами, для любых двух отношений возможно получение прямого произведения. Но результат не будет отношением! Элементами результата будут не кортежи, а пары кортежей.

Поэтому в реляционной алгебре используется специализированная форма операции взятия декартова произведения – расширенное декартово произведение отношений. При взятии расширенного декартова произведения двух отношений элементом результирующего отношения является кортеж, который представляет собой объединение одного кортежа первого отношения и одного кортежа второго отношения.

Приведем более точное определение операции расширенного декартова произведения. Пусть имеются два отношения R1{a1, a2, …, an} и R2{b1, b2, …, bm}. Тогда результатом операции R1 TIMES R2 является отношение R{a1, a2, …, an, b1, b2, …, bm}, тело которого является множеством кортежей вида {ra1, ra2, …, ran, rb1, rb2, …, rbm} таких, что {ra1, ra2, …, ran} входит в тело R1, а {rb1, rb2, …, rbm} входит в тело R2.

Но теперь возникает вторая проблема – как получить корректно сформированный заголовок отношения-результата? Поскольку схема результирующего отношения является объединением схем отношений-операндов, то очевидной проблемой может быть именование атрибутов результирующего отношения, если отношения-операнды обладают одноименными атрибутами.

Эти соображения приводят к введению понятия совместимости по взятию расширенного декартова произведения. Два отношения совместимы по взятию расширенного декартова произведения в том и только в том случае, если пересечение множеств имен атрибутов, взятых из их схем отношений, пусто. Любые два отношения всегда могут стать совместимыми по взятию декартова произведения, если применить операцию переименования к одному из этих отношений.

Для наглядности предположим, что в придачу к введенным ранее отношениям СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 в базе данных содержится еще и отношение ПРОЕКТЫ со схемой {ПРОЕКТ_НАЗВ, ПРОЕКТ_РУК} (имена доменов снова опущены) и телом, показанным на рис. 4. На этом же рисунке показан результат операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 TIMES ПРОЕКТЫ.

ПРОЕКТЫ

ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК
ПРОЕКТ 1	Иванов
ПРОЕКТ 2	Иваненко

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 TIMES **ПРОЕКТЫ**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР	ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК
2934	Иванов	22000.00	310	ПРОЕКТ 1	Иванов
2935	Петров	30000.00	310	ПРОЕКТ 1	Иванов
2936	Сидоров	18000.00	313	ПРОЕКТ 1	Иванов
2937	Федоров	20000.00	310	ПРОЕКТ 1	Иванов
2938	Иванова	22000.00	315	ПРОЕКТ 1	Иванов
2934	Иванов	22000.00	310	ПРОЕКТ 2	Иваненко
2935	Петров	30000.00	310	ПРОЕКТ 2	Иваненко
2936	Сидоров	18000.00	313	ПРОЕКТ 2	Иваненко
2937	Федоров	20000.00	310	ПРОЕКТ 2	Иваненко
2938	Иванова	22000.00	315	ПРОЕКТ 2	Иваненко

Рис. 4. Отношение ПРОЕКТЫ и результат операции СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 TIMES ПРОЕКТЫ

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

По поводу теоретико-множественных операций реляционной алгебры следует еще заметить, что все четыре операции являются ассоциативными. Т. е. если обозначить через OP любую из четырех операций, то (A OP B) OP C = A OP (B OP C), и, следовательно, без внесения двусмысленности можно писать A OP B OP C (A, B и C – отношения, обладающие свойствами, необходимыми для корректного выполнения соответствующей операции). Все операции, кроме взятия разности, являются коммутативными, т. е. A OP B = B OP A.

Специальные реляционные операции

Операция ограничения

Операция ограничения требует наличия двух операндов: ограничиваемого отношения и простого условия ограничения. Простое условие ограничения может иметь либо вид (a comp-op b), где a и b - имена атрибутов ограничиваемого отношения, для которых осмысленна операция сравнения comp-op, либо вид (a comp-op const), где a - имя атрибута ограничиваемого отношения, a const - литерально заданная константа.

В результате выполнения операции ограничения производится отношение, заголовок которого совпадает с заголовком отношения-операнда, а в тело входят те кортежи отношения-операнда, для которых значением условия ограничения является true.

Пусть UNION обозначает операцию объединения, INTERSECT - операцию пересечения, а MINUS - операцию взятия разности. Для обозначения операции ограничения будем использовать конструкцию A WHERE comp, где A - ограничиваемое отношение, а comp - простое условие сравнения. Пусть comp1 и comp2 - два простых условия ограничения. Тогда по определению:

A WHERE comp1 AND comp2 обозначает то же самое, что и (A WHERE comp1) INTERSECT (A WHERE comp2)

A WHERE comp1 OR comp2 обозначает то же самое, что и (A WHERE comp1) UNION (A WHERE comp2)

A WHERE NOT comp1 обозначает то же самое, что и A MINUS (A WHERE comp1)

С использованием этих определений можно использовать операции ограничения, в которых условием ограничения является произвольное булевское выражение, составленное из простых условий с использованием логических связок AND, OR, NOT и скобок.

На интуитивном уровне операцию ограничения лучше всего представлять как взятие некоторой "горизонтальной" вырезки из отношения-операнда.

Операция взятия проекции

Операция взятия проекции также требует наличия двух операндов - проецируемого отношения A и списка имен атрибутов, входящих в заголовок отношения A.

Результатом проекции отношения A по списку атрибутов a1, a2, ..., an является отношение, с заголовком, определяемым множеством атрибутов a1, a2, ..., an, и с телом, состоящим из кортежей вида <a1:v1, a2:v2, ..., an:vn> таких, что в отношении A имеется кортеж, атрибут a1 которого имеет значение v1, атрибут a2 имеет значение v2, ..., атрибут an имеет значение vn. Тем самым, при выполнении операции проекции выделяется "вертикальная" вырезка отношения-операнда с естественным уничтожением потенциально возникающих кортежей-дубликатов.

Операция соединения отношений

Общая операция соединения (называемая также соединением по условию) требует наличия двух операндов - соединяемых отношений и третьего операнда - простого условия. Пусть соединяются отношения A и B. Как и в случае операции ограничения, условие соединения comp имеет вид либо (a comp-op b), либо (a comp-op const), где a и b - имена атрибутов отношений A и B, const - литерально заданная константа, а comp-op - допустимая в данном контексте операция сравнения.

Тогда по определению результатом операции сравнения является отношение, получаемое путем выполнения операции ограничения по условию comp прямого произведения отношений A и B.

Если внимательно осмыслить это определение, то станет ясно, что в общем случае применение условия соединения существенно уменьшит мощность результата промежуточного прямого произведения отношений-операндов только в том случае, когда условие соединения имеет вид (a comp-op b), где a и b - имена атрибутов разных отношений-операндов. Поэтому на практике обычно считают реальными операциями соединения именно те операции, которые основываются на условии соединения приведенного вида.

Хотя операция соединение в нашей интерпретации не является примитивной (поскольку она определяется с использованием прямого произведения и проекции), в силу особой практической важности она включается в базовый набор операций реляционной алгебры. Заметим также, что в практических реализациях соединение обычно не выполняется именно как ограничение прямого произведения. Имеются более эффективные алгоритмы, гарантирующие получение такого же результата.

Имеется важный частный случай соединения - эквисоединение и простое, но важное расширение операции эквисоединения - естественное соединение. Операция соединения называется операцией эквисоединения, если условие соединения имеет вид (a = b), где a и b - атрибуты разных операндов соединения. Этот случай важен потому, что (a) он часто встречается на практике, и (b) для него существуют эффективные алгоритмы реализации.

Операция естественного соединения применяется к паре отношений A и B, обладающих (возможно составным) общим атрибутом c (т.е. атрибутом с одним и тем же именем и определенным на одном и том же домене). Пусть ab обозначает объединение заголовков отношений A и B. Тогда естественное соединение A и B - это спроектированный на ab результат эквисоединения A и B по A/c и BBC. Если вспомнить введенное нами в конце предыдущей главы определение внешнего ключа отношения, то должно стать понятно, что основной смысл операции естественного соединения - возможность

восстановления сложной сущности, декомпозированной по причине требования первой нормальной формы. Операция естественного соединения не включается прямо в состав набора операций реляционной алгебры, но она имеет очень важное практическое значение.

Операция деления отношений

Эта операция наименее очевидна из всех операций реляционной алгебры и поэтому нуждается в более подробном объяснении. Пусть заданы два отношения - A с заголовком {a1, a2, ..., an, b1, b2, ..., bm} и B с заголовком {b1, b2, ..., bm}. Будем считать, что атрибут bi отношения A и атрибут bi отношения B не только обладают одним и тем же именем, но и определены на одном и том же домене. Назовем множество атрибутов {aj} составным атрибутом a, а множество атрибутов {bj} - составным атрибутом b. После этого будем говорить о реляционном делении бинарного отношения A(a,b) на унарное отношение B(b).

Результатом деления A на B является унарное отношение C(a), состоящее из кортежей v таких, что в отношении A имеются кортежи <v, w> такие, что множество значений {w} включает множество значений атрибута b в отношении B.

Предположим, что в базе данных сотрудников поддерживаются два отношения: СОТРУДНИКИ (ИМЯ, ОТД_НОМЕР) и ИМЕНА (ИМЯ), причем унарное отношение ИМЕНА содержит все фамилии, которыми обладают сотрудники организации. Тогда после выполнения операции реляционного деления отношения СОТРУДНИКИ на отношение ИМЕНА будет получено унарное отношение, содержащее номера отделов, сотрудники которых обладают всеми возможными в этой организации именами.

3. Обзор реляционной алгебры А Дейта и Дарвена.

Базисом предложенной Крисом Дейтом и Хью Дарвеном Алгебры А являются операции реляционного отрицания (дополнения), реляционной конъюнкции (или дизъюнкции) и проекции (удаления атрибута). Реляционные аналоги логических операций определяются в терминах отношений на основе обычных теоретико-множественных операций и позволяют выражать напрямую операции пересечения, декартова произведения, естественного соединения, объединения отношений и т. д. Путем комбинирования базовых операций выражаются операции переименования атрибутов, соединения общего вида, взятия разности отношений. Алгебра А позволяет лучше осознать логические основы реляционной модели, хотя, безусловно, является в меньшей степени ориентированной на практическое применение, чем алгебра Кодда16). Даже сами авторы Алгебры А, Дейт и Дарвен, в своем учебном языке Tutorial D используют не Алгебру А напрямую, а некоторое ее надмножество, в большей степени напоминающее алгебру Кодда.

Базовые операции Алгебры А

Пусть r – отношение, A – имя атрибута отношения r, T – имя соответствующего типа (т. е. типа или домена атрибута A), v – значение типа T. Тогда:

-заголовком Hr отношения r называется множество атрибутов, т.е. упорядоченных пар вида <A, T>. По определению никакие два атрибута в этом множестве не могут содержать одно и то же имя атрибута A;

-кортеж tr, соответствующий заголовку Hr, – это множество упорядоченных триплетов вида <A, T, v>, по одному такому триплету для каждого атрибута в Hr;

-тело Br отношения r – это множество кортежей tr. Заметим, что (в общем случае) могут существовать такие кортежи tr, которые соответствуют Hr, но не входят в Br.

Заметим, что заголовок – это множество (упорядоченных пар вида <A, T>), тело – это множество (кортежей tr), и кортеж – это множество (упорядоченных триплетов вида <A, T, v>). Элемент заголовка – это атрибут (т. е. упорядоченная пара вида <A,T>); элемент тела – это кортеж; элемент кортежа – это упорядоченный триплет вида <A, T, v>. Любое подмножество заголовка – это заголовок, любое подмножество тела – это тело, и любое подмножество кортежа – это кортеж.

Определим несколько основных операций (как будет показано далее, некоторые из них избыточны). Каждое из последующих определений состоит из: формальной спецификации ограничений (если они имеются), применимых к операндам соответствующей

операции; формальной спецификации заголовка результата этой операции; формальной спецификации тела этого результата и неформального обсуждения формальных спецификаций.

Во всех формальных спецификациях exists обозначает квантор существования; exists tr означает «существует такой tr, что». Символ «» означает принадлежность одного множества другому; trBr означает, что элемент tr принадлежит множеству Br. Выражение trBr означает, что элемент tr не принадлежит множеству Br. Операции minus и union являются традиционными теоретико-множественными операциями взятия разности и объединения множеств.

Поскольку некоторые базовые операции Алгебры А имеют названия обычных логических операций, чтобы избежать путаницы, имена реляционных операций берутся в угловые скобки: <NOT>, <AND>, <OR> и т. д. В исходный базовый набор операций входят операции реляционного дополнения <NOT>, удаления атрибута <REMOVE>, переименования атрибута <RENAME>, реляционной конъюнкции <AND> и реляционной дизъюнкции <OR>.

Операция реляционного дополнения

Пусть s обозначает результат операции <NOT> r. Тогда:

Hs = Hr (заголовок результата совпадает с заголовком операнда);

Bs = {ts : exists tr (tr Br and ts = tr) } (в тело результата входят все кортежи, соответствующие заголовку и не входящие в тело операнда).

Операция <NOT> производит дополнение s заданного отношения r. Заголовком s является заголовок r. Тело s включает все кортежи, соответствующие этому заголовку и не входящие в тело r.

Во-первых, термин «дополнение» полностью соответствует сути операции <NOT>: тело результата операции <NOT> r является дополнением Br до полного множества кортежей, соответствующих Hr. Во-вторых, это не противоречит природе булевской операции NOT: у булевского типа имеются всего два значения – true и false, и NOT true = false, a NOT false = true.

Чтобы привести пример использования операции <NOT>, предположим, что в состав домена ДОПУСТИМЫЕ_НОМЕРА_ПРОЕКТОВ, на котором определен атрибут ПРО_НОМ отношения НОМЕРА_ПРОЕКТОВ с рис. 5 слева, входит всего пять значений {1, 2, 3, 4, 5}. Тогда результат операции <NOT> НОМЕРА_ПРОЕКТОВ будет таким, как показано на рис. 5 справа.

Рис. 5. Результат операции <NOT> НОМЕРА_ПРОЕКТОВ

Операция удаления атрибута

Пусть s обозначает результат операции r <REMOVE> A. Для обеспечения возможности выполнения операции требуется, чтобы существовал некоторый тип (или домен) T такой, что <A, T> Hr (т. е. в состав заголовка отношения r должен входить атрибут A). Тогда:

-Hs = Hr minus {<A, T>}, т. е. заголовок результата получается из заголовка операнда изъятием атрибута A;

-Bs = {ts : exists tr exists v (tr Br and v T and <A,T,v> tr and ts = tr minus {<A,T,v>})}, т. е. в тело результата входят все кортежи операнда, из которых удалено значение атрибута A.

Операция <REMOVE> производит отношение s, формируемое путем удаления указанного атрибута A из заданного отношения r. Операция эквивалентна взятию проекции r

на все атрибуты, кроме A. Заголовок s получается теоретико-множественным вычитанием из заголовка r множества из одного элемента {<A, T>}. Тело s состоит из таких кортежей, которые соответствуют заголовку s, причем каждый из них является подмножеством некоторого кортежа тела отношения r.

Примером операции REMOVE (конечно же, очень похожим на пример использования операции PROJECT из предыдущей лекции) является СЛУЖАЩИЕ REMOVE ПРО_НОМ (получить данные о служащих, участвующих в проектах). Результат этой операции над отношением СЛУЖАЩИЕ, тело которого приведено в верхней части рис. 6, показан на рис. 5.2 внизу.

СЛУЖАЩИЕ

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
2934	Иванов	22400.00	1
2935	Петров	29600.00	1
2936	Сидоров	18000.00	1
2937	Федоров	20000.00	1
2938	Иванова	22000.00	1
2934	Иванов	22400.00	2
2935	Петров	29600.00	2
2939	Сидоренко	18000.00	2
2940	Федоренко	20000.00	2
2941	Иваненко	22000.00	2

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП
2934	Иванов	22400.00
2935	Петров	29600.00
2936	Сидоров	18000.00
2937	Федоров	20000.00
2938	Иванова	22000.00
2939	Сидоренко	18000.00
2940	Федоренко	20000.00
2941	Иваненко	22000.00

Рис. 6. Результат операции СЛУЖАЩИЕ REMOVE ПРО_НОМ

Операция переименования

Пусть s обозначает результат операции r <RENAME> (A, B). Для обеспечения возможности выполнения операции требуется, чтобы существовал некоторый тип T, такой, что <A, T> Hr, и чтобы не существовал такой тип T, что <B, T> Hr. (Другими словами, в схеме отношения r должен присутствовать атрибут A и не должен присутствовать атрибут B.) Тогда:

-Hs = (Hr minus {<A, T>}) union {<B, T>}, т. е. в схеме результата B заменяет A;

-Bs = {ts : exists tr exists v (tr Br and v T and <A, T, v> tr and ts = (tr minus {<A, T, v>}) union {<B, T, v>})}, т. е. в кортежах тела результата имя значений атрибута A меняется на B.

Операция <RENAME> производит отношение s, которое отличается от заданного отношения r только именем одного его атрибута, которое изменяется с A на B. Заголовок s такой же, как заголовок r, за исключением того, что пара <B, T> заменяет пару <A, T>. Тело s включает все кортежи тела r, но в каждом из этих кортежей триплет <B, T, v> заменяет триплет <A, T, v>.

Операция реляционной конъюнкции

Пусть s обозначает результат операции r1 <AND> r2. Для обеспечения возможности выполнения операции требуется, чтобы если <A, T1>Hr1 и <A, T2>Hr2, то T1=T2. (Другими словами, если в двух отношениях-операндах имеются одноименные атрибуты, то они должны быть определены на одном и том же типе (домене).) Тогда:

-Hs = Hr1 union Hr2, т. е. заголовок результата получается путем объединения заголовков отношений-операндов, как в операциях TIMES и JOIN из предыдущей лекции;

-Bs = { ts : exists tr1 exists tr2 ((tr1Br1 and tr2Br2) and ts = tr1 union tr2)}; обратите внимание на то, что кортеж результата определяется как объединение кортежей операндов; поэтому:

-если схемы отношений-операндов имеют непустое пересечение, то операция <AND> работает как естественное соединение;

-если пересечение схем операндов пусто, то <AND> работает как расширенное декартово произведение;

-если схемы отношений полностью совпадают, то результатом операции является пересечение двух отношений-операндов.

Операция <AND> является реляционной конъюнкцией, в некоторых случаях выдающей в результате отношение rs, ранее называвшееся естественным соединением двух заданных отношений r1 и r2. Заголовок rs является объединением заголовков r1 и r2. Тело s включает каждый кортеж, соответствующий заголовку s и являющийся надмножеством некоторого кортежа из тела r1 и некоторого кортежа из тела r2.

Для иллюстрации воспользуемся примерными отношениями, показанными на рис. 7.

СЛУЖАЩИЕ_В_ПРОЕКТЕ_1

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2936	Сидоров	18000.00	313
2937	Федоров	20000.00	310
2938	Иванова	22000.00	315

СЛУЖАЩИЕ_В_ПРОЕКТЕ_2

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310
2939	Сидоренко	18000.00	313
2940	Федоренко	20000.00	310
2941	Иваненко	22000.00	315

ПРОЕКТЫ

ПРО_НОМ	ПРОЕКТ_РУК
1	Иванов
2	Иваненко

СЛУЖАЩИЕ

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ
2934	Иванов	22400.00	1
2935	Петров	29600.00	1
2936	Сидоров	18000.00	1
2937	Федоров	20000.00	1
2938	Иванова	22000.00	1
2934	Иванов	22400.00	2
2935	Петров	29600.00	2
2939	Сидоренко	18000.00	2
2940	Федоренко	20000.00	2
2941	Иваненко	22000.00	2

Рис. 7. Примерные отношения для иллюстрации операции <AND>

На рис. 8(a) у отношений СЛУЖАЩИЕ и ПРОЕКТЫ имеется общий атрибут ПРО_НОМ. Поэтому операция <AND> работает как операция естественного соединения. На рис. 8(b) пересечение заголовков отношений СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 и ПРОЕКТЫ пусто, и поэтому в результате реляционной конъюнкции производится расширенное декартово произведение этих отношений. Наконец, на рис. 8(c) схемы отношений СЛУЖАЩИЕ_В_ПРОЕКТЕ_1 и СЛУЖАЩИЕ_В_ПРОЕКТЕ_2 совпадают, и телом операции <AND> является пересечение тел отношений-операндов.

(a) Результат операции **СЛУЖАЩИЕ** <AND> **ПРОЕКТЫ**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	ПРО_НОМ	ПРОЕКТ_РУК
2934	Иванов	22400.00	1	Иванов
2935	Петров	29600.00	1	Иванов
2936	Сидоров	18000.00	1	Иванов
2937	Федоров	20000.00	1	Иванов
2938	Иванова	22000.00	1	Иванов
2934	Иванов	22400.00	2	Иваненко
2935	Петров	29600.00	2	Иваненко
2939	Сидоренко	18000.00	2	Иваненко
2940	Федоренко	20000.00	2	Иваненко
2941	Иваненко	22000.00	2	Иваненко

(b) Результат операции **СЛУЖАЩИЕ_В_ПРОЕКТЕ_1** <AND> **ПРОЕКТЫ**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР	ПРО_НОМ	ПРОЕКТ_РУК
2934	Иванов	22000.00	310	1	Иванов
2935	Петров	30000.00	310	1	Иванов
2936	Сидоров	18000.00	313	1	Иванов
2937	Федоров	20000.00	310	1	Иванов
2938	Иванова	22000.00	315	1	Иванов
2934	Иванов	22000.00	310	2	Иваненко
2935	Петров	30000.00	310	2	Иваненко
2936	Сидоров	18000.00	313	2	Иваненко
2937	Федоров	20000.00	310	2	Иваненко
2938	Иванова	22000.00	315	2	Иваненко

(c) Результат операции **СЛУЖАЩИЕ_В_ПРОЕКТЕ_1** <AND> **СЛУЖАЩИЕ_В_ПРОЕКТЕ_2**

СЛУ_НОМЕР	СЛУ_ИМЯ	СЛУ_ЗАРП	СЛУ_ОТД_НОМЕР
2934	Иванов	22000.00	310
2935	Петров	30000.00	310

Рис. 8. Иллюстрации операции реляционной конъюнкции

Операция реляционной дизъюнкции

Пусть s обозначает результат операции r1 <OR> r2. Для обеспечения возможности выполнения операции требуется, чтобы если <A, T1>Hr1 и <A, T2>Hr2, то должно быть T1 = T2 (одноименные атрибуты должны быть определены на одном и том же типе). Тогда:

-Hs = Hr1 union Hr2 (из схемы результата удаляются атрибуты-дубликаты);

-Bs = { ts : exists tr1 exists tr2 ((tr1Br1 or tr2Br2) and ts = tr1 union tr2)}; очевидно, что при этом:

-если у операндов нет общих атрибутов, то в тело результирующего отношения входят все такие кортежи ts, которые являются объединением кортежей tr1 и tr2, соответствующих заголовкам отношений-операндов, и хотя бы один из этих кортежей принадлежит телу одного из операндов;

-если у операндов имеются общие атрибуты, то в тело результирующего отношения входят все такие кортежи ts, которые являются объединением кортежей tr1 и tr2, соответствующих заголовкам отношений-операндов, если хотя бы один из этих кортежей принадлежит телу одного из операндов, и значения общих атрибутов tr1 и tr2 совпадают;

-если же схемы отношений-операндов совпадают, то тело отношения-результата является объединением тел операндов.

Операция <OR> является реляционной дизъюнкцией и обобщением того, что ранее называлось объединением. Заголовок s есть объединение заголовков r1 и r2. Тело s состоит из всех кортежей, соответствующих заголовку s и являющихся надмножеством либо некоторого кортежа из тела r1, либо некоторого кортежа из тела r2.

Предположим, у нас имеются отношения ПРОЕКТЫ_1 {ПРОЕКТ_НАЗВ, ПРОЕКТ_РУК} и НОМЕРА_ПРОЕКТОВ {ПРО_НОМ} (рис. 5.5). Предположим также, что

домен атрибута ПРОЕКТ_НАЗВ включает значения ПРОЕКТ_1, ПРОЕКТ_2, ПРОЕКТ_3, домен атрибута ПРОЕКТ_РУК ограничен значениями Иванов, Иваненко, а доменом атрибута ПРО_НОМ является множество {1, 2, 3}. Результат операции ПРОЕКТЫ <OR> НОМЕРА_ПРОЕКТОВ показан на рис. 9.

Как показано на рис. 9, операция <OR> при наличии операндов с несовпадающими схемами производит результат, гораздо более мощный, чем результат операции взятия расширенного декартова произведения из лекции 4, и еще менее осмысленный с практической точки зрения.

Для иллюстрации операции <OR> над операндами, схемы которых имеют непустое пересечение, воспользуемся отношением ПРОЕКТЫ_2 {ПРО_НОМ, ПРОЕКТ_РУК} (рис. 10) и унарным отношением НОМЕРА_ПРОЕКТОВ, схема и тело которого показаны на рис. 9. Будем предполагать, что множества значений доменов атрибутов такие же, как в предыдущем примере. Результат операции ПРОЕКТЫ_2 <OR> НОМЕРА_ПРОЕКТОВ показан на рис. 10.

Как уже отмечалось, при совпадении схем отношений-операндов результатом выполнения над ними операции <OR> является объединение отношений. Это непосредственно следует из спецификации операции. Если этот факт кажется неочевидным, еще раз внимательно посмотрите на спецификацию. Иллюстрирующий пример мы приводить не будем.

ПРОЕКТЫ_1

ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК
ПРОЕКТ 1	Иванов
ПРОЕКТ 2	Иваненко

НОМЕРА_ПРОЕКТОВ

ПРО_НОМ
1
2

Результат операции **ПРОЕКТЫ** <OR> **НОМЕРА_ПРОЕКТОВ**

ПРОЕКТ_НАЗВ	ПРОЕКТ_РУК	ПРО_НОМ
ПРОЕКТ 1	Иванов	1
ПРОЕКТ 2	Иванов	1
ПРОЕКТ 3	Иванов	1
ПРОЕКТ 1	Иваненко	1
ПРОЕКТ 2	Иваненко	1
ПРОЕКТ 3	Иваненко	1
ПРОЕКТ 1	Иванов	2
ПРОЕКТ 2	Иванов	2
ПРОЕКТ 3	Иванов	2
ПРОЕКТ 1	Иваненко	2
ПРОЕКТ 2	Иваненко	2
ПРОЕКТ 3	Иваненко	2
ПРОЕКТ 1	Иванов	3
ПРОЕКТ 2	Иваненко	3

Рис. 9. Результат операции <OR> над операндами без общих атрибутов

ПРОЕКТЫ_2

ПРО_НОМ	ПРОЕКТ_РУК
1	Иванов
2	Иваненко

ПРОЕКТЫ_2 <OR> **НОМЕРА_ПРОЕКТОВ**

ПРО_НОМ	ПРОЕКТ_РУК
1	Иванов
2	Иваненко
2	Иванов
1	Иваненко

Рис. 10. Результат операции <OR> над операндами, схемы которых частично пересекаются

1.6 Лекция № 11, 12, 13 (6 часов).

Тема: «Системы управления базами данных»

1.6.1 Вопросы лекции:

1. Общие свойства СУБД
2. Обобщенная схема обмена данных с использованием СУБД
3. Типовые информационные процедуры, реализуемые СУБД
4. Общие сведения о СУБД

1.6.2 Краткое содержание вопросов:

1. Общие свойства СУБД.

БД — это единое хранилище данных, которое однократно определяется, а после этого многократно используется разными пользователями для удовлетворения возникающих многообразных потребностей в информации.

ИС может быть

- одно- или
- многопользовательской.

Данные могут быть организованы в одну или несколько БД, данные в БД могут быть интегрированными (объединение нескольких отдельных файлов данных, полностью или частично не перекрывающихся) и общими (использование отдельных областей данных несколькими пользователями для разных целей).

Информация должна быть организована так, чтобы обеспечить минимальную избыточность. БД содержит информацию о данных, принятую называть «метаданными». В совокупности описания всех данных образуют словарь данных, обеспечивающий независимость данных от приложений. Данные должны быть логически связаны, для чего определяют объекты (то, о чем необходимо хранить информацию) и их свойства (простые и сложные), а затем выявляют связи между ними (то, что объединяет два или более объектов, они также являются частью данных и хранятся в БД).

Основные свойства БД:

- Целостность (В каждый момент времени существования БД сведения, содержащиеся в ней, должны быть непротиворечивы),
- Восстанавливаемость (Данное свойство предполагает возможность восстановления БД после сбоя системы или отдельных видов порчи системы),
- Безопасность (предполагает защиту данных от преднамеренного и непреднамеренного доступа, модификации или разрушения),
- Эффективность (сочетание минимального времени реакции на запрос пользователя и минимальной потребности в памяти),
- Предельные размеры и эксплуатационные ограничения.

К основным свойствам СУБД и базы данных можно отнести:

- отсутствие дублирования данных в различных объектах модели, обеспéчивающее однократный ввод данных и простоту их корректировки;
- непротиворечивость данных;
- целостность БД;
- возможность многоаспéктного доступа;
- всевозможные выборки данных и их использование различными задачами и приложениями пользователя;
- защита и восстановление данных при аварийных ситуациях, аппаратных и программных сбоях, ошибках пользователя;
- защита данных от несанкционированного доступа средствами разграничения доступа для различных пользователей;
- возможность модификации структуры базы данных без повторной загрузки данных;
- обеспéчение независимости программ от данных, позволяющей сохранить программы при модификации структуры базы данных;

- реорганизация размещения данных базы на машинном носителе для улучшения объемно-временных характеристик БД;

- наличие языка запросов высокого уровня, ориентированного на конечного пользователя, который обеспечивает вывод информации из базы данных по любому запросу и предоставление ее в виде соответствующих отчетных форм, удобных для пользователя.

СУБД является основой создания практических приложений пользователя для различных предметных областей.

Критерии выбора СУБД пользователем. Выбор СУБД для практических приложений пользователем определяется многими факторами, к которым относятся:

- имеющееся техническое и базовое программное обеспечение, их конфигурация, оперативная и дисковая память;

- потребности разрабатываемых приложений пользователя;

- тип поддерживаемой модели данных, специфика предметной области, топология информационно-логической модели;

- требования к производительности при обработке данных;

- наличие в СУБД необходимых функциональных средств;

- наличие русифицированной версии СУБД;

- уровень квалификации пользователей и наличие в СУБД диалоговых средств разработки и работ с БД.

СУБД — это программное обеспечение, с помощью которого пользователи могут определять, создавать и поддерживать базу данных, а также осуществлять к ней контролируемый доступ.

Различаются два класса СУБД — системы общего назначения (не ориентированы на какую-либо конкретную предметную область, обладают средствами настройки на работу с конкретной БД в условиях конкретного применения, реализуются как программный продукт, способный функционировать на некоторой модели ЭВМ в определенной операционной обстановке) и специализированные системы (весьма трудоемкий процесс, к которому прибегают, в исключительных ситуациях).

В процессе реализации своих функций СУБД постоянно взаимодействует с базой данных и с другими прикладными программными продуктами пользователя, предназначенными для работы с данной БД и называемыми приложениями.

СУБД включает язык определения данных, с помощью которого можно определить базу данных, ее структуру, типы данных, а также средства задания ограничений для хранимой информации.

СУБД позволяет вставлять, удалять, обновлять и извлекать информацию из базы данных посредством языка управления данными (язык запросов).

Большинство СУБД могут работать на компьютерах с разной архитектурой и под разными операционными системами.

Многопользовательские СУБД имеют достаточно развитые средства администрирования БД.

СУБД предоставляет контролируемый доступ к базе данных с помощью системы обеспечения безопасности, системы поддержки целостности базы данных, системы управления параллельной работой приложений, системы восстановления, доступного пользователям каталога, содержащего описание хранимой в базе данных информации.

Компоненты СУБД - это данные (наиболее важный компонентом среды СУБД, ради общения с ними на должном уровне требуется наличие остальных компонентов, они включают в себя имена, типы и размеры элементов данных, имена связей, ограничения целостности данных, имена зарегистрированных пользователей и их права по доступу к данным, используемые индексы и структуры хранения), пользователи, аппаратное обеспечение (это набор физических устройств, на которых существуют база данных, СУБД и другие компоненты ИС, оно должно соответствовать требованиям использующей его организации, СУБД, БД. Это может быть один персональный компьютер или сеть),

программное обеспечение (ОС, программное обеспечение самой СУБД, прикладные программы), процедуры.

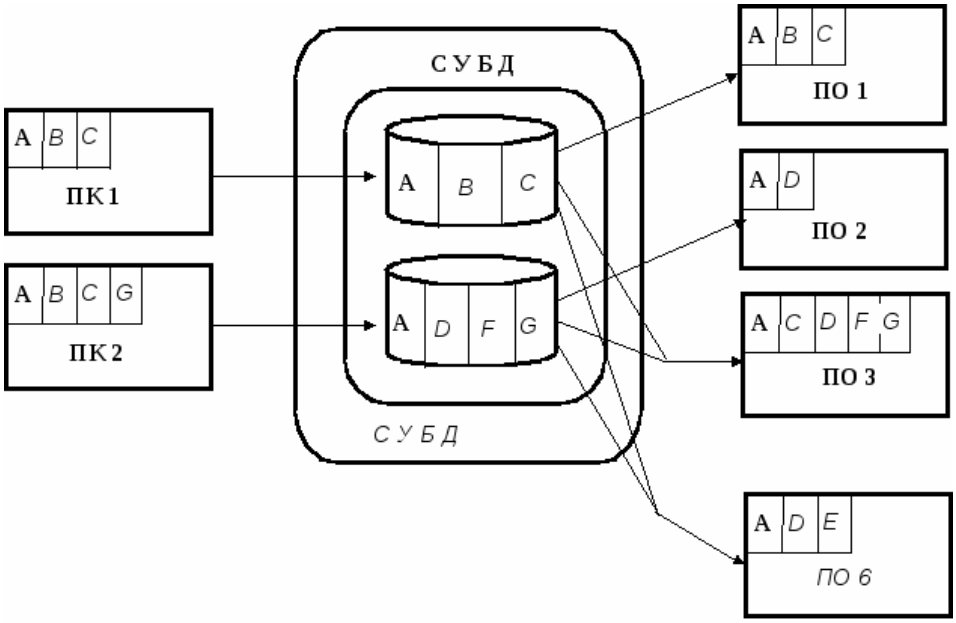
Среди пользователей БД можно выделить четыре категории лиц: администраторы данных (отвечают за концептуальное и логическое проектирование базы данных, управление данными, разработку и сопровождение стандартов, бизнес-правил и деловых процедур) и баз данных (отвечают за физическое проектирование и физическую реализацию базы данных, обеспечение безопасности и целостности данных, обеспечение максимальной производительности приложений), разработчики баз данных (группа пользователей, которая функционирует во время проектирования, создания и реорганизации базы данных и результатом деятельности которой является хорошо спроектированная БД, они делятся на разработчиков логической и физической БД, первые занимаются выявлением объектов, их свойств и связей между ними, вторые – разбираются в функциональных возможностях СУБД), прикладные программисты (разрабатывают приложения, предоставляющие пользователям необходимые им функциональные возможности), конечные пользователи (те, для кого проектируется, создается и поддерживается БД).

2. Обобщенная схема обмена данных с использованием СУБД.

СУБД — специализированные программные средства, предназначенные для организации и ведения баз данных.

СУБД используется для обеспечения эффективного доступа к базе данных со стороны программ (предоставление только необходимой информации, обеспечение независимости от возможных изменение в структуре той части базы данных, которую не обрабатывает программа), по существу она исполняет функции операционной системы по управлению данными. Схема иллюстрирует это положение.

В зависимости от способа взаимодействия СУБД и программы обработки либо в программу передается очередная запись требуемой структуры, не зависимо от того, где физически в БД расположены данные, составляющие требуемую структуру, либо для программы создается временный файл требуемой структуры.



1. СУБД включает язык определения данных, с помощью которого можно определить базу данных, ее структуру, типы данных, а также средства задания ограничений для хранимой информации.

2. СУБД позволяет вставлять, удалять, обновлять и извлекать информацию из базы данных посредством языка управления данными.

3. Большинство СУБД могут работать на компьютерах с разной архитектурой и под разными операционными системами, причем на работу пользователя тип платформы влияния не оказывает.

4. Многопользовательские СУБД имеют достаточно развитые средства администрирования БД.

5. СУБД предоставляет контролируемый доступ к базе данных с помощью:

системы обеспечения безопасности;

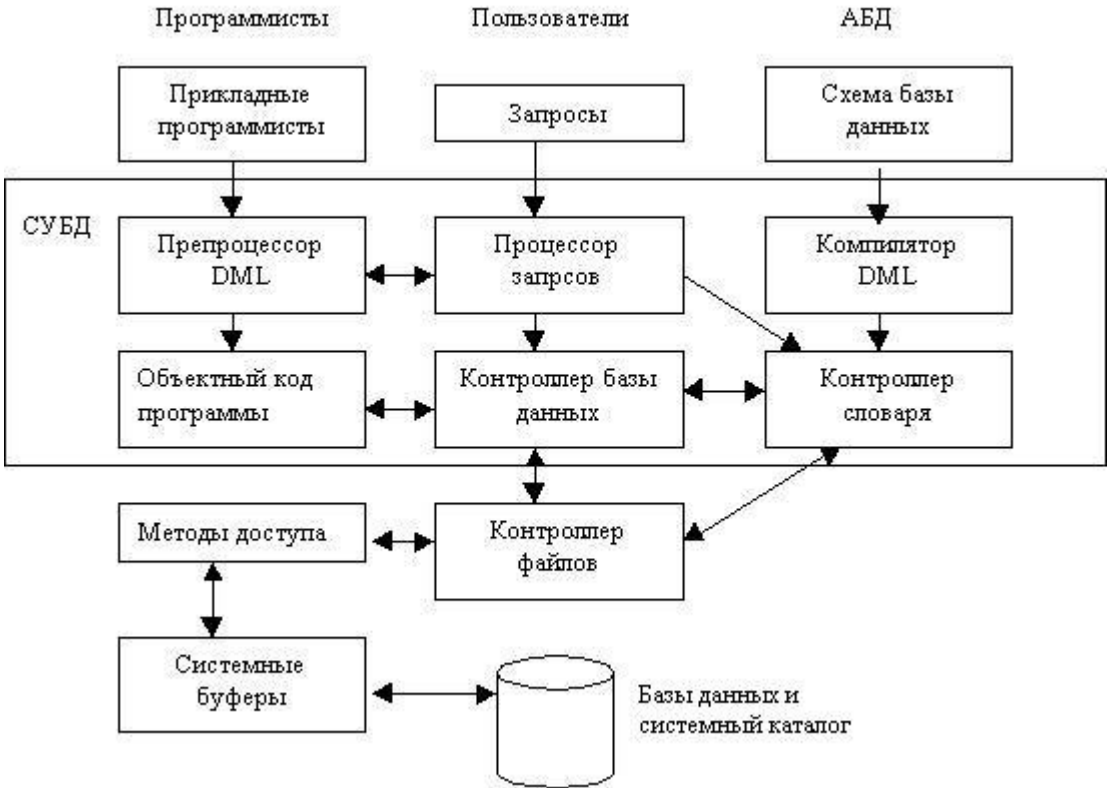
системы поддержки целостности базы данных;

системы управления параллельной работой приложений;

системы восстановления.

СУБД состоит из нескольких программных компонентов (модулей), каждый из которых предназначен для выполнения определенной операции. На схеме также показано, как СУБД взаимодействует с другими программными компонентами, например с такими, как пользовательские запросы и методы доступа (т.е. методы управления файлами, используемые при сохранении и извлечении записей с данными).

Процессор запросов. Это основной компонент СУБД, который преобразует запросы в последовательность низкоуровневых инструкций для контроллера базы данных.



Основные компоненты типичной системы управления базами данных

Контроллер базы данных. Этот компонент взаимодействует с запущенными пользователями прикладными программами и запросами. Контроллер базы данных принимает запросы и проверяет внешние и концептуальные схемы для определения тех концептуальных записей, которые необходимы для удовлетворения требований запроса. Затем контроллер базы данных вызывает контроллер файлов для выполнения поступившего запроса.

Контроллер файлов манипулирует предназначенными для хранения данных файлами и отвечает за распределение доступного дискового пространства. Он создает и поддерживает список структур и индексов, определенных во внутренней схеме. Если используются хешированные файлы, то в его обязанности входит и вызов функций хеширования для генерации адресов записей. Однако контроллер файлов не управляет физическим вводом и

выводом данных непосредственно, а лишь передает запросы соответствующим методам доступа, которые считывают данные в системные буферы или записывают их оттуда на диск.

Препроцессор языка DML. Этот модуль преобразует внедренные в прикладные программы DML-операторы в вызовы стандартных функций базового языка. Для генерации соответствующего кода препроцессор языка DML должен взаимодействовать с процессором запросов.

Компилятор языка DDL. Компилятор языка DDL преобразует DDL-команды в набор таблиц, содержащих метаданные. Затем эти таблицы сохраняются в системном каталоге, а управляющая информация - в заголовках файлов с данными.

Контроллер словаря. Контроллер словаря управляет доступом к системному каталогу и обеспечивает работу с ним. Системный каталог доступен большинству компонентов СУБД.

Ниже перечислены основные программные компоненты, входящие в состав контроллера базы данных.

Контроль прав доступа. Этот модуль проверяет наличие у данного пользователя полномочий для выполнения затребованной операции.

Процессор команд. После проверки полномочий пользователя для выполнения затребованной операции управление передается процессору команд.

Средства контроля целостности. В случае операций, которые изменяют содержимое базы данных, средства контроля целостности выполняют проверку того, удовлетворяет ли затребованная операция всем установленным ограничениям поддержки целостности данных (например, требованиям, установленным для ключей).

Оптимизатор запросов. Этот модуль определяет оптимальную стратегию выполнения запроса.

Контроллер транзакций. Этот модуль осуществляет требуемую обработку операций, поступающих в процессе выполнения транзакций.

Планировщик. Этот модуль отвечает за бесконфликтное выполнение параллельных операций с базой данных. Он управляет относительным порядком выполнения операций, затребованных в отдельных транзакциях.

Контроллер восстановления. Этот модуль гарантирует восстановление базы данных до непротиворечивого состояния при возникновении сбоев. В частности, он отвечает за фиксацию и отмену результатов выполнения транзакций.

Контроллер буферов. Этот модуль отвечает за перенос данных между оперативной памятью и вторичным запоминающим устройством - например, жестким диском или магнитной лентой. Контроллер восстановления и контроллер буферов иногда (в совокупности) называют контроллером данных.

Для воплощения базы данных на физическом уровне помимо перечисленных выше модулей нужны некоторые другие структуры данных. К ним относятся файлы данных и индексов, а также системный каталог. Группой DAFTG (Data-base Architecture Framework Task Group) была предпринята попытка стандартизации СУБД, и в 1986 году ею была предложена некоторая эталонная модель. Назначение эталонной модели заключается в определении концептуальных рамок для разделения предпринимаемых попыток стандартизации на более управляемые части и указания взаимосвязей между ними на очень широком уровне.

3. Типовые информационные процедуры, реализуемые СУБД.

Типовые информационные процедуры, реализуемые СУБД:

1. Определение структуры создаваемой базы данных, ее инициализация и проведение начальной загрузки. Данная процедура позволяет описать и создать в памяти структуру таблицы, провести начальную загрузку данных в таблицы.

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

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

3. Обеспечение независимости прикладных программ и данных (логической и физической независимости).Обеспечение логической независимости данных предоставляет возможность изменения логического представления базы данных без необходимости изменения физических структур хранения данных. Обеспечение физической независимости данных предоставляет возможность изменять способы организации базы данных в памяти ЭВМ не вызывая необходимости изменения "логического" представления данных.

4. Защита логической целостности базы данных.

Основной целью реализации этой функции является повышение достоверности данных в базе данных. Достоверность данных может быть нарушена при их вводе в БД или при неправомерных действиях процедур обработки данных, получающих и заносящих в БД неправильные данные.

5. Защита физической целостности.

Развитые СУБД имеют средства восстановления базы данных, ведение журнала транзакций.

6. Управление полномочиями пользователей на доступ к базе данных.

7. Синхронизация работы нескольких пользователей.

Коллизии(одновременной обращение к одним данным) могут привести к нарушению логической целостности данных, поэтому система должна предусматривать меры, не допускающие обновление данных другим пользователям, пока работающий с этими данными пользователь полностью не закончит с ними работать.

8. Управление ресурсами среды хранения.

9. Поддержка деятельности системного персонала.

При эксплуатации базы данных может возникать необходимость изменения параметров СУБД, выбора новых методов доступа, изменения (в определенных пределах) структуры хранимых данных, а также выполнения ряда других общесистемных действий.

4. Общие сведения о СУБД.

Первые СУБД появились в конце 60-х годов, расцвет их применения приходится на 70-е начало 80-х годов. В то время на первый план выступала способность СУБД хранить данные сложной структуры и значительного объема и использовать установленные связи между информационными элементами при проектировании приложений. Другое важное достоинство - это обеспечение относительной независимости программ от структур хранения. Первые СУБД были ориентированы на программистов. Интерфейс с такими СУБД осуществлялся путем обращения к программе – СУБД из программы приложения, написанной на одном из базовых языков программирования и такие системы стали называть системами с базовым языком. При этом СУБД выполняла лишь простые операции выборки записей, удовлетворяющих определенным условиям и в определенной последовательности, а также включения, замены и удаления записей. Но все эти операции осуществлялись с учетом зафиксированной структуры БД, что существенно сокращало алгоритмическую часть программы, касающуюся согласованной выборки связанных записей, и снижало риск нарушения структурной целостности БД.

IMS (Information Management System) являлись весьма распространенными СУБД, обеспечивающими хранение и доступ к БД иерархической структуры.

Элементом структуры является сегмент, который может состоять из одного или нескольких полей. Поле может иметь тип: символьный или числовой. Сегменты одного типа имеют единую для этого типа внутреннюю структуру и размер, в том числе фиксированные размеры и типы одноименных полей в различных реализациях сегментов.

БД сложной логической структуры может быть разнесена в 10 физических наборов данных (файлов). IMS поддерживает 4 способа физической организации БД: иерархический

последовательный метод доступа; иерархический индексно-последовательный метод доступа; иерархический индексно-прямой метод доступа и иерархический прямой.

Система IMS(ОКА) является системой с базовым языком.

Единицей обмена между прикладной программой и БД является реализация сегмента определенного типа (с подсоединенными ключами старших сегментов) или совокупности реализаций сегментов, расположенных в одной иерархической ветви.

Система обеспечивает выполнение всех типичных для СУБД функций: вставку, замену, удаление и чтение реализаций сегментов.

Система ID

СУБД IDS обеспечивают хранение и доступ к БД с сетевой структурой.

Основной структурной единицей, обеспечивающей в конечном счете сетевую структуру любой сложности, является цепь. В цепь объединяются информационно зависимые логические записи.

Запись представляет собой основную единицу информации и состоит из полей данных и служебных полей.

Служебные поля предназначены для идентификации записи и организации цепей с помощью адресных ссылок.

Информационные поля могут содержать числовые и символьные данные и имеют имена.

Записи одного типа содержат одноименные данные и имеют фиксированную длину, равно как и одноименные поля. Записям одного типа присваиваются уникальное имя и тип. Записи адресуются в БД с помощью адресных ссылок, состоящих из номера «страницы» БД и номера байта начала записи на «странице». Цепи образуются с помощью адресных ссылок, хранящихся в записи и указывающих на следующую запись в этой цепи. Каждый цепи присваивается уникальное имя. Каждая запись может входить в любое количество цепей.

Система также обеспечивает типовые операции доступа к данным – запоминание, выборку, модификацию и удаление записей из программ, написанных на классических языках программирования.

Система ADABAS

ADABAS сочетает в себе возможности СУБД с базовым языком и системы замкнутого типа и обеспечивает поддержание ограниченных сетевой и иерархической структур. БД в ADABAS является совокупностью связанных данных, организованных в виде файлов. Реализованное в системе динамическое установление связей между записями различных файлов позволяет создавать межзаписные иерархические и сетевые структуры. ADABAS осуществляет поиск по запросам пользователей записей в БД. Доступ к хранимым данным, обработку данных и выдачу результатов в виде справок и сводок заданной формы. Запросы вводятся в диалоговом режиме с видеотерминалов и в пакетном режиме в образе перфокарт. Языки запросов высокого уровня ориентированы на пользователей-непрограммистов. Обеспечивается и мультидоступ к БД.

ADABAS дает возможность обращаться к БД и из прикладных программ, написанных на АССЕМБЛЕРЕ, КОБОЛе, ФОРТРАНе и PL/1, в том числе под управлением телемонитора.

Система FoxPro

СУБД FoxPro обеспечивает возможность работы в трех режимах:

• выполнение откомпилированных программ, написанных на языке СУБД (xBASE-язык);

• выполнение команд языка xBASE или команд SQL в режиме интер-претации;

• обработка баз данных в экранном интерфейсе (без предварительного программирования и без знания команд языка xBASE).

База данных FoxPro состоит из совокупности взаимосвязанных файлов, каждый из которых может быть представлен в виде таблицы. Запись файла соответствует строке

таблицы, поле – столбцу таблицы. Файл и поля имеют имена – идентификаторы назначаемые при первоначальном создании файла.

Все записи одного файла однотипны, имеют фиксированную длину, потому как одноименные поля во всех записях имеют один и тот же размер и тип значения. В одной записи (строке таблицы) у одного поля может быть только единственное значение. В файле (таблице) не может быть двух одинаковых записей. Одно либо несколько полей являются ключом записи. Значение ключа уникально.

Файлы могут быть связаны друг с другом:

• Возможные виды связей 1 : 1, 1 : M, M : 1, M : N:

Система Access

СУБД Access является СУБД реляционного типа, в которой разумно сбалансированы все средства и возможности, типичные для современных СУБД. В отличии от FoxPro, в Access все данные вместе с запросами и формами физически хранятся в одном файле.

База данных Microsoft Access состоит из следующих объектов:

- таблицы,
- запросы,
- формы,
- отчеты,
- макросы,
- модули.

Поскольку, как правило, система управления базами данных обрабатывает одновременно несколько таблиц, то существует возможность установления реляционных связей между таблицами.

Клиент-сервер

При использовании клиент-серверной технологии, на самом сервере, содержащим базу данных, функционирует некоторое программное обеспечение, которое называется "Сервером баз данных" или "Сервером БД". Клиент-серверная СУБД позволяет обмениваться клиенту и серверу минимально необходимыми объёмами информации. При этом основная вычислительная нагрузка ложится на сервер. Клиент может выполнять функции предварительной обработки перед передачей информации серверу, но в основном его функции за-ключаются в организации доступа пользователя к серверу. Таким образом, архитектура клиент-сервер адаптирована для работы с большими объемами данных - сеть нагружается меньше, требования к пользовательским компьютерам, с точки зрения производительности, минимизируется. Однако возрастают требования к серверу, содержащему базу данных, поскольку теперь он один тянет нагрузку всех пользователей. Клиент-серверная СУБД располагается на сервере вместе с БД и осуществляет доступ к БД непосредственно, в монопольном режиме. Все клиентские запросы на обработку данных обрабатываются клиент-серверной СУБД централизованно. Недостаток клиент-серверных СУБД состоит в повышенных требованиях к серверу. Достоинства: потенциально более низкая загрузка локальной сети; удобство централизованного управления; удобство обеспечения таких важных характеристик как высо-кая надёжность, высокая доступность и высокая безопасность.

Примеры: Oracle, Firebird, Interbase, IBM DB2, Informix, MS SQL Server, Sybase Adaptive Server Enterprise, PostgreSQL, MySQL, Cache.

1.7 Лекция № 14, 15 (4 часа).

Тема: «Создание и модификация базы данных»

1.7.1 Вопросы лекции:

1. Проектирование с использованием метода сущность – связь
2. Традиционные методики проектирования БД
3. Современная интеграционная методика проектирования

4. Проектирование системы баз данных на принципах единой информационной среды

1.7.2 Краткое содержание вопросов:

1. Проектирование с использованием метода сущность – связь.

Метод "сущность–связь" (entity–relation, ER–method) является комбинацией двух предыдущих и обладает достоинствами обоих. Этап инфологического проектирования начинается с моделирования ПО. Проектировщик разбивает её на ряд локальных областей, каждая из которых (в идеале) включает в себя информацию, достаточную для обеспечения запросов отдельной группы будущих пользователей или решения отдельной задачи (подзадачи). Каждое локальное представление моделируется отдельно, затем они объединяются.

Выбор локального представления зависит от масштабов ПО. Обычно она разбивается на локальные области таким образом, чтобы каждая из них соответствовала отдельному внешнему приложению и содержала 6-7 сущностей.

Сущность – это объект, о котором в системе будет накапливаться информация. Сущности бывают как физически существующие (например, СОТРУДНИК или АВТОМОБИЛЬ), так и абстрактные (например, ЭКЗАМЕН или ДИАГНОЗ).

Для сущностей различают тип сущности и экземпляр. Тип характеризуется именем и списком свойств, а экземпляр – конкретными значениями свойств.

Типы сущностей можно классифицировать как сильные и слабые. Сильные сущности существуют сами по себе, а существование слабых сущностей зависит от существования сильных. Например, читатель библиотеки – сильная сущность, а абонемент этого читателя – слабая, которая зависит от наличия соответствующего читателя. Слабые сущности называют подчинёнными (дочерними), а сильные – базовыми (основными, родительскими).

Для каждой сущности выбираются свойства (атрибуты). Различают:

- Идентифицирующие и описательные атрибуты. Идентифицирующие атрибуты имеют уникальное значение для сущностей данного типа и являются потенциальными ключами. Они позволяют однозначно распознавать экземпляры сущности. Из потенциальных ключей выбирается один первичный ключ (ПК). В качестве ПК обычно выбирается потенциальный ключ, по которому чаще происходит обращение к экземплярам записи. Кроме того, ПК должен включать в свой состав минимально необходимое для идентификации количество атрибутов. Остальные атрибуты называются описательными и заключают в себе интересующие свойства сущности.

- Составные и простые атрибуты. Простой атрибут состоит из одного компонента, его значение неделимо. Составной атрибут является комбинацией нескольких компонентов, возможно, принадлежащих разным типам данных (например, ФИО или адрес). Решение о том, использовать составной атрибут или разбивать его на компоненты, зависит от характера его обработки и формата пользовательского представления этого атрибута.

- Однозначные и многозначные атрибуты (могут иметь соответственно одно или много значений для каждого экземпляра сущности).

- Основные и производные атрибуты. Значение основного атрибута не зависит от других атрибутов. Значение производного атрибута вычисляется на основе значений других атрибутов (например, возраст студента вычисляется на основе даты его рождения и текущей даты).

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

Далее осуществляется спецификация связей внутри локального представления. Связи могут иметь различный содержательный смысл (семантику). Различают связи типа "сущность-сущность", "сущность-атрибут" и "атрибут-атрибут" для отношений между атрибутами, которые характеризуют одну и ту же сущность или одну и ту же связь типа "сущность-сущность".

Каждая связь характеризуется именем, обязательностью, типом и степенью. Различают факультативные и обязательные связи. Если вновь порождённый объект одного типа оказывается по необходимости связанным с объектом другого типа, то между этими типами объектов существует обязательная связь (обозначается двойной линией). Иначе связь является факультативной.

По типу различают множественные связи "один к одному" (1:1), "один ко многим" (1:n) и "многие ко многим" (m:n). ER–диаграмма, содержащая различные типы связей, приведена на рис. 1. Обратите внимание, что обязательные связи на рис. 1 выделены двойной линией.

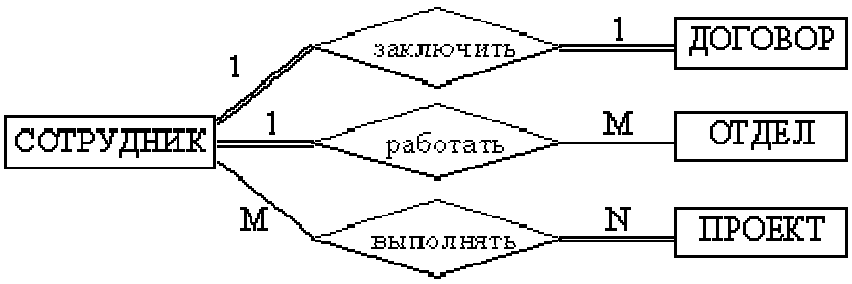


Рис.1. ER–диаграмма с примерами типов множественных связей

Степень связи определяется количеством сущностей, которые охвачены данной связью. Пример бинарной связи – связь между отделом и сотрудниками, которые в нём работают. Примером тернарной связи является связь типа экзамен между сущностями ДИСЦИПЛИНА, СТУДЕНТ, ПРЕПОДАВАТЕЛЬ. Из последнего примера видно, что связь также может иметь атрибуты (в данном случае это Дата проведения и Оценка). Пример ER–диаграммы с указанием сущностей, их атрибутов и связей приведен на рис. 2.

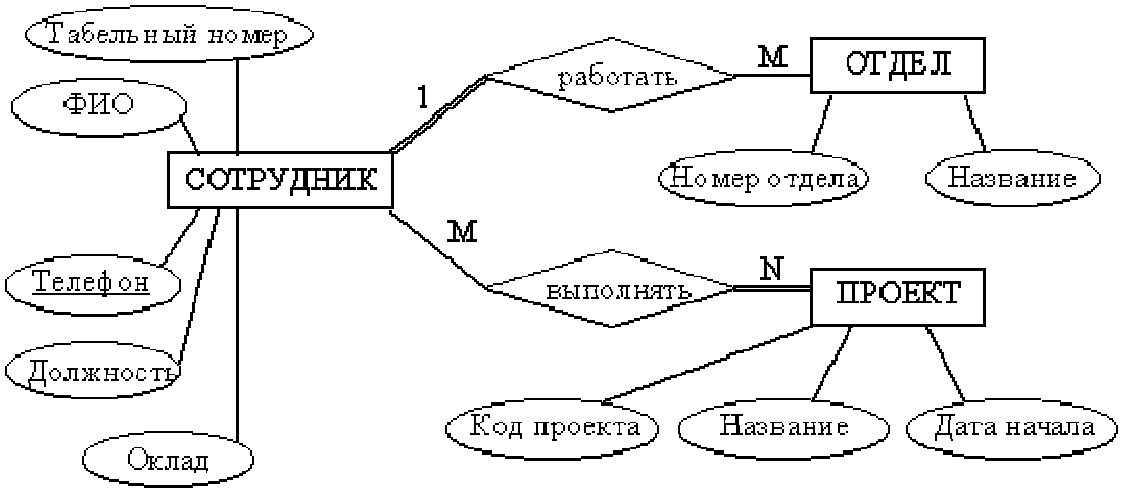


Рис.2. Пример ER–диаграммы с однозначными и многозначными атрибутами

После того, как созданы локальные представления, выполняется их объединение. При небольшом количестве локальных областей (не более пяти) они объединяются за один шаг. В противном случае обычно выполняют бинарное объединение в несколько этапов.

При объединении проектировщик может формировать конструкции, производные по отношению к тем, которые были использованы в локальных представлениях. Такой подход может преследовать следующие цели:

- объединение в единое целое фрагментарных представлений о различных свойствах одного и того же объекта;

- введение абстрактных понятий, удобных для решения задач системы, установление их связи с конкретными понятиями, использованными в модели;

- образование классов и подклассов подобных объектов (например, класс "изделие" и подклассы типов изделий, производимых на предприятии).

На этапе объединения необходимо выявить и устранить все противоречия. Например, одинаковые названия семантически различных объектов или связей или несогласованные ограничения целостности на одни и те же атрибуты в разных приложениях. Устранение противоречий вызывает необходимость возврата к этапу моделирования локальных представлений с целью внесения в них соответствующих изменений.

По завершении объединения результаты проектирования являют собой концептуальную инфологическую модель предметной области. Модели локальных представлений – это внешние инфологические модели.

2. Традиционные методики проектирования БД.

Методология проектирования, включающая в себя совокупность теоретических и инженерных знаний, обеспечивает упорядоченное создание больших информационных систем. Благодаря эффективной методологии проектирование информационных систем становится процессом создания промышленных, строго регламентированных изделий.

Методология проектирования включает три части:

- основные концепции и понятия, используемые при проектировании и реализации систем;

- технологию, организацию и управление процессом проектирования;

- инструментальные средства.

Попытка сформулировать общие требования к методологиям проектирования была предпринята в работе.

Главным требованием является охват методологией как можно, большего числа этапов жизненного цикла системы, которые предусматривают: оценку целей и возможностей создания системы, анализ требований; детальное проектирование; программирование и тестирование; интеграцию с существующей системой; внедрение и поддержку.

Важным требованием любой методологии является взаимосвязь этапов. Должна обеспечиваться связь с другими проектами, например преемственность стандартов.

Методология проектирования должна обеспечивать представление семантических требований к создаваемой системе и минимизировать потери информации при переходе от одного семантического уровня представления к другому. В этой связи большое значение имеют средства спецификации, используемые на различных уровнях представления данных.

Под спецификацией понимается точное, полное описание требований, ясно сформулированное в терминах, характерных для целей данной задачи, а не для ее реализации. Спецификации должны использоваться не только при проектировании программ и данных, но и при контроле их противоречивости.

Главное различие между методологиями, как правило, заключается в том, каким способом и в результате каких технологических операций может быть специфицирован проект создаваемой информационной системы.

Современные методологии проектирования ИС должны обеспечивать представление следующей информации:

- описание объекта автоматизации, а также места разрабатываемой информационной системы и целей, которые должны быть достигнуты в процессе разработки системы;

- описание функциональных возможностей ИС, достаточное для решения вопроса о том, что поставленные цели автоматизации достижимы;

- спецификации проекта, гарантирующие достижение заданных технических характеристик системы;

- описание реализации предлагаемой системы, достаточное для оценки времени разработки системы и необходимых для этой цели трудозатрат;

- детальный план создания системы с оценкой сроков разработки.

Таким образом, современная методология проектирования должна поддерживать сбор данных, их анализ, проектирование, оценку проекта и оценку возможности удовлетворения техническим характеристикам разрабатываемой системы.

Повышение производительности разработки программного обеспечения информационных систем достигается в основном за счет применения более совершенных системных программных средств и методологий проектирования.

Эволюция информационных систем от пакетной, массовой технологии обработки данных к системам, использующим базы данных, выявила три класса наиболее перспективных методологий проектирования. Первый из них ориентирован на концептуальное моделирование предметной области и технологию баз данных, второй — на выявление требований и спецификацию информационной системы через ее макетирование, третий — на системную архитектуру программных средств, поддерживаемую инструментальными средствами CASE (Computer Aided System Engineering) — технологии.

Методология проектирования информационных систем на основе концептуального (понятийного) моделирования предметной области (ПО) — одна из наиболее часто используемых. Она представляет собой структурированный процесс создания систем, который обычно разбивается на следующие шаги: анализ, проектирование, программирование, тестирование и внедрение.

При концептуальном моделировании ПО и применении технологии БД наиболее сложной задачей является выявление информационных и функциональных (динамических) связей между объектами реального мира.

Информационная структура ПО содержит все объекты и их связи, которые необходимы для построения ИС, а функциональная структура определяет, каким образом используются и обрабатываются эти объекты. Информационная и функциональная структуры совместно обеспечивают полную спецификацию информационной системы.

Создание ДИС на основе методологии концептуального проектирования предполагает четыре этапа проектирования:

- сбор и анализ информационных потребностей пользователей и системный анализ предметной области;
- построение концептуальной (понятийной) модели предметной области;
- создание концептуальной модели базы данных;
- разработку системы с помощью инструментальных средств выбранной СУБД.

Первый очень важный этап разработки системы — анализ требований может быть определен как этап понимания задач приложений. Именно здесь пользователи будущей системы должны сформулировать, а разработчики понять, что должна делать система, какие у нее особенности, какие понятия используются в задачах, какие ситуации предметной области должны моделироваться в базе данных.

На втором этапе разработчики системы должны определить устойчивые свойства данных и описать информационные и технологические процессы, использующие данные, их взаимосвязь и характеристики. Иногда эту работу определяют как построение концептуальной (инфологической) модели предметной области, содержащей описание понятий, не ориентированное ни на какую конкретную даталогическую модель.

Концептуальная модель предметной области ориентирована на восприятие человека (пользователя и разработчика), а не на обработку данных в ЭВМ. Именно с помощью этой модели разработчики ИС (да и сами пользователи) достигают высокого уровня понимания существа информационных потребностей пользователей. В настоящее время для построения концептуальной модели предметной области обычно используют два подхода. При первом подходе модель ПО строится на основе интеграции спецификаций информационных потребностей, а при втором — на основе непосредственного анализа самой ПО.

В первом случае более широко применяются инструментальные средства системных программных продуктов, которые интегрируются в единую программную систему,

обеспечивающую обработку доступной информации как на этапе анализа, так и на этапе проектирования прикладной системы.

В методологиях проектирования, основанных на непосредственном создании концептуальной модели предметной области, основной задачей является получение формального (независимого от СУБД) описания предметной области, которая должна моделироваться в БД. При этом система проектирования и методология проектирования должны поддерживать как получение от пользователей знаний о свойствах предметной области, так и отображение этих упорядоченных и организованных знаний в набор предварительных описаний, составляющих собственно концептуальную модель предметной области.

Концептуальная модель включает описание понятий предметной области и информационных процессов, протекающих в ней, т. е. содержит всю необходимую для проектирования системы информацию. Информации должно быть достаточно, чтобы принимать правильные решения при проектировании не только БД, но и программной реализации задач.

Однако не все понятия и операции, используемые при описании в рамках информационных процессов ПО, должны моделироваться в БД системы. Ряд семантических преобразований данных осуществляется при реализации приложений. Например, такие преобразования могут выполняться задачами, обеспечивающими специфическую обработку информации для формирования выходных отчетов. Поэтому основной проблемой третьего этапа является принятие решения о выделении из множества понятий концептуальной модели предметной области таких объектов, которые должны моделироваться в БД.

На первых трех этапах проводится не зависящий от технических и системных программных средств анализ целей и назначения проектируемой ДИС и моделируются основные информационные и технологические процессы ее функционирования. Результаты, полученные на этих этапах, имеют фундаментальный характер и не изменяются при развитии технической и программной базы ИС. Напротив, заключительный этап проектирования тесно связан с возможностями инструментальных средств конкретных СУБД.

Данный этап в свою очередь разбивают на следующие шаги:

логическое проектирование БД;

физическое проектирование БД;

реализация приложений.

В соответствии с вышеизложенным методология проектирования ИС на основе концептуального моделирования ПО может быть представлена в виде последовательности этапов, изображенных на рис. 1.4.

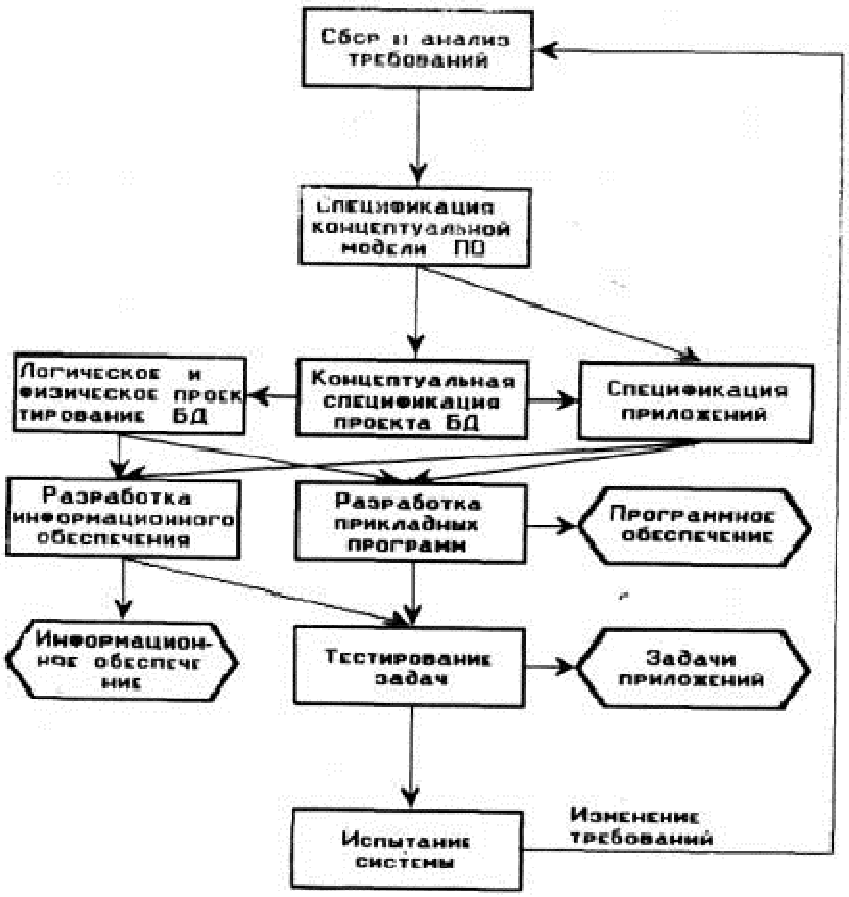


Рис. 1.4. Этапы проектирования ИС на основе концептуального моделирования ПО

Основу данной методологии составляет жестко регламентированная последовательность шагов, каждый из которых должен завершаться формальной спецификацией вне зависимости от того, насколько точно сформулированы спецификации предыдущих этапов проектирования. Такая стратегия проектирования обычно гарантирует разработку удовлетворительной системы при условии, что требования к системе фиксируются и их изменение не допускается до тех пор, пока система не предъявляется пользователям. Однако в этом случае ошибки проектирования могут быть обнаружены только на последних стадиях проектирования (либо при тестировании системы, либо во время проведения ее испытаний), что требует пересмотра всех спецификаций проекта и ведет к значительным трудозатратам.

Для поддержания методологии проектирования на основе концептуального моделирования ПО широко применяются языки спецификаций, непроцедурные языки программирования, инструментальные средства отладки программ и т. д. Однако возможность повышения производительности проектирования ИС при данном подходе остается ограниченной, так как на этапе внедрения часто выясняется, что важные требования к ИС остались неучтенными в созданной системе, а исправление допущенных ошибок требует больших трудозатрат.

Среди наиболее крупных работ в области концептуального моделирования, появившихся в последнее время, можно отметить модели Броуди, Кинга и Милополуоса, проект DATAID, где рассмотрены три модели концептуального моделирования, а также модель, используемую в методике REMORA. Значительным теоретическим достижением является также проект DIADA, в котором предложена трехуровневая структура инструментальных средств проектирования интерактивных информационных систем с интенсивным использованием данных.

Особенность указанных методологий заключается в интегрированном рассмотрении понятий, используемых для моделирования структурных и поведенческих свойств объектов реального мира.

Структурные свойства описываются с помощью абстракций типизации, агрегации, обобщения и ассоциации, а для описания поведенческих аспектов чаще всего употребляется концепция события.

Абстракции обеспечивают четкое, структурированное представление об объектах предметной области, их классах и взаимосвязях между объектами и классами объектов.

События, отражая факт изменения происшедшего в одном или нескольких объектах, позволяют адекватно учесть динамические процессы, происходящие в ПО.

3. Современная интеграционная методика проектирования.

Направление интегрированных или федеративных систем неоднородных БД и мульти-БД появилось в связи с необходимостью комплексирования систем БД, основанных на разных моделях данных и управляемых разными СУБД.

Основной задачей интеграции неоднородных БД является предоставление пользователям интегрированной системы глобальной схемы БД, представленной в некоторой модели данных, и автоматическое преобразование операторов манипулирования БД глобального уровня в операторы, понятные соответствующим локальным СУБД. В теоретическом плане проблемы преобразования решены, имеются реализации.

При строгой интеграции неоднородных БД локальные системы БД утрачивают свою автономность. После включения локальной БД в федеративную систему все дальнейшие действия с ней, включая администрирование, должны вестись на глобальном уровне. Поскольку пользователи часто не соглашаются утрачивать локальную автономность, желая тем не менее иметь возможность работать со всеми локальными СУБД на одном языке и формулировать запросы с одновременным указанием разных локальных БД, развивается направление мульти-БД. В системах мульти-БД не поддерживается глобальная схема интегрированной БД и применяются специальные способы именования для доступа к объектам локальных БД. Как правило, в таких системах на глобальном уровне допускается только выборка данных. Это позволяет сохранить автономность локальных БД.

Как правило, интегрировать приходится неоднородные БД, распределенные в вычислительной сети. Это в значительной степени усложняет реализацию. Дополнительно к собственным проблемам интеграции приходится решать все проблемы, присущие распределенным СУБД: управление глобальными транзакциями, сетевую оптимизацию запросов и т.д. Очень трудно добиться эффективности.

Как правило, для внешнего представления интегрированных и мульти-БД используется (иногда расширенная) реляционная модель данных. В последнее время все чаще предлагается использовать объектно-ориентированные модели, но на практике пока основой является реляционная модель. Поэтому, в частности, включение в интегрированную систему локальной реляционной СУБД существенно проще и эффективнее, чем включение СУБД, основанной на другой модели данных.

4. Проектирование системы баз данных на принципах единой информационной среды.

Говоря о единстве информационных процессов мы должны понимать, что информационные системы как таковые окружают нас повсюду. Более того, мы практически всегда имеем возможность найти основания для их концептуального объединения. Например, сетевая операционная система локальной вычислительной сети, функционирующая в рамках одного факультета университета, является объединяющей информационной системой для каждой отдельной инфосистемы – операционной системы персонального компьютера. Факультетская же сеть, в свою очередь, является элементом общей университетской локальной сети. Другой подход, акцентирующий программную

составляющую системной логики, позволяет нам рассматривать в качестве инфосистемы контроллер домена, с установленной на нем операционной системой Windows Server, и, соответственно, единой инфосистемой будет вся структура домена – комплекс входящих в него компьютеров под управлением главного сервера. Таким образом, мы видим, что различия в подходах к определению единой информационной системы связаны с различными основаниями для объединения отдельных инфосистем в единую сеть.

Один из авторов предлагает под единой средой взаимодействия (ЕСВ) понимать «совокупность технических и программных средств учреждения, реализующих идеи и методы автоматизации в сфере делового общения. Комплексная автоматизация подразумевает перевод в плоскость компьютерных технологий все основные деловые процессы организации (из которых нас интересует как раз сфера делового общения). Современные системы управления деловыми процессами позволяют интегрировать вокруг себя различное программное обеспечение, формируя единую информационную систему".

Из приведенного определения уже можно определить основную целеустановку ЕСВ в организации, либо преобразования существующих разрозненных систем в единую. Это автоматизация учета и обработки информационных потоков в рамках объединенной инфосистемы.

В крупных организациях и на предприятиях для создания единой информационной системы используют системы ERP (Enterprise Resource Planning). ERP-системы представляют собой набор интегрированных приложений, которые позволяют создать единую среду для автоматизации планирования, учета, контроля и анализа всех основных бизнес-операций в масштабе предприятия.

Единая среда взаимодействия – это информационная система, проектируемая согласно логической модели, отражающей цели ее создания. Технологически логическая модель системы создается на основе CMS (системы управления веб-контентом) и размещается в интернете.

Использование CMS Joomla (на основе лицензии GNU) практически не имеет аналогов по соотношению «цена-функциональность-удобство».

CMS Joomla представляет собой объектно-ориентированную модульную систему, написанную на скриптовом языке PHP. Удобный, модщный и, одновременно, несложный для освоения центр управления позволяет создавать веб-порталы практически любого уровня сложности.

Создание новых разделов осуществляется из центра управления. В дальнейшем, создавать и редактировать содержание разделов и модулей (форумов, блогов и т. д.) могут пользователи системы согласно присвоенным им правам доступа.

Содержание портала можно представить в виде иерархически распределенных документов с возможностью коллективного их редактирования (модерирования), реализовывая, таким образом, механизм wiki.

Единая регистрация пользователя в системе позволит ему получить доступ (согласно его уровню полномочий) сразу ко всем ресурсам системы – различным площадкам для общения, файловым хранилищам и т. д. Вся информация (и весь контент, и все способы представления данных – шаблоны) сохраняется в единой базе данных, что позволяет легко архивировать данные и восстанавливать всю систему в случае аварийного сбоя.

1.8 Лекция №16, 17, 18 (6 часов).

Тема: «Информационные системы, основанные на БД и СУБД»

1.8.1 Вопросы лекции:

1. Обобщенная схема информационной системы, основанной на БД и СУБД
2. Состав и функции средств актуализации БД, средств обработки БД в интересах пользователей, средств администрирования БД
3. Технологии файл-сервер и клиент-сервер

1.8.2 Краткое содержание вопросов:

1. Обобщенная схема информационной системы, основанной на БД и СУБД.

Информационная система – это любая система, реализующая или поддерживающая информационный процесс.

К информационным можно относить любые системы, включающие в себя работу с информацией. В настоящее время основным помощником человека при работе с информацией является компьютер, поэтому именно его мы и будем рассматривать в качестве источника, способа изменения и хранения информационных систем. А в качестве информационных систем будем рассматривать программное обеспечение компьютера.

В зависимости от предметной области информационные системы могут весьма значительно различаться по своим функциям, архитектуре, реализации. Однако можно выделить ряд свойств, которые являются общими.

Информационные системы предназначены организации и поддержке информационного процесса, поэтому в основе любой из них лежит среда хранения и доступа к информации.

Информационные системы ориентированы на конечного пользователя, не обладающего высокой квалификацией в области вычислительной техники. Поэтому клиентские приложения информационной системы должны обладать простым, удобным, легко осваиваемым интерфейсом.

Таким образом, при разработке информационной системы приходится решать две основные задачи:

- разработка базы данных, предназначенной для хранения информации;
- разработка графического интерфейса пользователя клиентских приложений.

Подавляющее большинство информационных систем работает в режиме диалога с пользователем.

В наиболее общем случае типовые программные компоненты, входящие в состав информационной системы, реализуют:

- диалоговый ввод-вывод;
- логику диалога;
- прикладную логику обработки данных;
- логику управления данными;
- операции манипулирования файлами и (или) базами данных.

Классификация информационных систем

Информационные системы классифицируются по разным признакам:

1. Классификация по масштабу (рис1)

По масштабу информационные системы подразделяются на следующие группы:

- одиночные;
- групповые;
- корпоративные.

Рис. 1. Деление информационных систем по масштабу.

Одиночные информационные системы реализуются, как правило, на автономном персональном компьютере (сеть не используется). Такая система может содержать несколько простых приложений, связанных общим информационным фондом, и рассчитана на работу одного пользователя или группы пользователей, разделяющих по времени одно рабочее место. Подобные приложения создаются с помощью так называемых настольных, или локальных, систем управления базами данных (СУБД). Среди локальных СУБД наиболее известными являются Clarion, Clipper, FoxPro, Paradox, dBase и Microsoft Access.

Групповые информационные системы ориентированы на коллективное использование информации членами рабочей группы и чаще всего строятся на базе локальной вычислительной сети. При разработке таких приложений используют-ся серверы баз данных (называемые также SQL (Structured Query Language – структурированный язык запросов)-серверами) для рабочих групп. Существует довольно большое количество различных SQL-серверов как коммерческих, так и свободно распространяемых. Среди них наиболее известны такие серверы баз данных, как Oracle, DB2, Microsoft SQL Server, InterBase, Sybase, Informix.

Корпоративные информационные системы являются развитием систем для рабочих групп, они ориентированы на крупные компании и могут поддерживать тер-риториально разнесенные узлы или сети. В основном они имеют иерархическую структуру из нескольких уровней. Для таких систем характерна архитектура клиент-сервер со специализацией серверов или же многоуровневая архитектура. При разработке таких систем могут использоваться те же серверы баз данных, что и при разработке групповых информационных систем. Однако в крупных информационных системах наибольшее распространение получили серверы Oracle, DB2 и Microsoft SQL Server.

2. Классификация по сфере применения

По сфере применения информационные системы обычно подразделяются на четыре группы (рис. 2):

- системы обработки транзакций (протоколов);
- системы поддержки принятия решений;
- информационно-справочные системы;
- офисные информационные системы.

Рис. 2. Деление информационных систем по сфере применения.

Системы обработки транзакций, в свою очередь, по оперативности обработки данных разделяются на пакетные информационные системы и оперативные информационные системы. В информационных системах организационного управления преобладает режим оперативной обработки транзакций (OnLine Transaction Pro-cessing, OLTP) для отражения актуального состояния предметной области в любой момент времени, а пакетная обработка занимает весьма ограниченную часть. Для систем OLTP характерен регулярный (возможно, интенсивный) поток довольно простых транзакций, играющих роль заказов, платежей, запросов и т.п. Важными требованиями для них являются:

- высокая производительность обработки транзакций;
- гарантированная доставка информации при удаленном доступе к БД по телекоммуникациям.

Системы поддержки принятия решений (Decision Support System, DSS) представляют собой другой тип информационных систем, в которых с помощью довольно сложных запросов производится отбор и анализ данных в различных разрезах: временных, географических, по другим показателям.

Обширный класс информационно-справочных систем основан на гипертекстовых документах и мультимедиа. Наибольшее развитие такие информационные системы получили в Интернете.

Класс офисных информационных систем нацелен на перевод бумажных документов в электронный вид, автоматизацию делопроизводства и управление документооборотом.

3. Классификация по способу организации

По способу организации групповые и корпоративные информационные системы подразделяются на следующие классы (рис. 3):

- системы на основе архитектуры файл-сервер;
- системы на основе архитектуры клиент-сервер;
- системы на основе многоуровневой архитектуры;
- системы на основе Интернет/интранет-технологий.

Рис. 3. Деление информационных систем по способу организации.

В любой информационной системе можно выделить необходимые функциональные компоненты (табл. 1), которые помогают понять ограничения различных архитектур информационных систем. Рассмотрим более подробно особенности вариантов построения информационных приложений.

Таблица 1.1. Типовые функциональные компоненты информационной системы

Обозначение	Наименование	Характеристика
PL	Presentation Logic (логика представления)	Управляет взаимодействием между пользователем и ЭВМ. Обрабатывает действия пользователя при выборе команды в меню, щелчке на кнопке или выборе пункта в списке
BL	Business Logic (прикладная логика)	Набор правил для принятия решений, вычислений и операций, которые должно выполнить приложение
DL	Data Logic (логика управления данными)	Операции с базой данных (реализуемые SQL-операторами), которые нужно выполнить для реализации прикладной логики управления данными
DS	Data Services (операции с базой данных)	Действия СУБД, реализующие логику управления данными, такие как манипулирование данными, определение данных, фиксация или откат транзакций и т. п. СУБД обычно компилирует SQL-предложения
FS	File Services (файловые операции)	Дисковые операции чтения и записи данных для СУБД и других компонентов. Обычно являются функциями операционной системы (ОС)

2. Состав и функции средств актуализации БД, средств обработки БД в интересах пользователей, средств администрирования БД.

К основным функциям ИС относятся функции сбора и регистрации информационных ресурсов, их хранение, обработка, актуализация, а так же обработка запросов пользователя.

Сбор и регистрация обеспечивает фиксирование информации о состоянии предметной области. Работы выполняется как до основного программно-аппаратного комплекса, так в его среде. Реализация функций зависит от источника информации, в качестве которого могут выступать бумажные носители, электронные, автоматизированные технические системы.

Сбор и регистрация могут осуществляться:

- путем измерений (наблюдений) фактов в реальном мире и ввода данных в систему с помощью клавиатуры или каких-либо манипуляторов;
- полуавтоматически путем ввода в компьютер с некоторых носителей и в случае необходимости их перекодировать (например, при использовании текстов на бумажных носителях или аналоговых аудизаписей);
- автоматический с помощью различного рода датчиков или обмена данными с другими автоматизированными системами.

С этими функциями связана необходимость обеспечения контроля, сжатие, конвертирование информации.

Обеспечение контроля информации – необходимая стадия предварительной обработки данных и подготовки их загрузки в систему, особенно в случаях, когда используются несколько источников данных. Обычно она включает процедуры фильтрации данных, верификации, обеспечение логической целостности, устранение несогласованности, избыточности и различных ошибок, восполнения пропусков, а также другие процедуры направленные на улучшение качества информации.

В результате фильтрации производится отбор нужных данных из множества имеющихся в распоряжении. Верификации призвана обеспечивать достоверность и логическую целостность информации. При выполнении данной функции устанавливается, адекватна ли или информация предметной области.

На разных операциях могут применяться различные методы контроля, существуют методы, применимые ко многим операциям, наиболее применимые:

- подсчета контрольных сумм;
- повторное выполнение операций другим оператором с дублированием действий и последующим их сличением;
- контроль набора на клавиатуре;
- контроль информации в соответствие с ее свойствами, структурой и на соответствие значениям.

Способами реализации могут быть:

- ручной (без использования технических средств);
- визуальный (с использованием технических средств и без них);
- аппаратный (технический);
- программный;
- организационные мероприятия.

Выбор конкретных обеспечения верификации зависит от характера, качества, источников данных, видов ограничения целостности.

В некоторых ИС информация хранится в сжатом виде. Сжатие информации минимизирует потребность во внешней памяти, нужной для хранения, а также снижает затраты на передачу данных по каналам связи.

Конвертирование данных при вводе в систему используется для преобразования одного формата данных в другой, допускающий автоматизированный импорт их в ИС. Конвертирование данных необходимо в случаях, когда источником данных является некоторая другая система. Для конвертирования используются специальные программы конверторы.

Хранение и накопление информации вызвано необходимостью многократного использования одни и те же данные при решении задач. Для хранения и поиска информации используются технологии баз данных.

Актуализация информационных ресурсов. Для того, чтобы информация была практически полезной, необходимо своевременно и адекватно отображать в ней изменения состояния предметной области. Актуализация информации в реляционных СУБД сводится к включению и/или удалению строк в таблицах баз данных, обновлению значений некоторых реквизитов. В случаях изменения структуры предметной области системы, актуализация информации заключается в изменении схемы базы данных – добавлении или удалении существующих столбцов таблиц, в создании новых таблиц и удалении существующих таблиц.

В информационно-справочные системах актуализация информации осуществляется путем ввода в систему новых документов, реже удалением существующих.

Актуализация информации в ИС производится дискретно, через определенные интервалы времени. Актуализация информации, т.о., обеспечивается с некоторым отставание во времени. Это отставание в различных ИС изменяется в широком диапазоне и зависит от назначения системы и особенностей ее предметной области. В информационных системах управления сложными техническими объектами, например в системе управления

космическими полетами, временной лаг измеряется в миллисекундах. В корпоративных ИС может составлять от нескольких минут до нескольких часов.

Для того чтобы ИС соответствовала своему назначению необходимо соблюдать установленный для нее регламент актуализации.

Предоставление информационных ресурсов пользователю. Все выше описанные операции необходимы для удовлетворения информационных потребностей пользователей.

Существует две технологии предоставления информации пользователю: pull-технология и/ или push-технология.

В случае pull-технологии – инициатором предоставления информации выступает пользователь, а push-технология сама система, в соответствие с регламентом и для определенного круга пользователей.

Для предоставления информации по pull-технологии в ИС предусматриваются пользовательские интерфейсы. Пользовательские интерфейсы – средства взаимодействия пользователя с системой.

При этом пользователь может влиять на последовательность применения тех или иных технологий. С точки зрения влияния пользователя на последовательность операций в процессе функционирования ИС, интерфейсы могут быть разделены на пакетные и диалоговые.

Экономические задачи, решаемые в пакетном режиме, характеризуются следующими свойствами:

- алгоритм решения задачи формализован, процесс ее решения не требует вмешательства человека;
- имеется большой объем входных и выходных данных, значительная часть которых хранится на магнитных дисках;
- расчет выполняется для большинства записей входных файлов;
- большое время решения задачи обусловлено большими объемами данных;
- регламентность, т.е. задачи решаются с заданной периодичностью.

Диалоговый режим не является альтернативой пакетному режиму, а его развитием. Если применение пакетного режима позволяет уменьшить вмешательство пользователя в процесс решения задачи, то диалоговый режим предполагает отсутствие жестко закрепленной последовательности операций обработки данных.

Примером push-технологии может служить рассылка информации среди пользователей Интернет.

Экономическая информационная система по своему составу напоминает предприятие по переработке данных и производству выходной информации. Методы и способы реализации функции ИС (сбора, накопления, хранения, поиска и обработки информации на основе применения средств вычислительной техники) называются информационной технологией.

Информационные технологии должны быть выстроены в последовательность действий, позволяющую из исходной информации получить результат с заданной достоверностью и безопасностью.

Упорядоченная последовательность взаимосвязанных действий, выполняющихся с момента возникновения информации до получения результата, называется технологическим процессом.

Понятие информационной технологии, таким образом, неотделимо от той специфической среды, в которой она реализована, т.е. от технической и программной Среды.

Администрирование базы данных – это функция управления базой данных (БД). Лицо ответственное за администрирование БД называется “Администратор базы данных” (АБД) или “Database Administrator” (DBA).

Функция “администрирования данных” стала активно рассматриваться и определяться как вполне самостоятельная с конца 60-х годов. Практическое значение это имело для предприятий, использующих вычислительную технику в системах

информационного обеспечения для своей ежедневной деятельности. Специализация этой функции с течением времени совершенствовалась, но качественные изменения в этой области стали происходить с началом использования так называемых интегрированных баз данных. Одна такая база данных могла использоваться для решения многих задач.

Таким образом, сформировалось определение БД как общего информационного ресурса предприятия, которое должно находиться всегда в работоспособном состоянии. И как для каждого общего ресурса значительной важности, БД стала требовать отдельного управления. Во многих случаях это было необходимо для обеспечения её повседневной эксплуатации, её развития в соответствии с растущими потребностями предприятия. К тому же БД и технология её разработки постоянно совершенствовались и уже требовались специальные знания высокого уровня для довольно сложного объекта, которым стала база данных. Отсюда функция управления базой данных и получила название “Администрирование базы данных”, а лицо ею управляющее стали называть “Администратор баз данных”.

Администратор базы данных (DBA)

DBA Администратор базы данных (АБД) или Database Administrator (DBA) – это лицо, отвечающее за выработку требований к базе данных, её проектирование, реализацию, эффективное использование и сопровождение, включая управление учётными записями пользователей БД и защиту от несанкционированного доступа. Не менее важной функцией администратора БД является поддержка целостности базы данных.

АБД имеет код специальности по общероссийскому классификатору профессий рабочих, должностей служащих и тарифных разрядов (ОКПДТР) — 40064 и код 2139 по Общероссийскому классификатору занятий (ОКЗ). Код 2139 ОКЗ расшифровывается следующим образом: 2 - СПЕЦИАЛИСТЫ ВЫСШЕГО УРОВНЯ КВАЛИФИКАЦИИ, 21 - Специалисты в области естественных* и инженерных наук, 213 - Специалисты по компьютерам, 2139 - Специалисты по компьютерам, не вошедшие в другие группы.

Классические подходы к наполнению содержанием понятия "АБД" стали формироваться после издания рабочего отчета группы по базам данных Американского Национального Института Стандартов ANSI/X3/SPARC в 1975 года. В этом отчете была описана трехуровневая архитектура СУБД, в которой выделялся уровень внешних схем данных, уровень концептуальной схемы данных и уровень схемы физического хранения данных. В соответствии с этой архитектурой определялись три роли АБД: администратор концептуальной схемы, администратор внешних схем и администратор хранения данных. Эти роли в случае очень маленькой системы мог играть один человек, в большой системе для выполнения каждой роли могла назначаться группа людей. Каждой роли соответствовал набор функций, а все эти функции вместе составляли функции АБД.

В 1980 - 1981 г. в американской литературе стало принятым включать в функции АБД:

- организационное и техническое планирование БД,
- проектирование БД,
- обеспечение поддержки разработок прикладных программ,
- управление эксплуатацией БД.

В нашей стране в это же время первое определение АБД в ГОСТ-ах задало слишком узкий состав функций АБД:

- подготовка вычислительного комплекса к установке СУБД, участие в установке и приемке СУБД и самой БД с комплексом прикладных программ
- управление эксплуатацией БД
- подготовка словарей и другой НСИ - нормативно-справочной информации - к моменту начала испытания БД
- Предполагалось, что функции АБД будут ориентированы только на эксплуатацию БД, а её разработка будет вестись силами специализированной организации.

К середине 90-х годов сложились еще не завершенные, но уже достаточно устойчивые и полные методологии разработки систем с базами данных. Основная работа по планированию информационных потребностей предприятия, проектированию концептуальной и логической схемы БД, внешних схем, используемых в отдельных процессах обработки информации, ложится теперь на группу проектирования Автоматизированной Системы (АС). Становиться и более определённым объем функций АБД. Это обеспечение надежной и эффективной работы пользователей и программ с БД, поддержка разработчиков в их доступе к БД и средствам разработки.

Как таковой официальной версии должностной инструкции администратора базы данных не существует. Имеется несколько документов, различающихся в основном оформлением и содержанием некоторых пунктов. Естественно, ни о какой подробной расшифровке задач администратора базы данных в этих документах речи не идёт. Должностная инструкция — это, прежде всего документ, регламентирующий производственные полномочия и обязанности работника. И хотя в некоторых организациях изменяют текст должностной инструкции в соответствии с условиями специфики работы, не стоит ожидать в ней прямых указаний на то, что надо делать администратору. В большинстве случаев такие детальные директивы работы регламентируются другими документами (например, инструкция по резервному копированию, инструкция по обеспечению информационной безопасности при работе с базами данных).

3. Технологии файл-сервер и клиент-сервер.

Архитектура файл-сервер.

В архитектуре файл-сервер сетевое разделение компонентов диалога PS и PL отсутствует, а компьютер используется для функций отображения, что облегчает построение графического интерфейса. Файл-сервер только извлекает данные из файлов, так что дополнительные пользователи и приложения лишь незначительно увеличивают нагрузку на центральный процессор. Каждый новый клиент добавляет вычислительную мощность к сети.

Объектами разработки в файл-серверном приложении являются компоненты приложения, определяющие логику диалога PL, а также логику обработки BL и управления данными DL. Разработанное приложение реализуется либо в виде законченного загрузочного модуля, либо в виде специального кода для интерпретации.

Однако такая архитектура имеет существенный недостаток: при выполнении некоторых запросов к базе данных клиенту могут передаваться большие объемы данных, загружая сеть и приводя к непредсказуемости времени реакции. Значительный сетевой трафик особенно сказывается при организации удаленного доступа к базам данных на файл-сервере через низкоскоростные каналы связи. Одним из вариантов устранения данного недостатка является удаленное управление файл-серверным приложением в сети. При этом в локальной сети размещается сервер приложений, совмещенный с телекоммуникационным сервером (обычно называемым сервером доступа), в среде которого выполняются обычные файл-серверные приложения.

Одним из традиционных средств, на основе которых создаются файл-серверные системы, являются локальные СУБД. Однако такие системы, как правило, не отвечают требованиям обеспечения целостности данных (в частности, они не поддерживают транзакции). Поэтому при их использовании задача обеспечения целостности данных возлагается на программы клиентов, что приводит к усложнению клиентских приложений. Тем не менее, эти инструменты привлекают своей простотой, удобством применения и доступностью. Поэтому файл-серверные информационные системы до сих пор представляют интерес для малых рабочих групп и, более того, нередко используются в качестве информационных систем масштаба предприятия.

Архитектура клиент-сервер.

Архитектура клиент-сервер предназначена для разрешения проблем файл-серверных приложений путем разделения компонентов приложения и размещения их там, где они будут функционировать наиболее эффективно. Особенностью архитектуры клиент-сервер является наличие выделенных серверов баз данных, понимающих запросы на языке структурированных запросов (Structured Query Language, SQL) и выполняющих поиск, сортировку и агрегирование информации.

Отличительная черта серверов БД – наличие справочника данных, в котором записана структура БД, ограничения целостности данных, форматы и даже серверные процедуры обработки данных по вызову или по событиям в программе. Объектами разработки в таких приложениях помимо диалога и логики обработки являются, прежде всего, реляционная модель данных и связанный с ней набор SQL-операторов для типовых запросов к базе данных.

Большинство конфигураций клиент-сервер использует двухуровневую модель, в которой клиент обращается к услугам сервера. Предполагается, что диалоговые компоненты PS и PL размещаются на клиенте, что позволяет реализовать графический интерфейс. Компоненты управления данными DS и FS размещаются на сервере, а диалог (PS, PL) и логика (BL, DL) – на клиенте. В двухуровневом определении архитектуры клиент-сервер используется именно этот вариант: приложение работает на клиенте, СУБД – на сервере (рис. 4).

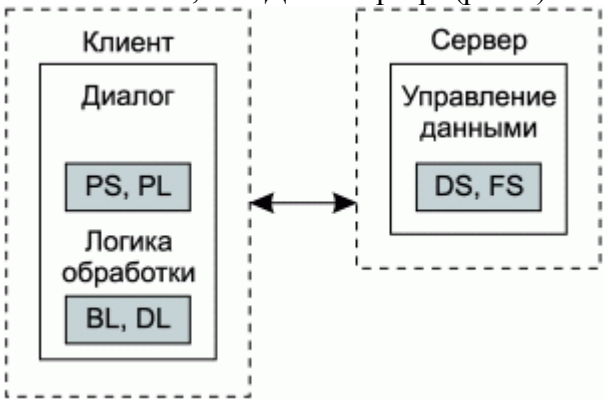


Рис. 4. Классический вариант клиент-серверной системы.

Поскольку эта схема предъявляет наименьшие требования к серверу, она обладает наилучшей масштабируемостью. Однако сложные приложения, активно взаимодействующие с БД, могут жестко загрузить как клиента, так и сеть. Результаты SQL-запроса должны вернуться клиенту для обработки, потому что там реализована логика принятия решения. Такая схема приводит к дополнительному усложнению администрирования приложений, разбросанных по различным клиентским узлам.

Для сокращения нагрузки на сеть и упрощения администрирования приложений компонент BL можно разместить на сервере. При этом вся логика принятия решений оформляется в виде хранимых процедур и выполняется на сервере БД.

Хранимая процедура – процедура с SQL-операторами для доступа к БД, вызываемая по имени с передачей требуемых параметров и выполняемая на сервере БД. Хранимые процедуры могут компилироваться, что повышает скорость их выполнения и сокращает нагрузку на сервер.

Хранимые процедуры улучшают целостность приложений и БД, гарантируют актуальность коллективных операций и вычислений. Улучшается сопровождение таких процедур, а также безопасность (нет прямого доступа к данным).

2. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

2.1 Лабораторная работа №1, интерактивная форма (2 часа).

Тема: «Персональные базы данных»

Занятие посвящается проверке полученных навыков по изучаемой теме. Студентам предлагаются для решения индивидуальные варианты заданий, а также, разбившись на группы (не более 3-х человек) они должны предложить конкретную экономическую задачу и решить ее с применением полученных навыков по методам решения и с использованием компьютерной техники. Далее представитель каждой группы представляет решенную ими задачу, защищает ее, отвечая на вопросы студентов других групп (в дискуссии также принимают участие и остальные члены группы), чем осуществляется реализация такой формы, как **разбор конкретных ситуаций**. Оценка выставляется по совокупности индивидуальных заданий и разбора реальных предложенных к рассмотрению задач.

2.1.1 Цель работы: изучить и проанализировать создание персональных баз данных

2.1.2 Задачи работы:

1. Изучение создания персональных баз данных
2. Анализ создания персональных баз данных

2.1.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер

2.1.4 Описание (ход) работы:

Персональные базы данных.

Иерархическая модель данных имеет много общих черт с сетевой моделью данных, хронологически она появилась даже раньше, чем сетевая. Допустимыми информационными конструкциями в иерархической модели данных являются отношение, веерное отношение и иерархическая база данных. В отличие от рассмотренных ранее моделей данных, где предполагалось, что информационным отображением одной предметной области является одна база данных, в иерархической модели данных допускается отображение одной предметной области в нескольких иерархических базах данных.

Понятия «отношение» и «веерное отношение» в иерархической модели данных не изменяются.

Иерархической базой данных называется множество отношений и веерных отношений, для которых соблюдаются два ограничения.

1. Существует единственное отношение, называемое корневым, которое не является зависимым ни в одном веерном отношении.
2. Все остальные отношения (за исключением корневого) являются зависимыми отношениями только в одном веерном отношении.

Схема иерархической базы данных по составу компонент идентична сетевой базе данных. Перечисленные выше ограничения поддерживаются иерархическими СУБД.

На рис. 1 изображена структура иерархической базы данных, представляющая студентов и преподавателей вуза, и удовлетворяющая всем ограничениям, указанным в определении.

В графических иллюстрациях структуры иерархической базы данных приводятся названия соответствующих отношений.

Ограничение, которое поддерживается в иерархической модели данных, состоит в невозможности нарушения требований, фигурирующих в определении иерархической базы данных. Это ограничение обеспечивается специальной укладкой значений отношений в памяти компьютера. Ниже будет рассмотрена одна из простейших реализаций укладки иерархической базы данных.

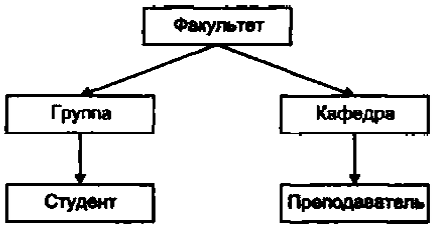


Рис. 1 – Иерархическая база данных «Вуз-Кафедра»

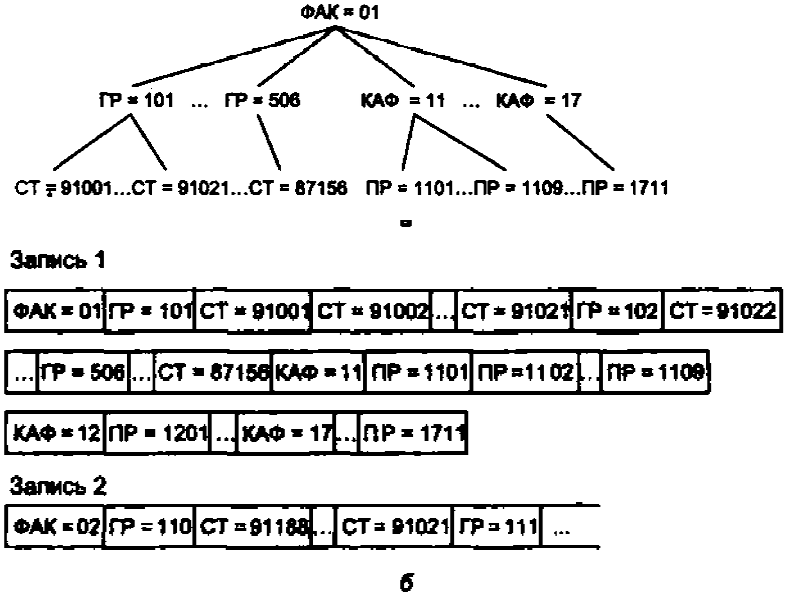


Рис. 2 – Экземпляр иерархической базы данных «Вуз-Кафедра»

На рис. 2 *а* приведена связь значений отношений из иерархической базы данных, структура которой показана на рис. 2.6. Каждое значение представляется соответствующей величиной первичного ключа. Использованы следующие сокращения: ФАК -

факультет, ГР - группа, КАФ - кафедра, СТ - студент, ПР - преподаватель.

Далее эта информация организуется в линейную последовательность значений (рис. 2 *б).*

Необходимо отметить, что известны различные возможности прохождения иерархически организованных значений в линейной последовательности. Принцип, применяемый для иерархических баз данных, называется концевым прохождением.

Правила концевого прохождения следующие:

Шаг 1. Начиная с первого значения корневого отношения перечисляются первые значения соответствующих отношений на каждом уровне вплоть до последнего.

Ш а г 2. Перечисляются все значения в том веерном отношении, на котором остановился шаг 1.

Ш а г 3. Перечисляются значения всех вееров этого веерного отношения.

Шаг 4. От достигнутого уровня происходит подъем на предыдущий уровень, и если возможно применить шаг 1, то процесс повторяется.

Записью иерархической базы данных называется множество значений, содержащих одно значение корневого отношения и все вееры, доступные от этого значения в соответствии со структурой иерархической базы данных. В нашем примере одну запись образуют данные, относящиеся к одному факультету.

Для веерных отношений в составе иерархической базы данных справедлива уже известная закономерность: если существует веерное отношение, то ключ зависимого отношения функционально определяет ключ основного отношения, и наоборот, если ключ одного отношения функционально определяет ключ второго отношения, то первое отношение может быть зависимым, а второе - основным в некотором веерном отношении.

Кроме того, ограничение на наличие единственного корневого отношения в иерархической базе данных трансформируется в требование: первичный ключ каждого некорневого отношения должен функционально определять первичный ключ корневого отношения.

Рассмотрим алгоритм формирования иерархической БД на основе известного множества реквизитов и функциональных зависимостей. Исходное множество функциональных зависимостей и реквизиты первичного ключа получаются так же, как при формировании множества отношений в ЗНФ. Алгоритм иллюстрируется тем же примером, что и для двухуровневой сети.

Алгоритм получения структуры иерархической БД

Исходные данные - список реквизитов и функциональных зависимостей в базе данных.

Шаг 1. Для каждой функциональной зависимости вида *A —> B* создать отношение Si(A,B). Каждый блок взаимно-однозначных соответствий также порождает отношение с ключом, равным старшему по объему понятия реквизиту.

В нашем примере будут созданы следующие отношения (ключи помечены знаком #):

S1 (НИИ #, Директор, Адрес),

S2(Отдел #, НИИ, Ксотр),

S3(Тема #, Датанач, Датакон, Приор),

S4(ФИО #, Отдел),

S5(Тема #, Работа #, ФИО #, Прод),

S6(Тема #, Заказ #, Обфин).

Ш а г 2. Разделить отношения на группы по признаку: два отношения находятся в общей группе, если их ключи функционально определяют хотя бы один общий реквизит.

Для отношений S1 - S6 получаем две группы:

• S1, S2,S4, S5 (все ключи функционально определяют реквизит НИИ);

• S3, S6, S5 (все ключи функционально определяют реквизит ТЕМА).

Далее шаги 3, 4, 5 выполняются раздельно для каждой группы. Количество групп определяет количество иерархических БД.

Шаг 3. У всех пар отношений группы проверить условие для ключей отношений *Ki — > Kj* Если оно соблюдается, то из соответствующих отношений создается веерное отношение Wij(Si,Sj).

Ш а г 4. Найти в группе цепи веерных отношений и образовать из них дерево. Элемент цепи образуется по условию Wij -Wik

В нашем примере получим:

группа 1) -цепь W12(S1,S2), W24(S2,S4), W45(S4,S5) образует дерево;

группа 2) - цепей нет, но W35(S3,S5) и W36(S3,S6) образуют дерево.

Ш а г 5. Реквизиты, оставшиеся вне цепей на шаге 4, добавить в структуру тех отношений, где они будут неключевыми, либо в структуру отношений, соответствующих висячим вершинам дерева.

Ш а г 6. Если группы, полученные на шаге 2, содержат общие отношения, то необходимо решить вопрос о целесообразности установления логических связей между иерархическими БД.

Ш а г 7. Сократить список реквизитов в сегментах за счет удаления реквизитов зависимого отношения, общих в паре «основной - зависимый».

Итоговая иерархическая структура для рассматриваемого примера показана на рис 3. Она содержит две иерархические базы данных. В некоторых иерархических СУБД не допускается логическая связь баз данных, так как формально это является нарушением ограничения иерархической модели данных.

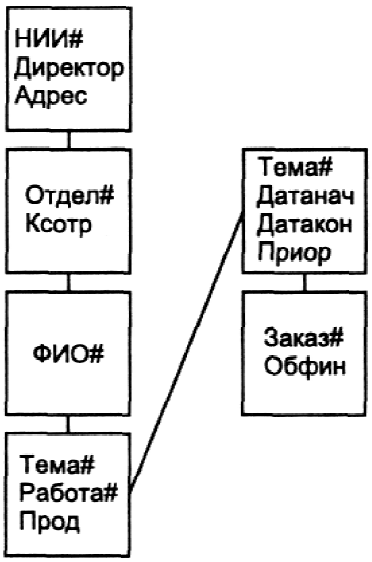


Рис. 2.8. Иерархическая база данных НИИ

Рис. 3 – Иерархическая база данных НИИ

Манипулирование иерархической базой данных происходит с применением включающего языка. Подпрограмма обращения к базе данных содержит ряд параметров, из которых мы будем использовать лишь код требуемой операции и одно или несколько условий выбора. По причинам, которые уже были перечислены при рассмотрении сетевой модели данных, для иерархической модели данных необходимы только операции выборки.

Минимальное множество вариантов выборки соответствует трем операциям:

1. GU - получить уникальную запись по известным значениям первичного ключа на каждом уровне дерева иерархической базы данных.

2. GN - получить следующую запись на том уровне дерева, где находится текущая запись после выполнения оператора СШ.

3. GNP - получить следующую запись на расположенном непосредственно ниже уровне дерева, относительно позиции, где находится текущая запись после выполнения оператора СШ или ОК.

Например, для запроса «получить информацию о преподавателе с кодом 1103 кафедры 11 факультета 01» потребуется оператор

```
GU Факультет (Факультет = "01")
   Кафедра (Кафедра = "11")
   Преподаватель (Преподаватель = "1103")
   print Преподаватель.
```

В этом примере названия отношений совпадают с названиями соответствующих первичных ключей.

Запрос «получить список всех студентов группы 10» реализуется следующей последовательностью операторов

```
GU Факультет (Факультет = "01")
   Группа (Группа="10")
   Студент M1: GN Студент
   print Студент
   goto M1.
```

Поскольку в операторе GU не указано условие выборки в отношении Студент, текущей записью станет первая запись этого отношения и далее циклическое повторение оператора (GN обеспечит требуемое извлечение всех записей о студентах группы 10. Выход из цикла произойдет в результате получения кода возврата «конец отношения» и эта ситуация должна проверяться средствами включающего языка.

ЗАДАНИЯ

Задание 1. Детализируйте приведенную ниже иерархическую структуру (рис. 4). Необходимые имена реквизитов выбрать самостоятельно. Проверьте соблюдение всех требований алгоритма получения структуры иерархической базы данных.

Рис. 4 – Структура ВУЗа

Задание 2. Для приведенной ниже иерархической структуры базы данных укажите минимально возможный набор реквизитов в отношениях.

Реквизиты: Музей, Город, Экспонат, Год, Выставка, ФИО реставратора.

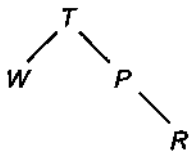
Отношения: W(Музей, Город), С(Экспонат, Год поступления), Т(Экспонат, Год реставрации, ФИО), S(Выставка, Экспонат, Год выставки).

Веерные отношения: *(W,C), (C, S),* (С, *T).*

Названия музеев и выставок не повторяются.

Задание 3. Реализуйте иерархическую структуру. Разрешается добавлять или исключать имена реквизитов в отношениях. Разработайте пример записи иерархической базы данных.

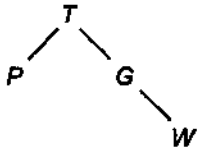
Реквизиты	**Отношения**
Автор	WАвтор, Год аттестации, Число статей)
Журнал	Т(Автор, Ученая степень)
Статья	Р(Автор, Журнал, Статья)
Год аттестации	R{(Журнал, Страна)
Число статей	
СтранаУченая степень	



Названия журналов не повторяются.

Задание 4. Реализуйте иерархическую структуру. Разрешается добавлять или исключать имена реквизитов в отношениях. Разработайте пример записи иерархической базы данных.

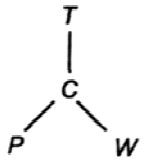
Реквизиты	**Отношения**
Учреждение	Т(Учреждение, Отдел)
Отдел	G(Отдел, Тема)
Тема	Р(Фио, Отдел)
Код оборудования	W(Тема, Код оборудования, Продолжительность)
Фио сотрудника	

Продолжительность работы 

Каждая тема выполняется в единственном отделе.

Задание 5. Реализуйте иерархическую структуру. Разрешается добавлять или исключать имена реквизитов в отношениях. Разработайте пример записи иерархической базы данных.

Реквизиты	**Отношения**
Город	T(Город, Страна)
Страна	C(Команда, Город)
Команда	P(Фио, Команда, Год рождения)
Фио игрока	W(Дата, Команда-соперник, Счет)
Год рождения	
Дата	
Команда-соперник Счет	



Названия команд не повторяются.

2.2 Лабораторная работа № 2 (2 часа).

Тема: «Реляционная алгебра Кодда»

Занятие посвящается проверке полученных навыков по изучаемой теме. Студентам предлагаются для решения индивидуальные варианты заданий, а также, разбившись на группы (не более 3-х человек) они должны предложить конкретную экономическую задачу и решить ее с применением полученных навыков по методам решения и с использованием компьютерной техники. Далее представитель каждой группы представляет решенную ими задачу, защищает ее, отвечая на вопросы студентов других групп (в дискуссии также принимают участие и остальные члены группы), чем осуществляется реализация такой формы, как **разбор конкретных ситуаций**. Оценка выставляется по совокупности индивидуальных заданий и разбора реальных предложенных к рассмотрению задач.

2.2.1 Цель работы: изучить операции реляционной алгебры Кодда. Использовать для решения задач программный продукт MS Excel

2.2.2 Задачи работы:

1. Изучение операций реляционной алгебры Кодда
2. Использование программного продукта MS Excel при решение данных задач

2.2.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Excel

2.2.4 Описание (ход) работы:

Реляционная алгебра Кодда.

Операции реляционной алгебры Кодда можно разделить на *две* группы: *базовые теоретико-множественные* и *специальные реляционные.* Первая группа операций включает в себя четыре классические операции теории множеств: объединение, разность, пересечение и произведение. Вторая группа представляет собой развитие обычных теоретико-множественных операций в направлении к реальным задачам манипулирования данными, в ее состав входят следующие операции: проекция, селекция, деление и соединение. Операции реляционной алгебры могут выполняться над одним отношением (например, проекция) или над двумя

отношениями (например, объединение). В первом случае *операция называется унарной*, а во втором — *бинарной*. При выполнении бинарной операции участвующие в операциях отношения должны быть совместимы по структуре.

Совместимость структур отношений означает совместимость имен атрибутов и типов соответствующих доменов. Для устранения конфликтов имен атрибутов в исходных отношениях (когда совпадение имен недопустимо), а также для построения произвольных имен атрибутов результирующего отношения применяется операция переименования атрибутов. Структура результирующего отношения по определенным правилам наследует свойства структур исходных отношений.

a. Классические операции.

Объединением двух совместимых отношений R1 и R2 одинаковой размерности (Rl UNION R2) является отношение R, содержащее все элементы исходных отношений (с исключением повторений).

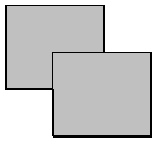


Рис.1.

Пример 1.

Пусть отношением R1 будет множество поставщиков из Лондона, а отношение R2 — множество поставщиков, которые поставляют деталь P1. Тогда отношение R обозначает поставщиков, находящихся в Лондоне, или поставщиков, выпускающих деталь P1, либо тех и других.

Отношение R1

П#	**Имя**	**Статус**	**Город_п**
S1	**Сергей**	**20**	**Москва**
S2	**Николай**	**20**	**Москва**

Отношение R2

П#	**Имя**	**Статус**	**Город_п**
S1	**Сергей**	**20**	**Москва**
S2	**Иван**	**10**	**Киев**

Отношение R(R1 UNION R2) – результирующее отношение содержит все элементы исходных отношений

П#	Имя	Статус	Город_п
S1	Сергей	20	Москва
S2	Иван	10	Киев
S4	Николай	20	Москва

Вычитание (разность) совместимых отношений R1 и R2 одинаковой размерности (Rl MINUS R2) есть отношение, тело которого состоит из множества кортежей, принадлежащих R1, но не принадлежащих отношению R2.

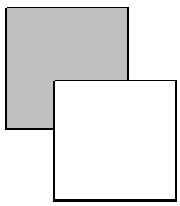


Рис.2.

Для тех же отношений R1 и R2 из примера 1 отношение R будет представлять собой множество поставщиков, находящихся в Лондоне, но не выпускающих деталь P1, т. е. R={(S4, Николай, 20, Москва)}.

П#	Имя	статус	Город_п
S4	Николай	20	Москва

Заметим, что результат операции вычитания зависит от порядка следования операндов, т. е. Rl MINUS R2 и R2 MINUS R1 — не одно и то же.

Пересечение двух совместимых отношений R1 и R2 одинаковой размерности (Rl INTERSECT R2) порождает отношение R с телом, включающим в себя кортежи, одновременно принадлежащие обоим исходным отношениям.

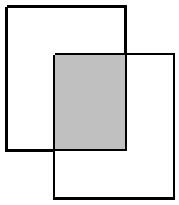


Рис.3.

Для отношений R1 и R2 результирующее отношение R будет означать всех производителей из Лондона, выпускающих деталь P1. Тело отношения R состоит из единственного элемента (S1, Сергей, 20, Москва).

П#	Имя	статус	Город_п
S1	Сергей	20	Москва

Произведение отношения R1 степени к1 и отношения R2 степени к2 (Rl TIMES R2), которые не имеют одинаковых имен атрибутов, есть такое отношение R степени (к1+к2), заголовок которого представляет сцепление заголовков отношений R1 и R2, а тело — имеет кортежи, такие, что первые к1 элементов кортежей принадлежат множеству R1, а последние к2 элементов — множеству R2. При необходимости получить произведение двух отношений, имеющих одинаковые имена одного или нескольких атрибутов, применяется операция переименования RENAME, рассматриваемая далее.

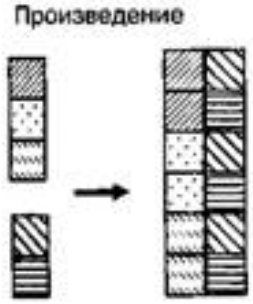


Рис.4.

В теории множеств результатом операции прямого произведения является множество, каждый элемент которого является парой элементов, первый из которых принадлежит R1, а второй — принадлежит R2. Поэтому кортежами декартова произведения бинарных отношений будут кортежи вида: ((а, б), (в, г)), где кортеж (а, б) принадлежит отношению R1, а кортеж (в, г) — принадлежит отношению R2.

Пример 2.

Пусть отношение R1 представляет собой множество номеров всех текущих поставщиков {SI, S2}, а отношение R2 — множество номеров всех текущих деталей {P1, P2 }. Результатом операции R1 TIMES R2 является множество всех пар типа «поставщик — деталь», т. е.{(S1.P1), (S1.P2), (S2.P1), (S2.P2)}.

Отношение R1

П#	Имя	Город_п
S1	Сергей	Москва
S2	Николай	Москва

Отношение R2

P#	название	тип
P1	Гайка	Каленый
P2	Угол	Твердый

Результирующее отношение R (R1 TIMES R2) – тело состоит из кортежей – множества всех пар.

П#	Имя	Город_п	P#	название	тип
S1	Сергей	Москва	P1	Гайка	Каленый
S1	Сергей	Москва	P2	Угол	Твердый
S2	Николай	Москва	P1	Гайка	Каленый
S2	Николай	Москва	P2	Угол	Твердый

b. Специальные операции.

Выборка (R WHERE f) отношения R по формуле f представляет собой новое отношение с таким же заголовком и телом, состоящим из таких кортежей отношения R, которые удовлетворяют истинности логического выражения, заданного формулой f. Для записи формулы используются операнды — имена атрибутов (или номера столбцов), константы, логические операции (AND — И, OR — ИЛИ, NOT — НЕ), операции сравнения и скобки.

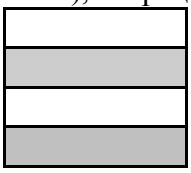


Рис.5

Пример 3. Пусть дано отношение R1, которое содержит сведения о деталях, материалах их которых они изготавливаются, вес каждой детали и города, выпускающие эти детали. Необходимо получить сведения о деталях, у которых вес не превышает 13 и получить сведения о деталях, изготовленных из каленого материала с весом не более 13.

R1

P#	название	тип	вес
P1	Гайка	Каленый	12
P2	Угол	Твердый	13
P3	Шайба	Каленый	10,5
P4	Болт	Каленый	14
P5	Гайка	Твердый	11

Результирующее отношение
R(1 WHERE вес <13)

P#	название	тип	вес
P1	Гайка	Каленый	12
P3	Шайба	Каленый	10,5
P5	Гайка	Твердый	11

Результирующее отношение
R (R1 WHERE тип = «каленый» AND вес <13)

P#	название	тип	вес
P1	Гайка	Каленый	12
P3	Шайба	Каленый	10,5

Проекция отношения A на атрибуты X, Y,..., Z (A [X, Y,..., Z]), где множество {X, Y,..., Z} является подмножеством полного списка атрибутов заголовка отношения A, представляет собой отношение с заголовком X, Y,..., Z и телом, содержащим кортежи отношения A, за исключением повторяющихся кортежей. Повторение одинаковых атрибутов в списке X, Y,..., Z запрещается.

Операция проекции допускает следующие дополнительные варианты записи:

– отсутствие списка атрибутов подразумевает указание всех атрибутов (операция тождественной проекции);

– выражение вида R[ ] означает *пустую* проекцию, результатом которой является пустое множество;

– операция проекции может применяться к произвольному отношению, в том числе и к результату выборки.

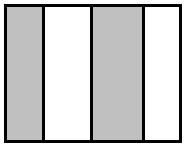


Рис. 6

Пример 4. На основании отношения R1 создана проекция R (название)

название
Гайка
Угол
Шайба
Болт

Пример 5. На основании отношения R1 создана проекция R(R1 whereТип = «каленый») [название]

название	тип
Гайка	Каленый
Шайба	Каленый
Болт	Каленый

Результатом ***деления*** отношения R1 с атрибутами A и B на отношение R2 с атрибутом B (Rl DIVIDEBY R2), где A и B простые или составные атрибуты, причем атрибут B — общий атрибут, определенный на одном и том же домене (множестве доменов составного атрибута), является отношение R с заголовком A и телом, состоящим из кортежей r таких, что в отношении R1 имеются кортежи (r, s), причем множество значений s включает множество значений атрибута B отношения R2.

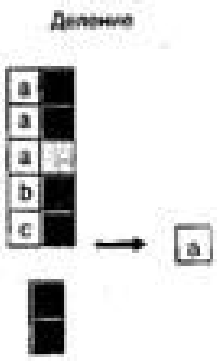


Рис. 7

Пример 6. Даны два отношения R1 и R2.

R1

P#	название	P#	тип	вес
P1	Гайка	P1	Каленый	12
P2	Угол	P2	Твердый	13
P3	Шайба	P3	Каленый	10,5
P4	Болт	P4	Каленый	14
P5	Гайка	P5	Твердый	11

R2

P#	вес
P1	12
P2	13
P3	10,5
P4	14

Результирующее отношение R(R1 DIVIDEBY R2) –вычитание из множества полей первого отношения множества полей второго отношения (одинаковые записи не дублируются)

P#	название	тип
P1	Гайка	Каленый
P2	Угол	Твердый
P3	Шайба	Каленый
P4	Болт	Каленый

Соединение C_f(Rl, R2) отношений R1 и R2 по условию, заданному формулой f, представляет собой отношение R, которое можно получить путем Декартова произведения отношений R1 и R2 с последующим применением к результату операции выборки по формуле f. Правила записи формулы f такие же, как и для операции селекции.

Другими словами, соединением отношения R1 по атрибуту A с отношением R2 по атрибуту B (отношения не имеют общих имен атрибутов) является результат выполнения операции вида:

(Rl TIMES R2) WHERE A 0 B,

где Q — логическое выражение над атрибутами, определенными на одном (нескольких — для составного атрибута) домене. Соединение C_f(Rl, R2), где формула f имеет произвольный вид (в отличие от частных случаев, рассматриваемых далее), называют также *Q-соединением.*

Важными с практической точки зрения частными случаями соединения являются эквисоединение и естественное соединение.

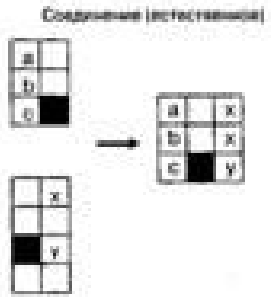


Рис.8.

Операция *эквисоединения* характеризуется тем, что формула задает равенство операндов. Приведенный выше пример демонстрирует частный случай операции эквисоединения по одному столбцу. Иногда эквисоединение двух

Пример 7. Соединение отношений R1, R2

R1

P#	название	тип	Город_П
P1	Гайка	Каленый	Киев
P2	Угол	Твердый	Киев
P3	Шайба	Каленый	Урай

R2

П#	Имя	статус	Город_Д
S1	Сергей	20	Москва
S2	Иван	10	Киев
S3	Петр	15	Урай

Результирующее отношение R(R1,R2) по атрибутам Город_П и Город_Д.

P#	название	тип	Город_П	П#	Имя	статус	Город_Д
P1	Гайка	Каленый	Киев	S2	Иван	10	Киев
P2	Угол	Твердый	Киев	S2	Иван	10	Киев
P3	Шайба	Каленый	Урай	S3	Петр	15	Урай

2.3 Лабораторная работа №3 (2 часа).

Тема: «Реляционная алгебра А Дейта и Дарвена»

Занятие посвящается проверке полученных навыков по изучаемой теме. Студентам предлагаются для решения индивидуальные варианты заданий, а также, разбившись на группы (не более 3-х человек) они должны предложить конкретную экономическую задачу и решить ее с применением полученных навыков по методам решения и с использованием компьютерной техники. Далее представитель каждой группы представляет решенную ими задачу, защищает ее, отвечая на вопросы студентов других групп (в дискуссии также принимают участие и остальные члены группы), чем осуществляется реализация такой формы, как **разбор конкретных ситуаций**. Оценка выставляется по совокупности индивидуальных заданий и разбора реальных предложенных к рассмотрению задач.

2.3.1 Цель работы: Проанализировать операции реляционной алгебры А Дейта и Дарвена и способы решения данных задач в MS Excel

2.3.2 Задачи работы:

1. Анализ операций реляционной алгебры А Дейта и Дарвена

2. Изучить способы решения данных задач в MS Excel

2.3.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Excel

2.3.4 Описание (ход) работы:

Реляционная алгебра А Дейта и Дарвена.

В качестве основы реляционных языков предложены Коддом реляционная алгебра и реляционное исчисление. Реляционная алгебра – процедурный язык, реляционное исчисление – непроцедурный язык. Реляционная алгебра определяется следующими операторами: пять основных – выборка, проекция, декартово произведение, объединение, разность и дополнительными операциями – соединение, пересечение, деление.

Операции выборка, проекция – унарные, т.к. работают с одним отношением. Другие операции работают с парами отношений и называются бинарными.

Выборка – операция над одним отношением и определяет результирующее отношение, которое содержит только те кортежи исходного отношения, которые удовлетворяют заданному предикату. Например, составьте список все студентов со средним баллом, превышающим 4 - $\sigma_{SRBAL>4}$(STUDENT)

Проекция – операция над одним отношением. Определяет новое отношение, создаваемое посредством извлечения значений указанных атрибутов и исключения из результатов строк-дубликатов. Например, создать список среднего балла студентов с указанием атрибутов, FIO, NGR, SRBAL – $\Pi_{FIO,NGR,SRBAL}$(STUDENT).

Декартово произведение – определяет новое отношение на основании отношений R и S, которое является результатом конкатенации каждого кортежа из отношения R с каждым кортежем из отношения S.(RxS).

Объединение – операция над отношениями R и S, при условии их совместимости с образованием одного отношения в результате их конкатенации с максимальным количеством кортежей и исключением дубликатов.($R \cup S$).

Разность – разность отношений R и S состоит из кортежей, которые имеются в отношении R, но отсутствуют в отношении S. R и S должны быть совместимыми отношениями.(R-S).

Соединение – определяет отношение, которое содержит кортежи из декартового произведения отношений R и S , удовлетворяющие предикату F.(R join S).

Пересечение определяет отношение, которое содержит кортежи, которые присутствуют как в отношении R , так и в отношении S.($R \cap S$).

Деление – операция, результатом которой является набор кортежей отношения R, определенных на множестве атрибутов C, которое соответствует комбинации всех кортежей отношения S.($R \div S$).

Реляционное исчисление в контексте баз данных существует в двух формах : реляционное исчисление кортежей и реляционное исчисление доменов. В реляционном исчислении кортежей задача состоит в нахождении таких кортежей, для которых предикат является истинным.

Так, для указания отношения Staff в качестве области определения переменной S используется следующая форма записи: RANGE OF S IS STAFF. Запрос: отыскать сотрудников, которые получают зарплату > 100 000, запишется следующим образом RANGE OF S IS STAFF {S| S.Salary>100 000}.

Для указания количества экземпляров, к которым должен быть применен предикат в формулах могут использоваться два типа кванторов. Квантор существования ($\exists$) используется в формуле, которая должна быть истинной хотя бы для одного экземпляра, например: RANGE OF B IS Branch $\exists$B (B.Bno=S.Bno$\wedge$B.City=”Brest”)/

Квантор общности ($\forall$) используется в выражениях, которые относятся ко всем экземплярам, например, $\forall$B (B.City$\neg$= “Minsk”.

В реляционном отношении доменов зачастую требуется проверить условие принадлежности , чтобы определить, принадлежат ли значения указанному отношению.

Например, найти имена всех менеджеров , зарплата которых >250000.

{fname, lname | ∃position, ∃salary

(Staff(lname, position,salary)∧position=”Manager”∧salary>250000}

Пример выполнения работы.

1. Схема БД состоит из четырех отношений:

Product(market, model, type)
PC(code, model, speed, ram, hd, cd, price)
Laptop(code, model, speed, ram, hd, screen, price)
Printer(code, model, color, type, price)

Используя операции реляционной алгебры построить следующие запросы:

1. Найдите номер модели, скорость и размер жесткого диска для всех ПК стоимостью менее 500 $. Вывести: model, speed, hd.
2. Получить информацию о ценах на лазерные принтеры.
3. Вывести информацию о ПК-блокнотах, имеющих стоимость ниже заданной.
4. Отыскать производителей цветных лазерных принтеров.
5. Какие модели ПК-блокнотов имеются в базе
6. Получить информацию о ПК и ПК-блокнотах, имеющих одинаковую цену.

2. Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
3. Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.

Ход работы:

1. Найдите номер модели, скорость и размер жесткого диска для всех ПК стоимостью менее 500 $. Вывести: model, speed, hd.

$\Pi_{model,\ speed,\ hd}(\sigma_{price < 500}(PC))$;

Range of P is PC
{P.model, P.speed, P.hd | P.price < 500 };

{ model, speed, hd | $\exists$ price [PC(model,speed,hd,price)^ price < 500] }

2. Получить информацию о ценах на лазерные принтеры:

$\Pi_{model,\ color,\ price}(\sigma_{type = Laser}(Printer))$;

Range of P is Printer
{ P.model, P.color, P. price | P.type = Laser };

{ model, color, price | $\exists$ type[Printer(model, color, price, type) ^ type = Laser] }

3. Вывести информацию о ПК-блокнотах, имеющих стоимость ниже заданной

$\sigma_{price < x}(Laptop)$;

Range of L is Laptop
{ L | L.price < X };

{ model, speed, ram, hd, screen, price | $\exists$ price

[Laptop (model, speed, ram, hd, screen, price) ^ price < X] }

4. *Отыскать производителей цветных лазерных принтеров*

Π_{market} (Product *Natural-join* Printer) $_{\text{Printer.Color = ”y” ^ ^Printer.Type = Laser}}$;

Range of P is Product
Range of A is Printer
{ P.market | P.type=Printer^A.Type=Laser^A.Color=“y” };
{ market | ($\exists$ type Product(type, market) ^ Product .type = Printer) ^
($\exists$ type, $\exists$ color Printer (type, color) ^ Printer .type=Laser ^ Printer .Color= “y”)};

5. *Какие модели ПК-блокнотов имеются в базе*

Π_{model} (Laptop);

Range of L is Laptop
{L.model | L };
{ model | Laptop(model, speed, ram, hd, screen, price) };

6. *Получить информацию о ПК и ПК-блокнотах, имеющих одинаковую цену*

$\sigma_{PC.Model,\ Laptop.Model}$ (PC × Laptop) $_{PC.Price = \ Laptop.Price}$;

Range of P is PC
Range of L is Laptop
{P.model, L.model | P.price = L. price };

{PC. model, Laptop. model | ($\exists$ price PC(model, price))^
($\exists$ price Laptop(model, price)) ^ PC.price = Laptop .price};

Правило *целостности сущности* выполняется для каждого из отношений и заключается в наличии первичных ключей в каждом из отношений:

- Product.Model, Printer.Code, Laptop.Code, PC.Code => PK;

Правило *ссылочной целостности* выполняется для приведённых отношений и заключается в соответствии внешних ключей (FK) каждого из отношений (Printer. Model, Laptop. Model, PC. Model => FK) значению первичного ключа (PK) в базовом отношении (Product.Model).

Задание по работе

Для определенных в предыдущей работе таблиц выполнить следующее:

Вариант 1

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Перечислить всех авторов,
2. Отыскать читателей с отчеством «Иванович»,
3. Отыскать фамилии читателей, за которыми числится книга «Война и мир» Л.Толстого,
4. Отыскать читателей (фамилия, имя , отчество, телефон, адрес), которые не взяли ни одной книги,
5. Получить список номеров читателей, которые в срок не сдали книги,
6. Получить информацию о читателях, за которыми числятся книги, изданные после 2005года,
7. Получить список книг, которые ни разу не брали читатели,
8. Составить список фамилий (читатели и авторы).

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 2

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Получить список типов карточек,
2. Составить список владельцев, срок действия карточек которых истек. (фамилия, номер карточки, тип),
3. Получить список владельцев с именем «Борис»,
4. Составить список владельцев, имеющих на счету сумму ниже 5.000бел.руб.
5. Вывести номера карточек владельца с личным номером =123456
6. Вывести номера карточек типа «Visa» с нулевой суммой на счету,
7. Получить информацию о карточках, с которыми не производились никакие операции,
8. Получить список карточек, с которыми выполнялись только операции «зачисление»

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 3

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Составить список подразделений сотрудников, в которых работают сотрудники с фамилией «Иванов»,
2. Составить список должностей,
3. Вывести фамилии сотрудников подразделения «САПР ПО»,
4. Определить каких должностей нет в подразделении «АСУ»,
5. Составить список сотрудников подразделения «САПР ПО»имеющих надбавки,
6. Получить табельные номера сотрудников в возрасте от 18 до 25 лет,
7. Получить сведения о подразделениях, в котором все сотрудники имеют надбавки,
8. Составить список сотрудников подразделения «АСУ» с указанием размера заработной платы.

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 4

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Составить список старост факультета ФИТР,
2. Составить список студентов заданной группы с указанием фамилии старосты,
3. Получить информацию: есть ли у деканов однофамильцы,
4. Отыскать номера групп, в которых учатся студенты с фамилией Петров,

5. Составить список студентов для старосты Ковальчук,
6. Вывести номера групп на факультете декана Сидорова,
7. Вывести номера групп, в которых учатся студентки с именем Каролина,
8. Вывести название факультетов, на которых есть старосты с фамилией Степанова.

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Проанализируйте возможности используемой вами СУБД . Определите, предоставляет ли она средства поддержки первичных ключей, реляционной целостности. Какие типы реляционных языков поддерживаются в этой системе

Вариант 5

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Отыскать номера машин, требующих ремонта,
2. Составить список водителей, которые ездили в командировки заданного числа,
3. Получить информацию о машинах, имеющих грузоподъемность свыше 5 тонн,
4. Составить список водителей, которые ездили в поездки с километражем более 200 км,
5. Составить список водителей, имеющих стаж свыше 10 лет,
6. Отыскать фамилии и стаж водителей, которые ездили в командировки заданного числа,
7. Отыскать водителей, которые еще не ездили в поездки с километражем свыше 100 км,
8. Составить список номеров машин , которые не были ни в одной поездке.

Для таблиц, указанных в варианте 1, написать выражения реляционной алгебры, позволяющие выполнить следующие запросы:

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 6.

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Определите дисциплины преподавателя Иванова
2. Определите даты пропусков заданного студента
3. Составить список преподавателей, которые проводили занятия заданного числа
4. Составить список студентов заданной группы
5. Составить список преподавателей, которые ведут дисциплину «Математика».
6. Отыскать группы, в которых проводит занятия преподаватель Петров.
7. Составить список студентов, которые не имеют пропусков.
8. Найти однофамильцев среди преподавателей и студентов.

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 7.

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Вывести сведения о гостиницах и их номерах стоимостью ниже 30 000руб.
2. Отыскать свободные номера в гостинице «Беларусь»
3. Отыскать постояльцев, которые проживали в гостинице «Беларусь»
4. более 10 дней.
5. Отыскать свободные однокомнатные номера.
6. Получить полные сведения о постояльцах гостиницы «Беларусь», проживающих в однокомнатных номерах.
7. Получить полные сведения обо всех номерах гостиницы «Аврора»
8. Получить полные сведения о постояльцах гостиницы «Беларусь», снимающих номер только одни сутки.
9. Перечислить все гостиницы;

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 8.

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Найдите ПК-блокноты, скорость которых меньше скорости любого из ПК. Вывести: type, model, speed
2. Найдите пары моделей PC, имеющих одинаковые скорость и RAM.
3. Найдите производителей, которые производили бы как ПК со скоростью не менее 750 МГц, так и ПК-блокноты со скоростью не менее 750 МГц. Вывести: Maker
4. Отыскать ПК и ПК-блокноты одинаковой скорости.
5. Отыскать модели цветных принтеров
6. Отыскать производителей, которые не производят принтеры.
7. Отыскать производителей, которые проиводят ПК, ПК-блокноты и принтеры.
8. Отыскать ПК-блокноты с заданной скоростью и ценой.

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 9.

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Вывести всех клиентов
2. Вывести города и клиентов этих городов
3. Вывести сотрудников, которые смогут выплатить кредит (кредит выплачивается, если сумма кредита каждого сотрудника не превосходит 10 его заработных плат);
4. Вывести клиентов, работающих в должности менеджер.
5. Составить список организаций, имеющих одинаковые должности.
6. Составить список: страна, город, организация.
7. Вывести фирмы, в которых нет сотрудников старше 45 лет.
8. Получить сведения о заработной плате клиентов из города Минска, возраст которых не превышает 35 лет.

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 10.

Написать выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Выведите список версий всех горелок типа "FiredNow".
2. Выведите имена и адреса электронной почты всех покупателей, на которых зарегистрирована горелка типа "FiredNow".
3. Выведите список имен покупателей, не ремонтировавших свои горелки.
4. Получить полную информацию о горелках покупателя с номером NN.
5. Отыскать горелки, которые не были в ремонте.
6. Отыскать покупателей, у которых стоимость ремонта горелки превышала 50 тыс.руб.
7. отыскать серийные номера горелок типа "FiredNow".
8. Определите типы горелок, которые зарегистрированы, но не были в ремонте.

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант 11.

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Выведите имена продавцов, у которых процент квоты меньше 30%.
2. Отыщите однофамильцев среди покупателей и продавцов.
3. Выведите тип промышленности и имена продавцов для заказов от компаний, находящихся в Гродно.
4. Выведите заказы каждого продавца.
5. Получить полные сведения о покупателях, сделавших заказ на сумму более 10 базовых величин.
6. Выведите имена и возраст продавцов, имеющих заказы от покупателя Иванова.
7. Составить список продавцов в возрасте до 35 лет, имеющих зарплату свыше 1млн.руб.
8. Составить список продавцов, не имеющих заказов.

- Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.
- Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.
- Определите степени исходных отношений и степени результирующих отношений.

Вариант12.

- Напишите выражения реляционной алгебры, позволяющие выполнить следующие запросы:

1. Получить номера изделий, для которых детали поставляет поставщик S1.
2. Получить номера и фамилии поставщиков, поставляющих деталь P1
3. Составить общий список городов, представленных в базе.

4. Получить номера поставщиков из Минска, которые поставляли детали в количестве большем чем 1000.

5. Выдать номера и названия изделий из города Минска.

6. Получить цвета деталей, поставляемых поставщиком S1.

7. Получить номера деталей, поставляемых для какого-либо изделия поставщиком, находящимся в том же городе, где изготавливается это изделие.

8. Составить список изделий(номер изделия, название изделия), для которых поставлялась деталь с номером P!.

9. Для перечисленных выше запросов создайте эквивалентные команды в реляционном исчислении кортежей и доменов.

10. Объясните какие правила целостности сущностей и ссылочной целостности могут быть применены к этим отношениям.

Определите степени исходных отношений и степени результирующих отношений.

2.4 Лабораторная работа №4 (2 часа).

Тема: «Реляционное исчисление»

Занятие посвящается проверке полученных навыков по изучаемой теме. Студентам предлагаются для решения индивидуальные варианты заданий, а также, разбившись на группы (не более 3-х человек) они должны предложить конкретную экономическую задачу и решить ее с применением полученных навыков по методам решения и с использованием компьютерной техники. Далее представитель каждой группы представляет решенную ими задачу, защищает ее, отвечая на вопросы студентов других групп (в дискуссии также принимают участие и остальные члены группы), чем осуществляется реализация такой формы, как **разбор конкретных ситуаций**. Оценка выставляется по совокупности индивидуальных заданий и разбора реальных предложенных к рассмотрению задач.

2.4.1 Цель работы: изучить операции реляционного исчисления

2.4.2 Задачи работы:

1. Изучение операций реляционного исчисления
2. Использование при решении задач программный продукт Microsoft Office Excel

2.4.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Excel,

2.4.4 Описание (ход) работы:

Реляционное исчисление.

На основании форм документов, описания и ограничений предметной области «Отдел реализации», а также разработанной информационно-логической модели создать логическую структуру реляционной базы данных и разработать данные контрольного примера.

Порядок создания структуры РБД:

Логическая структура реляционной базы данных определяется совокупностью логически взаимосвязанных реляционных таблиц. Каждая таблица имеет структуру, определяемую реквизитным составом одного из информационных объектов полученной информационно-логической модели. Логические связи между таблицами соответствуют структурным связям между объектами. Логическая структура реляционной базы данных (схема данных), построенная на основе информационно-логической модели предметной области «Отдел реализации» приведена на рис. 1.

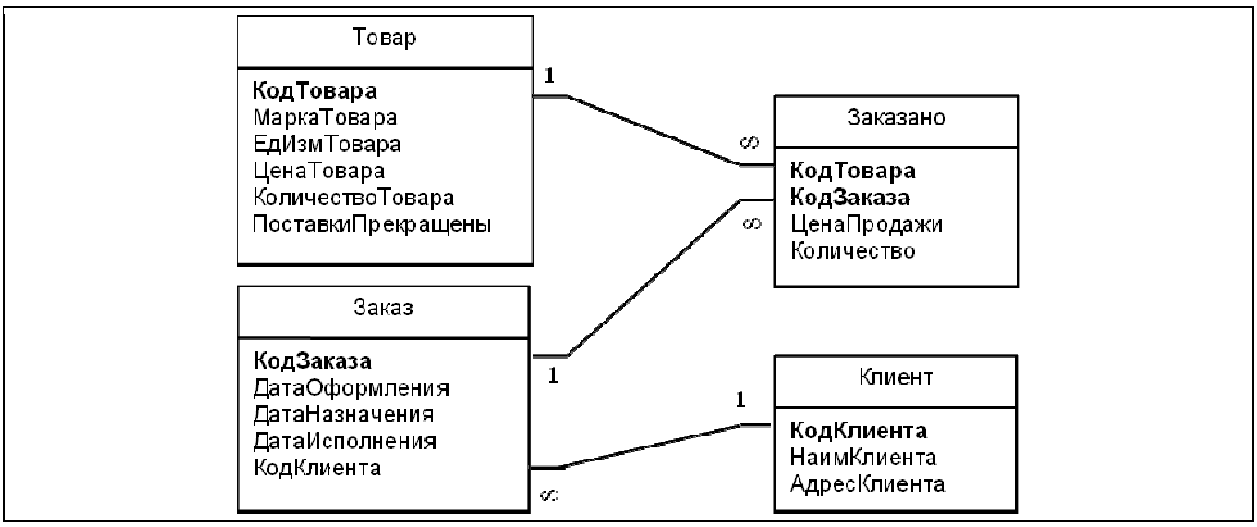


Рис. 1. Логическая структура реляционной базы данных

Логическая структура реляционных таблиц зависит от СУБД, с помощью которой будет разрабатываться база данных. В различных СУБД поддерживаются различные типы данных, существуют различные ограничения на длину имени поля и символы, используемые в имени, могут отличаться свойства полей и т. п. В таблицах 1 –4 представлена логическая структура реляционных таблиц базы данных MS Access (количество столбцов в разделе **Формат поля** может быть различным, и зависит от количества задаваемых свойств полей).

Таблица 1

Таблица: ***Клиенты***

Атрибут (поле)	Вид ключа	Формат поля			
Имя	Наименование	Тип	Размер	Обязательное поле	
КодКлиента	Код клиента	П, У	Счетчик		
НаимКлиента	Наименование клиента		Текстовый	50	Да
АдресКлиента	Адрес клиента		Текстовый	70	Да

Таблица 2

Таблица: ***Заказы***

Атрибут (поле)	Вид ключа	Формат поля					
Имя	Наименование	Тип	Размер	Знач. по умолчанию	Условие на значение	Обязат. поле	
КодЗаказа	Код заказа	П, У	Счетчик				
ДатаОформления	Дата оформления		Дата/время		Date()		Да
ДатаНазначения	Дата назначения		Дата/время			>= Date()	Да
ДатаИсполнения	Дата исполнения		Дата/время				Нет
КодКлиента	Код клиента	В	Числовой	Длинное целое			Да

Таблица 3

Таблица: ***Товары***

Атрибут (поле)	Вид ключа	Формат поля					
Имя	Наименование	Тип	Размер	Точность	Условие на значение	Знач. по умолчанию	
КодТовара	Код товара	П, У	Текстовый	6			
МаркаТовара	Марка товара		Текстовый	40			
ЕдИзмТовара	Ед. изм. товара		Текстовый	8			
ЦенаТовара	Цена		Числовой	Один. с пл. точкой	2	>=0	0
КоличествоТовара	Количество		Числовой	Целое		>=0	0
ПоставкиПрекращены	Поставки прекращены		Логический				Нет

Таблица 4

Таблица: ***Заказано***

Атрибут (поле)	Вид ключа	Формат поля				
Имя	Наименование	Тип	Размер	Точность	Условие на значение	
КодТовара	Код товара	В С, У В	Текстовый	6		
КодЗаказа	Код заказа	Числовой	Длинное целое			
ЦенаПродажи	Цена		Числовой	Один. с пл. точкой	2	>=0
Количество	Количество		Числовой	Целое		>=0

Разработка данных контрольного примера:

Основное требование к данным контрольного примера – их представительность. В данных контрольного примера необходимо отразить сведения из описания и ограничений предметной области, а также подтвердить выбор типов данных и настроек свойств полей, указанных в логической структуре реляционных таблиц.

В данных контрольного примера должны быть отражены следующие сведения из описания и ограничений предметной области «Отдел реализации»:

– каждый заказ имеет уникальный номер по предприятию;
– каждый заказ соответствует только одному клиенту;
– один и тот же клиент может оформлять много заказов на поставку товаров;
– в одном заказе может быть заказано много товаров;
– один и тот же товар может быть заказан в разных заказах;
– каждый товар идентифицируется уникальным кодом (артикулом);
– каждый товар может иметь только одну единицу измерения;

– цена товара может изменяться на основании переоценки;

– стоимость выполненных заказов изменяться не может;

– оформленные заказы должны храниться в течение нескольких лет;

– заказы могут быть исполненными или находиться в стадии исполнения (при исполнении заказа в Бланке заказа заполняется реквизит Дата исполнения).

Для обеспечения ссылочной целостности данных, в полях подчиненных таблиц, которые являются ключами связи с главными таблицами (внешними ключами), должны находиться только такие значения, которые есть в связующих полях главных таблиц.

При разработке данных контрольного примера необходимо придерживаться следующих правил:

– таблицы, находящиеся в канонической модели на 0 уровне, должны содержать не менее 3 строк с данными;

– таблицы, находящиеся в канонической модели на 1 уровне, должны содержать не менее 5 строк с данными;

– таблицы, находящиеся в канонической модели на 2 уровне, должны содержать не менее 10 строк с данными,

– таблицы, находящиеся в канонической модели на 3 уровне, должны содержать не менее 15 строк с данными,

Т. е. чем дальше от вершины модели находится таблица, тем больше строк с данными она должна содержать.

Таблица 5

Таблица: Товары (уровень 0)

Код Товара	Марка Товара	ЕдИзм Товара	Цена Товара	Количество товара	Поставки прекращены
АН4848	Ниам. товара 1	Шт.	25,55	5	
АР4848	Ниам. товара 2	Шт.	5,50	0	Да
АР5222	Ниам. товара 3	Шт.	12,50	8	
КС0588	Ниам. товара 4	Пачка	22,50	25	
ТП1025	Ниам. товара 5	Рулон	10,45	12	
ТП1026	Ниам. товара 6	Рулон	9,50	6	Да
ТП1027	Ниам. товара 7	Рулон	7,50	6	

Таблица 6

Таблица: Клиенты (уровень 0)

Код Клиента	Наим Клиента	Адрес Клиента
1	ЧП Воробьянинов К.	Г. Симферополь, ул. Литейная, 25
2	МП «Мечты сбываются»	Г. Алушта, ул. Морская, 25
3	ЧП Сидоров И. И.	П. Бузина, Симферопольский р-н, ул. Ким, 12/4
4	АО «ПырЧирПроект»	Г. Севастополь, ул. Ушакова, 35

Таблица 7

Таблица: Заказы (уровень 1)

Код Заказа	Дата Оформления	Дата Назначения	Дата Исполнения	Код Клиента

1	15/12/2002	28/12/2002	29/12/2002	1
2	18/12/2002	29/12/2002	29/12/2003	2
3	10/01/2003	22/01/2003	25/01/2003	1
4	01/02/2003	15/02/2003		3
5	05/02/2003	19/02/2003	19/02/2003	3
6	05/02/2003	15/02/2003	15/02/2003	4
7	12/02/2003	27/02/2003	27/02/2003	1
8	15/02/2003	29/02/2003		2

Таблица 8
Таблица: Заказано (уровень 2)

Код Заказа	**Код Товара**	Цена Продажи	Количество
1	AP5222	12,15	5
1	ТП1025	10,10	2
1	AP4848	5,20	3
1	AH4848	25,05	4
2	AP5222	12,15	3
3	ТП1026	9,50	2
3	AH4848	25,55	1
4	ТП1027	7,50	25
5	AP5222	12,50	5
5	ТП1025	10,45	4
5	AH4848	25,55	6
6	ТП1027	7,50	25
6	AP5222	12,50	5
7	AH4848	25,55	12
7	AP5222	12,50	15
7	ТП1025	10,45	4
8	AP4848	5,50	7
8	КС0588	22,50	2

2.5 Лабораторная работа №5 (2 часа).

Тема: «Основы проектирования баз данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.5.1 Цель работы: изучить и проанализировать основы проектирования баз данных

2.5.2 Задачи работы:

1. Изучение основ проектирования баз данных
2. Анализ персональных баз данных

2.5.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.5.4 Описание (ход) работы:

Основы проектирования баз данных.

Краткие сведения о программе

Поскольку база данных - это компьютерный эквивалент организованного списка информации, то ее назначение состоит в возможности быстро получать из нее точную информацию автоматизированным способом. Одним из самых популярных программных продуктов, обеспечивающих функции обработки, просмотра, поиска больших объемов данных является СУБД Access. Приложение Microsoft Access 2003 – система управления реляционными базами данных, предназначенная для работы на автономном ПК или в локальной вычислительной сети под управлением Microsoft Windows. Все преимущества Windows доступны в Access (например, обмен данными с другими приложениями Windows), базовые объекты интерфейса - **меню**, **панели инструментов**, **диалоговые окна** - работают точно так же, как в других продуктах Office или других приложениях для Microsoft Windows. Средства Access 2003 предоставляют пользователю возможность выполнять следующие операции:

- Проектирование базовых объектов ИС – двумерных таблиц с разными типами данных (разработка макетов таблиц).
- Установление связей между таблицами с поддержкой целостности данных.
- Ввод информации в таблицы, хранение, просмотр, корректировка с использованием различных средств контроля информации, индексирования таблиц.
- Создание и использование форм, запросов и отчетов для обработки данных таблиц в соответствии с требованиями пользователя.
- Подготовка собственных приложений (программ) на языке VBA (Visual Basic for Applications).

- Таблицу Access можно связать с данными, находящимися на другом компьютере или сервере, а также использовать таблицу, созданную в СУБД Paradox или Dbase. Данные Access легко комбинируются с данными Excel.

В СУБД Access предусмотрено много дополнительных возможностей. Удобные и гибкие средства визуального проектирования объектов с помощью **Мастеров** позволяют практически неподготовленному пользователю довольно быстро создать полноценную ИС на уровне таблиц, форм, запросов и отчетов. Система Access содержит набор инструментов для управления базами данных, включающий **Конструкторы** таблиц, форм, запросов и отчетов. Использование **Макросов** позволяет автоматизировать многие процессы программирования для разработки приложений пользователя. Для обеспечения взаимодействия Access с другими приложениями используются такие возможности языка программирования Си, как функции и обращения к Windows API (Application Programming Interface – интерфейс прикладных программ Windows). Access также располагает средствами для облегчения работы в Internet и создания приложений для Web.

Объектом обработки MS Access является файл базы данных, имеющий произвольное имя и расширение .mdb. Этот файл содержит основные объекты MS Access:

- **таблицы** - основа БД, столбцы которых называется полями, а строки – записями;
- **формы** – окна, позволяющие более наглядно отобразить информацию, содержащуюся в одной записи БД, позволяют упростить операции ввода и просмотра данных;
- **запросы** - предназначены для поиска и получения информации из БД по различным критериям;
- **макросы** – последовательности макрокоманд для расширения возможностей СУБД. С их помощью можно изменять ход выполнения приложения, открывать, фильтровать и изменять данные в формах и отчетах, выполнять запросы и создавать новые таблицы. **Макросы** могут использоваться для того, чтобы сделать часто повторяющиеся действия доступными в виде командных кнопок на формах, которые помогают менее опытным пользователям работать с вашей базой данных;
- **Модули** - это программы на Microsoft Visual Basic for Applications (VBA). В то время как макросы могут автоматизировать многие действия, VBA может использоваться для выполнения задач, которые слишком сложны для выполнения с помощью макросов.
- **Отчеты** отображают информацию из таблиц на экране компьютера или на бумаге в красиво отформатированном виде. **Отчет** может включать элементы информации, выбранные из нескольких таблиц и запросов, значения, вычисленные на основе информации из базы данных, и форматирующие элементы, такие, как заголовки, колонтитулы, названия и «шапки».
- **страницы** показывают все ярлыки страниц доступа к данным Internet и Intranet, дают возможность ввода, редактирования, просмотра и манипулирования данными из сети.

Каждый объект MS Access имеет имя, длина которого – не более 64 произвольных символов (за исключением точки и некоторых служебных символов). Любой объект БД (таблицу, форму, запрос, отчет) можно создать либо вручную в режиме **конструктора**, либо с помощью **мастера**.

В состав системы MS Access разработчики включили несколько готовых БД, которые находятся в папке Programs Files\MS Office\Office\Samples. Некоторые из них, например, Борей.mdb (Access 2000) или Acwzdat.mdb (Access 2003) можно использовать для знакомства с приемами работы с БД.

Основные операции с файлами БД (Создание, открытие, сохранение, закрытие) в Access, также как и в других Windows-приложениях выполняются с помощью стандартных

диалоговых окон-файлеров, которые открываются командами меню **Файл**, или соответствующими кнопками. Настройка инструментария системы, выбор режима просмотра данных осуществляется с помощью команд пункта меню **Вид.** Настройка вида экрана, клавиатуры, рабочего каталога, порядка сортировки БД, вызов служебных программ и др. производится командами меню **Сервис**. При возникновении затруднений в работе с программой можно обратиться к справочной системе Access, которая имеет удобные и простые в использовании содержание, предметный указатель, систему поиска, журнал хронологии и закладки. Для получения полной справки используется пункт меню **Справка** или кнопка  на панели инструментов. **Контекстно-зависимая справка** вызывается клавишей **F1**. Access, также использует новое средство – **Помощник**, который отвечает на вопросы, выдает советы и справки об особенностях используемой программы.

Запуск программы. Создание БД

Запуск MS Access осуществляется:

- Через ярлык на рабочем столе;
- Кнопкой **Пуск/Программы/MS Office/MS Access.**

При открытии системы появляется диалоговое окно, в котором предлагается выбор: создать новую БД самостоятельно (пустую неструктурированную БД), использовать мастер или проект для разработки БД (пустую структурированную БД на основе шаблона) или открыть уже существующую ранее созданную БД, выбрав нужный файл из списка. В Access 2003 **область задач** у правой границы окна предлагает приступить к работе в программе в одном из перечисленных ниже режимов. Если программа Access была открыта ранее для работы с другой БД, для создания нового файла БД можно воспользоваться кнопкой  на панели инструментов или командой меню **Файл/Создать,** выбрав на вкладке **Общие** режим **База данных** (в Access 2003 командой **Создать Файл** перейти в область задач **Создание файлов**, выбрать команду **Новая база данных**).

Microsoft Office Access 2003 включает набор **мастеров**, которые помогают быстро и просто создавать **базы данных** и другие объекты, такие, как **таблицы**, **запросы**, **формы** и **отчеты**.

Разработка таблицы. Типы данных в Access

Таблицы являются основой БД и состоят из **полей** и **записей**. Каждая запись таблицы содержит всю необходимую информацию об отдельном элементе БД. При разработке структуры таблицы для каждого поля задается уникальное имя длиной не более 64 символов (оно обычно совпадает с названием атрибута, значения которого будут храниться в этом поле), определяется тип данных поля. Значение типа поля может быть задано только в режиме конструктора. Ниже в таблице представлены типы данных Access и их описание.

Таблица 1.2. Типы данных в Access

Тип данных	Описание
Текстовый (Значение по умолчанию)	Текст или числа, не требующие проведения расчетов, например, номера телефонов (до 255 знаков)
Числовой	Числовые данные различных форматов, используемые для проведения расчетов
Дата/время	Для хранения информации о дате и времени с 100 по 9999 год включительно
Денежный	Денежные значения и числовые данные, используемые в математических расчетах, проводящихся с точностью до 15 знаков в целой и до 4 знаков в дробной части
Поле MEMO	Для хранения комментариев; до 65535

Счетчик	Специальное числовое поле, в котором Access автоматически присваивает уникальный порядковый номер каждой записи. Значения полей типа счетчика обновлять нельзя.
Логический	Может иметь только одно из двух возможных значений (Да/Нет, True/False)
Поле объекта OLE	Объект (например, электронная таблица Microsoft Excel, документ Microsoft Word, рисунок, звукозапись или другие данные в двоичном формате), связанный или внедренный в таблицу Access.
Гиперссылка	Строка, состоящая из букв и цифр и представляющая адрес гиперссылки. Адрес гиперссылки может состоять максимум из трех частей: текст, выводимый в поле или в элементе управления; путь к файлу (в формате UNC) или к странице (адрес URL). Чтобы вставить адрес гиперссылки в поле или в элемент управления, выполните команду **Вставка, Гиперссылка**
Мастер подстановок	Создает поле, в котором предлагается выбор значений из списка или из поля со списком, содержащего набор постоянных значений или значений из другой таблицы. Это в действительности не тип поля, Access способ хранения поля

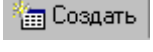
Создать таблицу в Access можно несколькими способами:

- В режиме **конструктора таблиц**, который предусматривает определение всех параметров структуры (макета) таблицы вручную;
- С помощью **мастера таблиц**. Мастер позволяет выбрать поля для данной таблицы из множества определенных ранее таблиц в одной из категорий – Деловые (контакты, клиенты, заказы…) или Личные (личное имущество, гости, рецепты…);
- В **режиме таблицы путем ввода данных** непосредственно в пустую таблицу. При сохранении таблицы Access анализирует данные и каждому полю присваивает необходимый тип данных и формат;
- С помощью **импортирования** – импорт данных и объектов из внешнего файла в текущую БД;
- С помощью **связывания** с другими таблицами – создание таблиц в текущей БД, связанных с таблицами внешнего файла;
- Использованием **мастера баз данных** для создания всей БД, содержащей все требуемые отчеты, таблицы, формы за одну операцию. Мастер БД создает новую БД целиком, его нельзя использовать для добавления новых таблиц, форм и отчетов в уже существующую БД.

Независимо от способа, использованного для создания таблицы, пользователь может использовать режим конструктора для изменения макета таблицы – добавления или удаления полей, установления ограничений на ввод значений, для установки значений по умолчанию.

Так как только режим конструктора позволяет пользователю сразу задать ту структуру таблицы, которая ему нужна, для разработки таблиц используется именно этот способ.

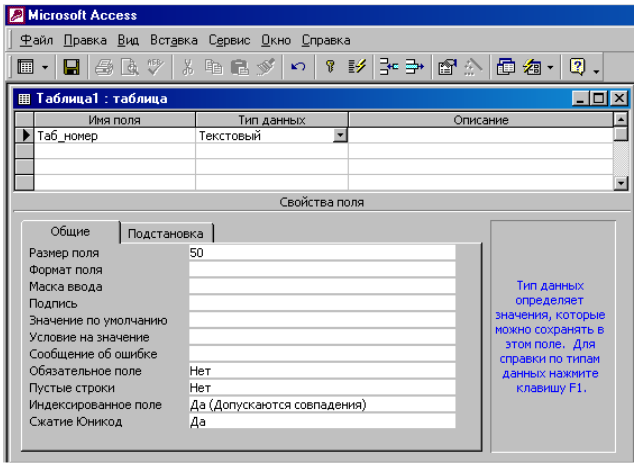
Вызов конструктора таблиц можно осуществить:

- объект **таблицы** (в окне открытой БД) – **Создать таблицу в режиме конструктора;**
- кнопка 

(в окне открытой БД) – **Конструктор**;
- меню **Вставка/Таблица** – **Конструктор**;

В режиме **конструктора** таблица содержит три колонки: **Имена полей**, **Тип данных**, **Описание. Имена полей** могут содержать русские, латинские буквы, цифры без пробелов. После присвоения имени задается **Тип поля** (по умолчанию программой выбирается Текстовый). **Описание** полей не обязательно, оно добавляет наглядности БД. Перемещение,

удаление, добавление полей осуществляется с помощью мыши. Для перемещения поля его следует выделить (щелкнуть левой кнопкой мыши слева от имени перемещаемого поля в области маркировки записи) и «перетащить» в нужное место. Для удаления поля его следует выделить (выделить всю строку, соответствующую этому полю) и удалить клавишей **Delete** (или выполнить команду **Правка/Удалить**). Чтобы выделить несколько полей, следует выделение выполнять совместно с клавишей **Shift** (для смежных полей) или **Ctrl** (если поля расположены не подряд). Для добавления поля используется команда **Вставка/Строки**. Новая строка будет вставлена над строкой, в которой находится курсор. Удаление и вставку полей можно выполнять через контекстное меню, которое открывается правой кнопкой мыши или с помощью кнопок на панели инструментов  - добавить строки,  - удалить строки.



Прежде чем сохранить таблицу в файле БД, следует определить первичный ключ (для таблиц многотабличной БД). Для установки ключевого поля нужно предварительно его выделить и выполнить команду **Правка/Ключевое** поле в главном меню или команду **Ключевое поле** в контекстном меню, можно воспользоваться кнопкой  на панели инструментов. Если поле определено ключевым по ошибке, можно использовать команду **Правка/Отменить** ключевое поле или повторно щелкнуть на кнопке  .

В нижней части окна конструктора указываются **свойства полей**. Для их определения нужно:

- установить курсор в верхней части окна в нужное поле;
- перейти в нижнюю часть (**Свойства поля**) клавишей **F6** или мышью;
- установить курсор в строке нужного параметра;
- ввести с клавиатуры значение этого параметра или выбрать его из раскрывающегося списка. Заполнение некоторых свойств можно выполнить с помощью вспомогательного окна (мастера) **Построителя выражений**, вызываемого кнопкой  , расположенной справа от ячейки соответствующего свойства.

В окне **Свойства поля** определяются следующие параметры:

- **Размер поля** – максимальный размер данных в поле. Для текстовых значений по умолчанию устанавливается – 50, для числовых – Длинное целое (4 байта).

Основные размеры полей для числового типа данных приведены в таблице 1.3.

Таблица 1.3.

Тип	**Размер**
Байт (1 байт)	Целые числа от 0 до 255
Целое (2 байта)	Целые числа от -32768 до +32767
Длинное целое (4 байта)	Целые числа от –2 147 483 648 до +2 147 483 647
Одинарное с плавающей точкой (4 байта)	С точностью до 6 знаков от $-3{,}4 \times 10^{38}$ до $+3{,}4 \times 10^{38}$
Двойное с плавающей точкой (8 байт)	С точностью до 10 знаков от $-1{,}797 \times 10^{308}$ до $+1{,}797 \times 10^{308}$

- **Формат поля** – задает формат представления данных при выводе на экран или печать. Для типов данных **Числовой**, **Денежный** и **Счетчик** существует набор форматов:
  - **Стандартный** – отсутствуют разделители тысяч и знаки денежных единиц, число десятичных знаков зависит от точности данных (устанавливается по умолчанию);
  - **Денежный** или **евро** – символы валют и два знака после запятой;
  - **Фиксированный** – два знака после запятой;
  - **С разделителями разрядов** – разделители тысяч и два знака после запятой;
  - **Процентный** – два знака после запятой и символ % (значение увеличивается в 100 раз);
  - **Экспоненциальный** – запись числа в экспоненциальной форме (например, 3.46E+7 – это экспоненциальная форма числа 3,46 x10^7 или 34,6 x10^6;

Для типов **Дата/Время** существует набор форматов:

  - Полный формат даты (15.06.99 07:15:12 PM)
  - Длинный формат даты (Вторник, 20 сентября 2005);
  - Средний формат даты (20-сен-05)
  - Краткий формат даты (20.09.05)
  - Длинный формат времени (07:15:12 PM);
  - Средний формат времени (07:15 PM);
  - Краткий формат времени (07:15).

Для **Логического** типа определяются форматы: Да/Нет, Истина/Ложь, Вкл/Выкл.

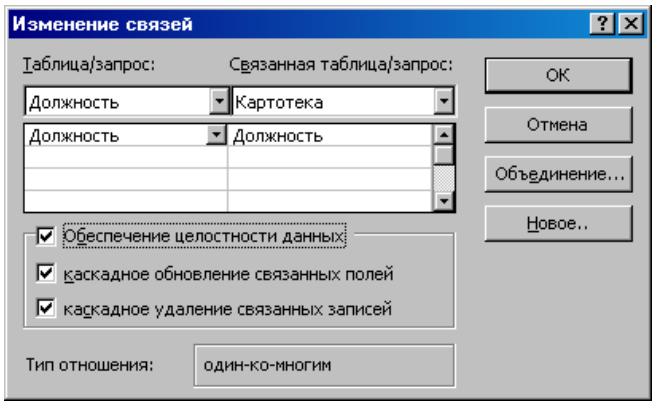
- **Маска ввода** – автоматически изображает неизменяемые символы поля. При вводе данных в поле, заданное маской, достаточно заполнить пустые позиции. Маску ввода можно ввести с клавиатуры или использовать Мастер масок(...). Для ввода маски с клавиатуры используются следующие обязательные символы:
  - 0 – цифра
  - L – буква
  - A – буква или цифра
  - @ - любой символ или пробел
  - <(>) – преобразует все символы справа к нижнему (верхнему) регистру
  - ! – маску следует заполнять справа налево.
- **Подпись поля** – определяет подпись для использования в формах и отчетах, если она отличается от имени поля.
- **Значение по умолчанию** – значение, которым программа автоматически заполняет данное поле во всех новых записях таблицы.
- **Условие на значение** – указывает, каким условиям должны удовлетворять значения, вводимые в данное поле. Оно задается выражением, состоящим из операторов сравнения и значений, используемых для сравнения (операндов). При вводе данных производится автоматическая проверка (контроль) соответствия данных указанным типам и проверка выполнения заданных условий. Условие задается либо с клавиатуры, либо с помощью построителя выражений (...).
- **Сообщение об ошибке** – позволяет задать текст, выводимый на экран, если значение не удовлетворяет **Условию на значение**.
- **Обязательное поле** – определяет, может ли это поле остаться незаполненным при вводе данных.
- **Индексированное поле** – задает построение индекса для полей с типом данных **Текстовый**, **Числовой**, **Денежный**, **Дата/Время** и **Счетчик** для ускорения выполнения запросов, поиска и сортировки.

- **Смарт-тэги.** Когда создается таблица в Access 2003, можно к каждому из полей применить несколько **смарт-тэгов**. Когда информация из такого поля отображается в таблице, форме или запросе, и вы подводите к ней указатель мыши, отображается кнопка со списком **Действия для смарт-тэгов** (**Smart Tag Action**), и вы можете выполнить некоторые действия, связанные с данным типом информации (аналогично контекстному меню).

Ввод данных в таблицу осуществляется с клавиатуры, либо методом **подстановки** значений из другой таблицы (вкладка **Подстановка** в окне **Свойства поля**). Данные в текущую таблицу можно также **скопировать** из таблицы другого файла БД командами меню **Правка/Копировать** и **Правка/Вставить** или соответствующими кнопками на панели инструментов.

Установление связей между таблицами

Все таблицы БД могут функционировать самостоятельно, т.е. отдельно друг от друга. Но связи между ними помогают работать с БД более эффективно, предоставлять пользователю сведения в **большем** объеме и **более** конкретные. Связь между двумя таблицами устанавливается обычно по одноименным полям, совпадающим по **типу** и **размеру** (совпадение **имен** полей связи необязательно, но желательно). В одной из них поле связи **обязательно** должно быть **ключевым**. **Таблица, в которой поле связи является первичным ключом, называется *главной*, а ее первичный ключ – *уникальным ключом*. Таблица, в которой поле связи не является ключевым, называется *подчиненной*, а поле связи в ней – *внешним ключом*.** Установить, посмотреть, отредактировать Связи между таблицами можно в окне **Схемы данных**, которое открывается командой меню **Сервис/Схема данных** или кнопкой - **схема данных**. При первоначальной установке связей открывается окно **Добавление Таблицы**, из которого нужные таблицы переносятся в окно схемы данных.

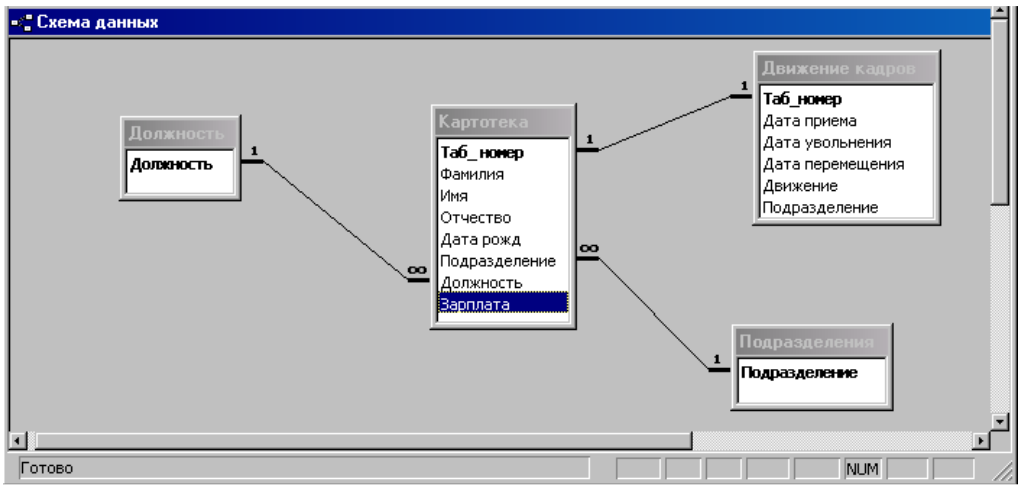


Для установки связи в одной из связываемых таблиц нужно выделить поле связи и совместить (перетащить) его с соответствующим полем второй таблицы. В открывающемся диалоге **Изменение Связей** (его можно также открыть двойным щелчком на линии связи) нужно установить флажок у режима **Обеспечение целостности данных** и связанных с ним режимов **Каскадное обновление связанных полей** и **Каскадное удаление связанных записей**. **Целостность данных** – это набор правил, защищающих данные от случайных изменений или удалений с помощью механизма поддержки корректности связей между связанными таблицами.

Контролировать целостность данных можно, если выполнены некоторые условия:

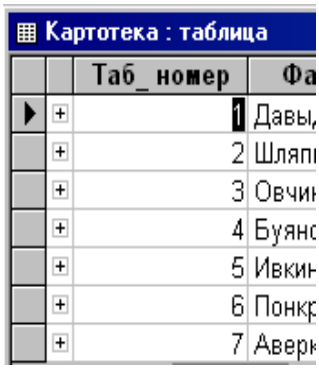
- поле связи является ключевым хотя бы в одной таблице;
- связанные поля имеют одинаковый тип данных. Поле **счетчика** является исключением, оно может быть связано с числовым полем, если для него определен тип **Длинное целое**;
- связываемые таблицы принадлежат одной БД Access.

Система автоматически определяет тип устанавливаемой связи: **один-к-одному**, **один-ко-многим**, **многие-ко-многим** (в нижней части диалога). На линии связи в окне схемы данных появляются значки «**1**» и «∞», обозначающие отношение «**один**» или «**многие**». Пример схемы данных изображен на рисунке.



Тип создаваемой связи зависит от полей, для которых определяется связь:

- связь **Один-ко-многим** создается в том случае, когда только в одной из связываемых таблиц поле связи является ключевым или имеет уникальный индекс, т.е. значения в нем не повторяются;
- связь **Один-к-одному** создается в том случае, когда оба связываемых поля являются ключевыми или имеют уникальные индексы;
- связь **Многие-ко-многим** фактически представляет две связи типа **один-ко-многим** через третью таблицу, ключ которой состоит, по крайней мере, из двух полей, общих для двух других таблиц.



Если связь между таблицами системой определена, то при просмотре главной таблицы слева от первого поля радом с полосой выделения появится колонка со знаками «+». Щелчок на «+» откроет подчиненную таблицу.

Поиск информации в БД

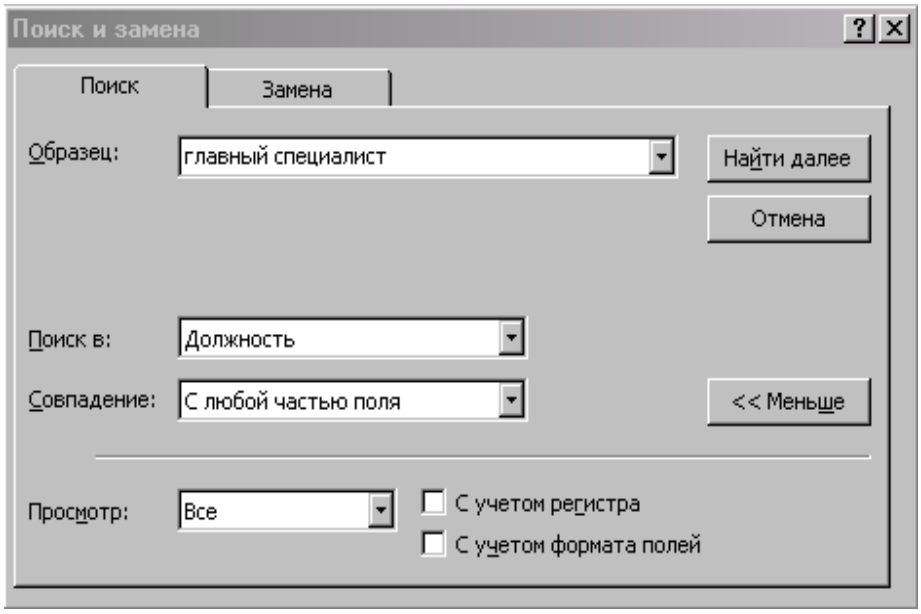
Одной из основных функций СУБД является **поиск** информации в БД и ее предоставление для просмотра или печати – **выборка**. В зависимости от информационных потребностей пользователя можно использовать простые приемы поиска данных или более сложные, конкретизирующие условия отбора данных с помощью непростых выражений. Microsoft Office Access 2003 предоставляет различные инструменты, которые можно использовать для организации отображения информации из базы данных и для поиска конкретных частей этой информации.

Классификацию средств поиска в Access можно представить в виде таблицы-схемы.

Таблица 1.4.

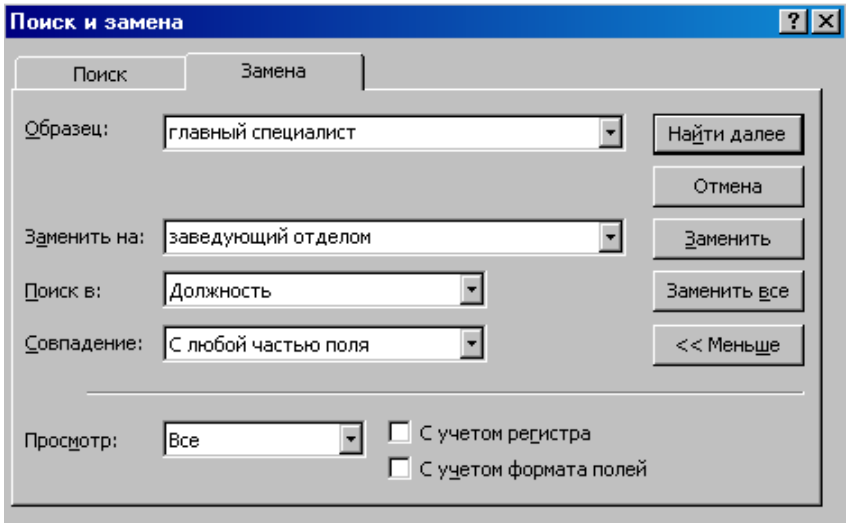
<table>
<tr><td colspan="4">Поиск данных в БД</td></tr>
<tr><td colspan="2">Простые средства поиска</td><td colspan="2">Запросы</td></tr>
<tr><td>Найти
Заменить
Сортировка</td><td>Фильтры</td><td>На выборку</td><td>На изменение</td></tr>
<tr><td rowspan="4"></td><td>по выделенному</td><td>итоговые</td><td>на обновление</td></tr>
<tr><td>по форме</td><td>параметрические</td><td>на удаление</td></tr>
<tr><td>по вводу</td><td rowspan="2">перекрестные</td><td>на добавление</td></tr>
<tr><td>расширенный фильтр</td><td>на создание таблицы</td></tr>
</table>

К простейшим видам поиска относятся операции с использованием команд **Правка/Найти** и **Правка/Заменить**. Диалог **Найти** (**Заменить**) можно открыть также кнопкой  . В появившемся окне нужно ввести образец искомых данных (значение, фраза или выражение), задать режимы поиска, щелкнуть на кнопке Найти далее.



Если значение найдено системой, курсор остановится в таблице в соответствующей ячейке, при перемещении курсора он будет останавливаться в ячейках со значениями, совпадающими с образцом поиска. Если необходимо большое число одинаковых данных заменить новым значением, используется команда **Правка/Заменить**.

В диалоге замены вводится образец значения, которое надо найти и значение, на которое нужно заменить найденное, устанавливаются режимы поиска и замены. Для выполнения поиска используется кнопка Найти далее, для замены кнопка Заменить или Заменить все.



В условиях поиска могут быть использованы операции сравнения (>,<,>=,<=,=,<>), а также подстановочные символы:

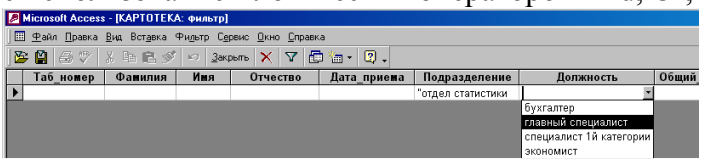
* - любая цифра или символ;
? – любой текстовый символ;
[ ] – любой один символ из заключенных в скобки;
! - любой один символ, кроме заключенных в скобки;
\- - любой символ из диапазона;
- любая цифра.

Использование фильтров. Сортировка

Фильтр используется для отображения в окне только тех записей БД, которые удовлетворяют условиям пользователя. Фильтры просты в использовании и представляют собой одноразовые запросы без имени. Их можно применять к таблице, запросу или форме. В фильтре всегда отображаются все поля и используются данные только одной таблицы. В СУБД Access используются фильтры четырех видов:

Фильтр по выделенному фрагменту – оставляет в таблице записи, совпадающие по значению какого-либо поля. В таблице предварительно курсором помечается значение отбора (в любой строке).

Фильтр по форме (**изменение фильтра**) – представляет таблицу в виде пустой табличной строки с пиктограммой списка в каждом поле, где можно задавать критерии отбора с использованием логических операторов And, Or, Not.



Фильтр по вводу - устанавливается через контекстное меню нужного поля таблицы. Позволяет найти записи, удовлетворяющие нескольким условиям одновременно.

Расширенный фильтр – позволяет с помощью специального бланка фильтра, в котором задаются условия отбора, в открытой форме или таблице выделять подмножество записей, отвечающих заданным условиям. В бланке фильтра можно задать порядок сортировки для одного или нескольких полей.

Фильтры устанавливаются командой **Записи/Фильтр…** или с помощью кнопок на панели инструментов: - **фильтр по выделенному**, - **изменить фильтр**, - **удалить фильтр**.

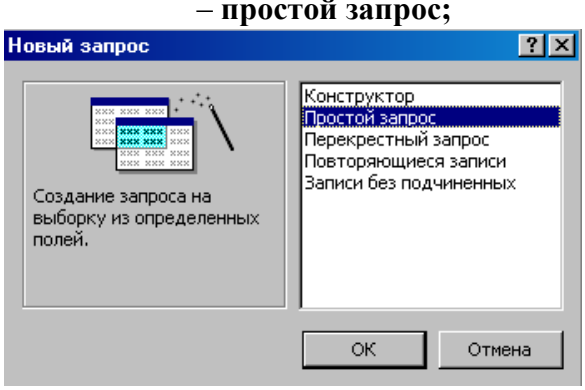
Сортировка – определяет порядок вывода записей в таблице или запросе. Выполнить сортировку данных можно с помощью команды меню **Записи/Сортировка** или кнопками , предварительно установив курсор в поле ключа сортировки.

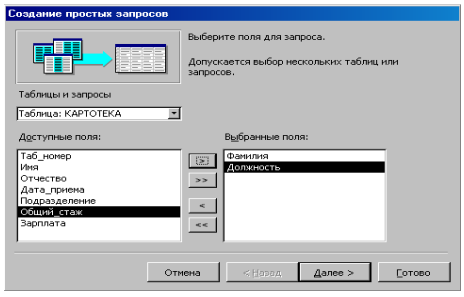
Запросы в Access

Запрос к БД является мощным средством Access для поиска, выборки, просмотра, анализа и изменения данных одной или нескольких таблиц. Он представляет собой сформулированную информационную потребность. В работе с запросом выделяется два этапа: создание запроса (проектирование) и выполнение. В результате выполнения запроса из БД (из одной или нескольких таблиц) выбирается информация в соответствии с критериями запроса. Запросы можно создавать самостоятельно – в режиме **Конструктора** или – с помощью **Мастера запросов**, который автоматически выполняет основные действия в ответ на задаваемые пользователю вопросы. Непрофессиональному пользователю рекомендуется формировать запрос с помощью **Мастера**, а критерии поиска, выборки, изменения данных добавлять в режиме **Конструктора запросов**.

Мастер запросов можно открыть из окна БД, выделив объект **Запросы**

- двойным щелчком мыши на режиме **Создание запроса с помощью мастера;**
- или щелкнув на кнопке Создать, и выбрав из предложенного списка режимов – **простой запрос;**





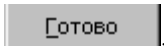
- выполнив команду меню **Вставка/Запрос/Простой запрос**.

В окне мастера запросов нужно:

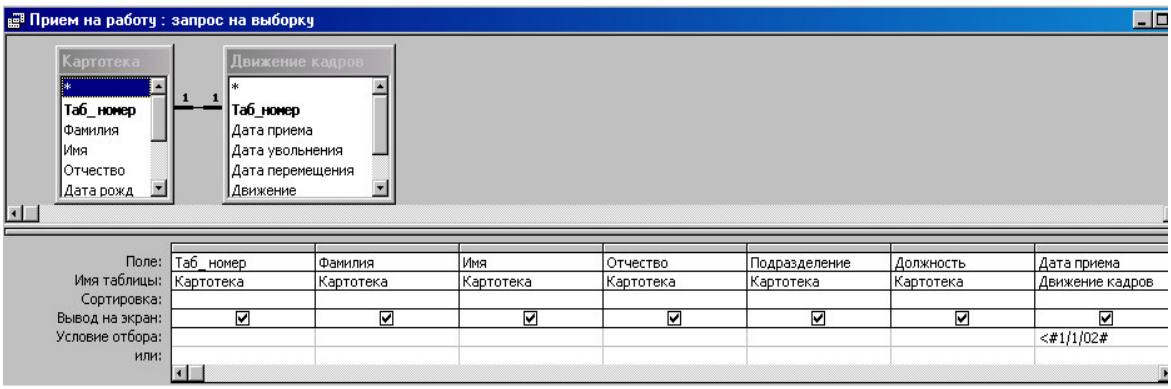
- выбрать объект (таблицу или запрос), на основе которого создается запрос;
- из перечисленного списка полей выбрать те, которые содержат данные для просмотра или изменения (т.е. поля, которые должны выводится на экран в результате выполнения запроса).

Для перехода к каждому следующему окну мастера запросов используется кнопка 

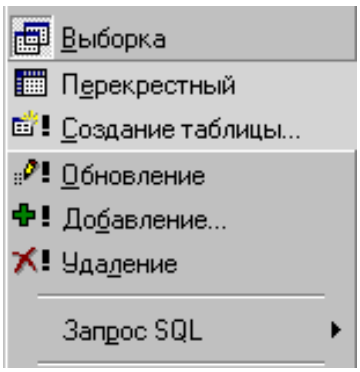
. Для просмотра результатов запроса на экране используется кнопка 

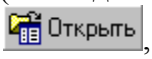
. Для определения условий отбора записей можно перейти в режим Конструктора запроса. Для этого после задания имени запроса, нужно установить переключатель в позицию режима **Изменить макет запроса**, щелкнуть на кнопке 

. В конструктор можно перейти из режима просмотра кнопкой  или командой меню **Вид/Конструктор.**



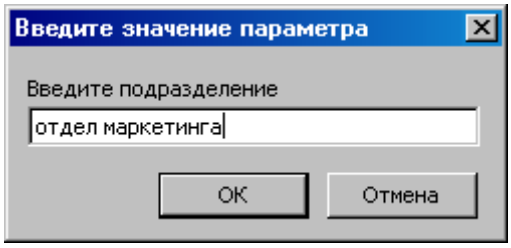
Окно конструктора запроса состоит из двух областей. В верхней области система отображает структуру таблиц (запросов) из которых были выбраны поля в создаваемый запрос, указывая связи между таблицами. В нижней части окна формируется структура текущего запроса: в столбцах отображаются имена выбранных полей (строка **Поле**), имена таблиц, из которых были выбраны поля (строка **Имя таблицы**), условия сортировки записей (строка **Сортировка**), режим отображения полей на экране (строка **Вывод на экран**, помеченные галочкой поля будут выводиться на экран в результате выполнения запроса), критерии поиска и выборки записей (Строки **Условие отбора**, **или**). В качестве **Условия отбора** могут быть выражения, даты, числовые значения, текст, которые либо вводятся с клавиатуры, либо с помощью **Построителя выражений** -  , либо с помощью команды контекстного меню **Построить**. Все критерии отбора, указанные в одной строке должны выполняться совместно (логическое объединение And). Если критерии отбора указываются в разных строках “**Условие отбора”** и “**Или”,** предусматривается выбор между этими условиями (логический выбор Or).



Поскольку в Access существует несколько видов запросов (на выборку, на создание, на обновление, перекрестный, итоговый и др.) при создании запроса необходимо указать его вид. По умолчанию формируется запрос **на выборку**. Тип запроса может быть преобразован в любой другой командами меню **Запрос** или кнопкой  . Запрос запускается на выполнение из окна конструктора - кнопкой  (команда меню **Запрос/Запуск**), из окна БД - кнопкой 

, или двойным щелчком мыши на имени запроса. При выполнении запроса **на выборку** система извлекает записи из таблиц и формирует результирующий набор данных, который выглядит как таблица, но по сути не является ею. Такой набор данных является динамическим и не хранится в БД. При сохранении запроса сохраняется только его структура: перечень таблиц, состав полей, условия отбора, тип

запроса. Такой способ организации обеспечивает возможность многократного выполнения запроса с учетом любых изменений в базовых таблицах.



ПАРАМЕТРИЧЕСКИЕ ЗАПРОСЫ . Для того чтобы сделать запрос на выборку более универсальным, можно вместо конкретного значения отбора включить в запрос **параметр**, который обеспечит диалоговый режим выполнения запроса. Для этого в строку **Условие отбора** в конструкторе запроса вводится фраза в квадратных скобках [ ], которая будет выводиться на экран в качестве подсказки пользователю в процессе выполнения запроса, например [Введите фамилию], [Введите должность сотрудника].

Аналогичные параметры могут быть заданы для нескольких полей запроса одновременно, причем имя каждого параметра должно быть уникальным. Для корректировки параметров можно использовать команду меню **Запрос/Параметры**.

ИТОГОВЫЕ ЗАПРОСЫ . Иногда при выполнении поиска и отбора данных необходимо найти какое-либо итоговое значение, например, сумму значений или среднее значение в поле. Запросы, выполняющие вычисления с группой записей, называются **итоговыми**. Для их создания в режиме конструктора нужно добавить новую строку **Групповая операция** командой меню **Вид/Групповые операции** или кнопкой Σ. В этой строке все поля запроса будут заполнены значением **Группировка**. Открыв список группировки в поле, для которого рассчитывается итоговое значение (это должно быть числовое поле), нужно выбрать одну из возможных функций (Sum, Avg, Min, Max, Count и др.).

ПЕРЕКРЕСТНЫЕ ЗАПРОСЫ – особый тип итоговых запросов, представляющих результаты поиска в виде матрицы. **Перекрестный запрос** состоит из трех полей, из которых одно определяет заголовки строк, второе – заголовки столбцов, третье (обязательно должно быть числовым) – определяет значения, по которым рассчитываются итоги. Если в перекрестный запрос выбираются поля из разных таблиц и запросов, то предварительно нужно создать простой запрос на выборку, содержащий данные поля, а затем на его основе создать перекрестный запрос. Пример перекрестного запроса для расчета **средней зарплаты** по всем **подразделениям** организации, для каждой **должности** сотрудников приведен ниже.

Картотека_перекрестный : перекрестный запрос

Подразделение	Итоговое значение	Вед_ Спец	Главн_ Спец	Зам_ Началь	Консультант	Начальник	Специал_ 1-й категор
Отдел демографии и перепис	5250			5500	5000		
Отдел маркетинга	2000						20(
Отдел статистики предприяти	3750	3000	4500				
Отдел труда и бюджетов	4666.66666666667		5000			7000	20(

Запись: 2 из 4

В перекрестных запросах можно использовать дополнительные условия отбора и сортировку.

ЗАПРОСЫ НА ИЗМЕНЕНИЕ БД – за одну операцию приводят к **необратимым** изменениям данных в нескольких записях. Существует четыре типа запросов на изменение, каждый из которых помечается в окне БД специальным значком:

Запрос на удаление – удаляет группу «старых» или неиспользуемых записей, удовлетворяющих заданным условиям, одной или нескольких таблиц.

Запрос на обновление – изменяет значения в группе записей одной или нескольких таблиц.

Запрос на добавление – пополняет таблицы БД группой новых записей. Данные могут быть добавлены из другой БД.

Запрос на создание таблицы – создает новую таблицу на основе данных одной или нескольких таблиц, которая будет являться подмножеством исходных таблиц.

ЗАПРОСЫ SQL – особый тип запросов, которые создаются при помощи конструкций SQL (Structured Query Language – язык структурного программирования) для обращения к серверам баз данных в сетевой версии СУБД. Этот тип запросов обычно используется опытными пользователями, имеющими навыки программирования, и для начинающих пользователей достаточно сложен.

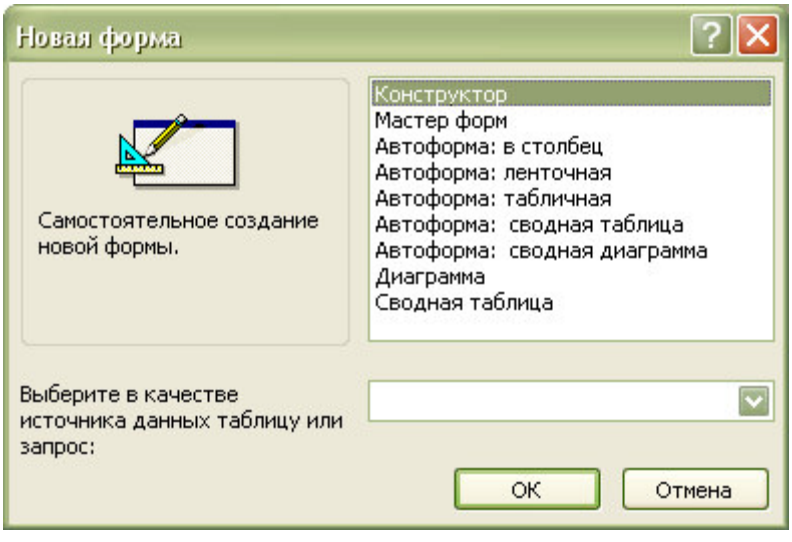
ЗАПРОСЫ С ВЫЧИСЛЯЕМЫМИ ПОЛЯМИ Можно создать запрос с дополнительным полем, которого нет в исходной таблице. Такое поле Access заполняет автоматически в соответствии с заданным для него выражением и присваивает ему имя **ВыражениеN** (таких полей может быть в запросе несколько). Для построения выражений используется **Построитель выражений**, вызываемый кнопкой или командой **Построить** в контекстном меню.

Формы в Access

Формы используются для организации удобного интерфейса БД. Они обеспечивают более гибкий способ ввода, просмотра и редактирования данных, позволяют вывести на экран одну или несколько записей в виде электронного бланка (шаблона). Формы могут использоваться для просмотра данных таблиц, содержащих очень большое количество полей. Такая форма может вмещать несколько десятков полей на одном экране. Кроме того, режим формы позволяет отобразить графические объекты и содержимое полей типа OLE, что невозможно в режиме таблицы. Для ввода новой информации, редактирования, поиска или удаления существующей информации используются **элементы управления** формы. Каждое текстовое поле формы связано с конкретным полем из таблицы. Таблица является источником записей, а поле является источником элемента управления. Каждый элемент управления имеет некоторое количество свойств, таких, как **стиль шрифта**, **размер** и **цвет**, которые вы можете изменять для улучшения внешнего вида формы (в режиме конструктора).

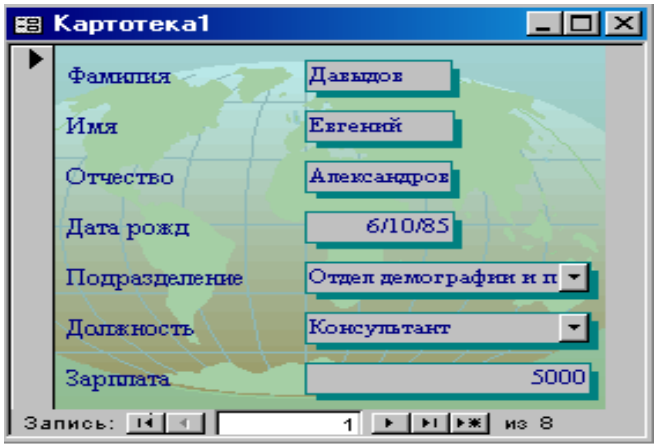
Форму можно создать разными способами: **Автоматически**, с помощью **Мастера форм**, с помощью **Конструктора форм**. Любой из них можно выбрать из списка окна **Новая форма**, которое открывается из окна БД:

- Щелчком на кнопке Создать, предварительно должен быть выделен объект **Формы**;
- Командой меню **Вставка/Форма**.

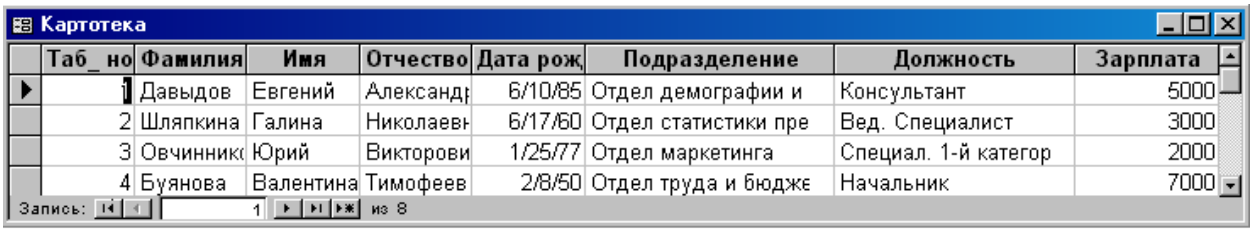


Автоматический режим – **Автоформа** - предусматривает создание формы без участия пользователя, все параметры настройки формы программа задает по умолчанию. Пользователь лишь выбирает таблицу, для которой создается форма, определяет тип формы (в столбец, ленточная или табличная) и задает имя. Автоформы являются частными случаями мастера форм.

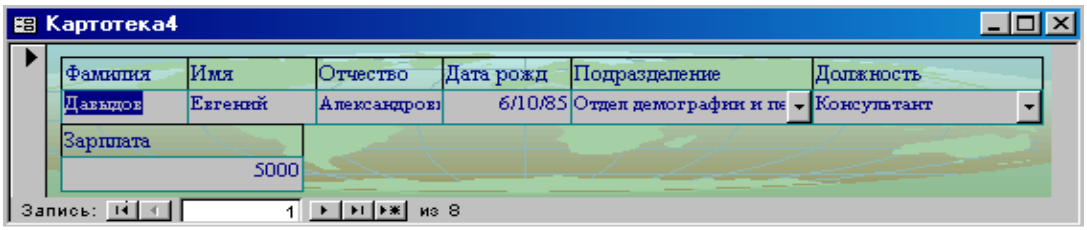
Мастер форм позволяет полуавтоматически создать форму на основе выбранных полей. Выбор полей для формы выполняется аналогично формированию запроса. Мастер форм позволяет настроить цвет и стиль оформления формы, а также ее вид: **в один столбец, ленточный, табличный, выровненный** (основные виды форм представлены на рисунке).



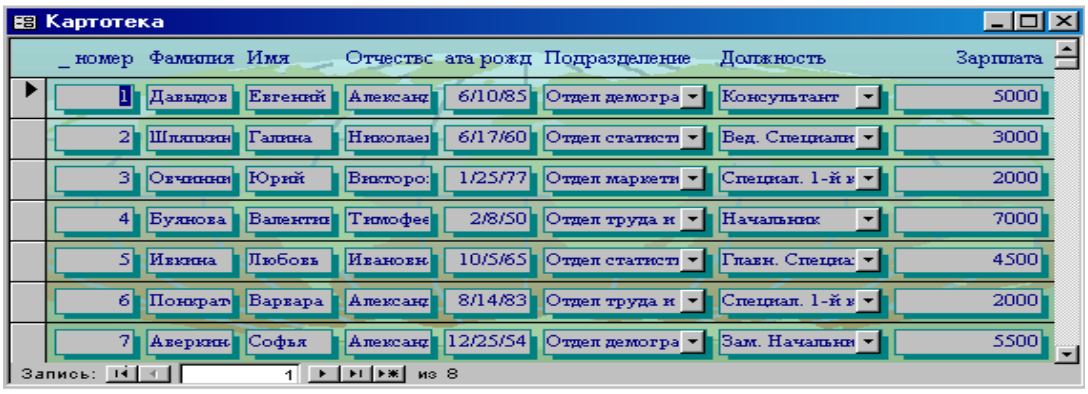
форма в один столбец



табличная форма



выровненный вид формы



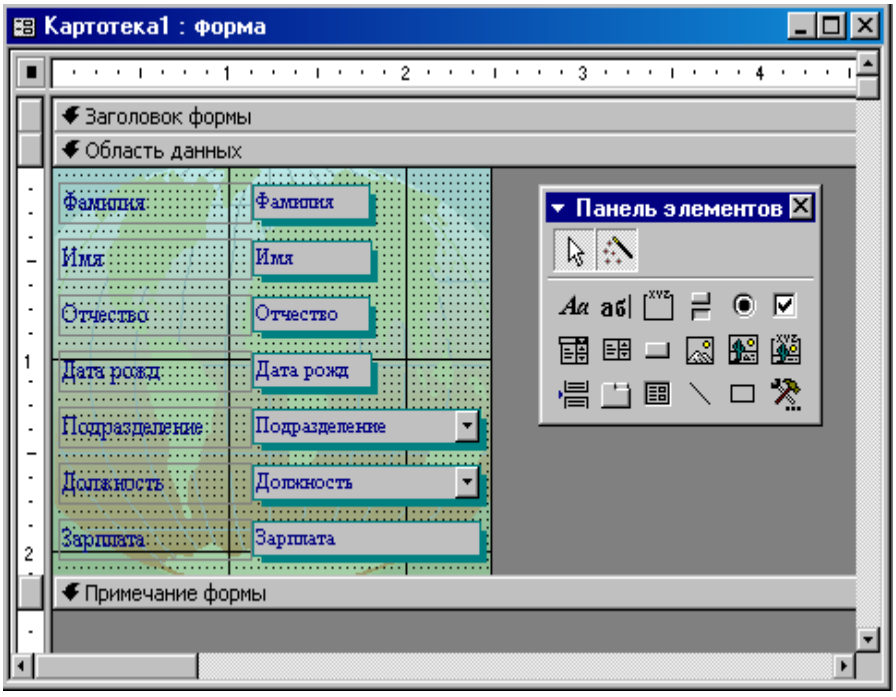
ленточная форма

Способ создания формы с помощью мастера наиболее удобен для начинающих пользователей, так как параметры формы настраиваются в режиме диалога.

Режим **Диаграмма** позволит создать форму со встроенной диаграммой, режим **Сводная таблица** – со сводной таблицей Excel.

Конструктор форм позволит создать форму самостоятельно, но для непрофессиональных пользователей это сложный и долгий процесс. Поэтому создавать форму удобнее с помощью **Мастера**, а дополнять и улучшать ее внешний вид – в режиме **Конструктора**.

Для открытия готовой формы в режиме конструктора можно воспользоваться кнопкой  на панели инструментов или командой меню **Вид/Конструктор**. В конструкторе для добавления в форму элементов (полей ввода, вычисляемых полей, надписей, кнопок и др.) используются кнопки **панели элементов,** а также окно **свойств формы (**см. рис.).



Макет формы состоит из разделов: **область данных** (содержит данные из таблиц), **заголовок формы** (верхняя часть первой страницы), **примечание формы** (нижняя часть последней страницы), **верхний и нижний колонтитулы** (при печати формы). Выделяя мышью отдельный элемент формы можно с помощью команд меню и кнопок панели инструментов изменять настройки данного элемента, а также перемещать элемент в пределах формы. В режиме конструктора вы можете изменять размер трех основных областей формы: **Заголовка формы**, **Области данных** и **Примечания формы**. Можно настраивать любой из этих разделов формы, добавляя и удаляя надписи, перемещая надписи и элементы управления текстом, и добавляя логотипы и другую графику. Наиболее популярные элементы управления находятся в **Панели элементов**.

Элементы формы могут выполнять различные действия, определяемые макросом или процедурой VBA, прикрепленных к ним.

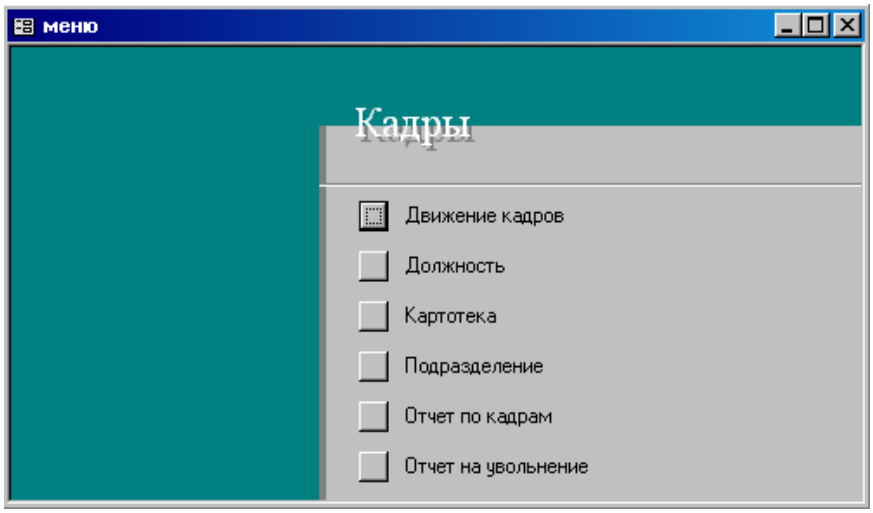
Для возврата в режим просмотра формы используется кнопка  или команда меню **Вид/Форма**.

Если разрабатывается форма на основе нескольких таблиц, Access предложит создать либо **подчиненную**, либо **связанную** форму. Такие формы называются **сложными**. **Подчиненная форма** представляет собой форму внутри другой формы. Первичная форма называется **главной**, а внутренняя – **подчиненной** (субформой). Подчиненная форма удобна для просмотра данных таблиц и запросов, связанных по типу **один-ко-многим** или **один-к-одному**. В подчиненной форме выводятся только те записи, которые связаны с текущей записью в главной форме. Главная форма может отображаться только в виде столбца, подчиненная – в столбцовом, табличном или ленточном виде. Главная форма может иметь несколько подчиненных форм и до семи уровней вложенности.

Для манипулирования записями в режиме формы используются те же способы, что и в режиме таблицы – поиск, замена, сортировка, фильтрация данных. Настройка параметров

просмотра и печати форм, так же как и таблиц, производится с помощью команд **Параметры страницы**, **Предварительный просмотр** и **Печать** в меню **Файл**.

Особым типом формы является **Кнопочная форма** (**Кнопочное меню**) – форма с расположенными на ней элементами управления, кнопками с поясняющими надписями (см. рисунок).



Щелчком на командной кнопке можно открыть соответствующую таблицу, форму или отчет. Это удобный инструмент работы с БД. Кнопочное меню можно создать вручную с помощью **конструктора** (однако для этого требуется достаточно много времени) или воспользоваться специальной программой - **Диспетчером кнопочных форм**. Созданную с помощью диспетчера кнопочную форму можно отредактировать или дополнить в режиме конструктора. Диспетчер кнопочных форм открывается командой меню **Сервис/Служебные программы/Диспетчер кнопочных форм**.

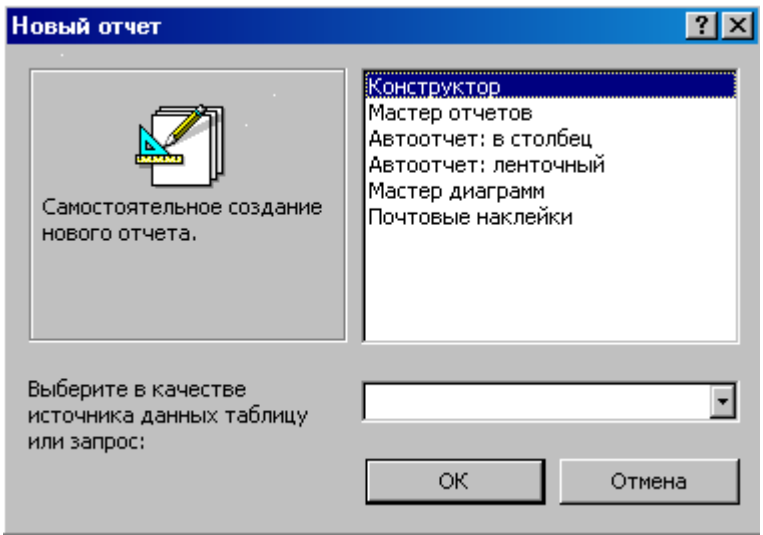
Отчеты в Access

Для предоставления конечного документа по работе с БД данные таблиц и запросов могут быть оформлены в виде отчета. **Отчет** – гибкое и эффективное средство для организации просмотра и распечатки итоговой информации, позволяющее представить данные на печать в желаемом формате. В Access 2003 вы можете создать **главный отчет**, который служит как оболочка для одного или нескольких **суботчетов**.

В ваш **отчет** и/или **суботчеты** вы можете добавлять **элементы управления** и можете устанавливать **свойства** этих элементов управления. Вы можете отображать информацию из одной или нескольких записей из одной или нескольких таблиц или запросов, и можете устанавливать несколько наборов заголовков и нижних колонтитулов.

Отчет можно дополнить результатами сложных расчетов и статистической обработки, рисунками и диаграммами. Также как и формы, отчеты могут содержать вычисляемые поля. В формах значения вычисляемых полей рассчитываются на основе полей в текущей записи. В отчетах вычисляемые поля (итоги) формируются на основе группы записей, страницы записей или всех записей отчета. Вы можете сделать отчет или суботчет более полезным, выполнив вычисления в самом отчете. Для создания выражений, по которым нужно выполнить вычисления, вы можете использовать **Построитель Выражений**.

Предварительный просмотр печати позволяет вам проверять отчеты перед их печатью.



Просмотр **Образец** отображает только ту информацию, которой достаточно для того, чтобы показать все элементы отчета.

Разработать новый отчет, как и любой другой объект БД можно самостоятельно (в режиме **Конструктора**) или с помощью **Мастера отчетов**. Режим создания отчета можно выбрать в окне **Новый отчет**, которое открывается

кнопкой  из окна БД или командой меню **Вставка/Отчет**. Создать отчет можно также двойным щелчком на соответствующем режиме в окне БД: **Создание отчета в режиме конструктора** или **Создание отчета с помощью мастера**. Мастер отчетов является наиболее удобным средством, как для начинающих, так и для опытных пользователей. Мастер отчетов состоит из нескольких диалоговых окон, содержащих приглашения ввести необходимые данные (диалоговый режим), и отчет создается на основании ответов пользователя. Такой способ создания отчета позволяет быстро разработать макет, на основе которого можно с помощью инструментов конструктора разработать высоко профессиональный документ, имеющий более привлекательный вид. Для начинающих пользователей также как и при создании формы, разработку отчета рекомендуется выполнять с помощью мастера, а детальную настройку отдельных элементов и добавление новых элементов – в режиме конструктора.

В зависимости от того, какой отчет создается – простой или сложный (с группировкой данных) – мастер предлагает различные варианты макетов отчета, образец каждого макета приводится в окне мастера.

Простейшие способы создания отчетов – разработка их в автоматическом режиме (**Автоотчет в столбец** и **Автоотчет ленточный**), пользователю достаточно лишь выбрать таблицу или запрос, на основе которого будет создаваться отчет.

Сформированный отчет выводится в окно предварительного просмотра печати.

Мастер диаграмм поможет создать отчет в виде диаграммы.

Мастер Почтовых наклеек создаст отчет, отформатированный для печати почтовых наклеек.

В режиме конструктора бланк отчета разделяется на несколько областей:

- **Область данных** - где размещаются записи из таблицы или запроса;
- **Заголовок отчета** – размещается в начале бланка отчета;
- **Область примечаний** – размещается в конце отчета;
- **Верхний** и **нижний колонтитулы** (в области нижнего колонтитула обычно выводятся номера страниц отчета и текущая дата);
- **Заголовок** и **Примечание группы** – присутствуют в бланке отчета, если задана группировка записей. Количество этих разделов - по числу уровней группировки.

Группа – набор записей, выделяемых по определенному признаку. Она состоит из заголовка, области данных и примечания. Группировка позволяет разбить записи на логические группы и отобразить промежуточную и итоговую информацию для каждой.

Для настройки отдельных элементов отчета, а также для добавления новых используются команды меню **Правка**, **Вид**, **Вставка**, **Формат**, **кнопки панелей инструментов** и **панель элементов**, а также команды контекстного меню для выделенного элемента. Для перехода в режим конструктора используется кнопка  . Возврат в окно просмотра осуществляется кнопкой  .

Обмен информацией с другими программами

Access 2003 облегчает импорт информации из многочисленных форматов, созданных другими программами. Если информация из другой программы все еще активно обновляется, вы можете связать вашу базу данных Access с этой информацией, не извлекая ее из оригинальной программы для дальнейшей обработки.

Вы можете импортировать в новую или существующую таблицу из рабочего листа Excel весь лист или именованный диапазон. Вы также можете импортировать из рабочего листа отдельные поля.

Вы можете использовать для импорта в вашу базу данных Access текстовых файлов с разделителями и с фиксированной шириной данных **Мастер импорта (Import Wizard)**.

Вы легко можете импортировать один или несколько стандартных объектов Access: таблицы, запросы, формы, отчеты, страницы, макросы и модули.

Вы можете импортировать в Access 2003 данные из некоторых версий **dBASE**, **Lotus 1-2-3** и **Paradox**. Можно импортировать информацию в первоначальном виде ,и обработать ее в Access, а можете перенести ее в другую программу, такую, как **Excel** или **Word**, и обработать ее перед импортом в Access.

Можно импортировать в Access 2003 документ, сохраненный с помощью другой программы в формате HTML. При попытке импортировать HTML-документ в Access 2003, он проведет разбор документа и идентифицирует все, что выглядит как структурированные данные. Затем эти данные можно просмотреть, и решить, импортировать ли это или нет.

Вы можете импортировать в Access 2003 файлы **Extensible Markup Language** (**XML**). XML состоит из данных и схемы, которая описывает структуру данных. Программы, которые экспортируют XML, могут объединять данные и схему в одном файле, а могут создавать два отдельных файла. Если программа экспортирует два отдельных файла, для импорта данных и структуры в Access 2003 нужно иметь оба этих файла.

Вы можете экспортировать информацию из вашей базы данных Access в различные другие форматы, набор которых зависит от объекта, который вы пытаетесь экспортировать.

Вы можете оставить данные в другой программе и создать связь с ней.

Вы также можете скопировать данные из вашей базы данных в другую программу через **буфер обмена**. Быстрым способом совместного с Word или Excel использования информации в базе данных Access 2003 является использование кнопки **Связи с Office** (OfficeLinks) на панели инструментов. Вы можете объединить данные из таблицы с документом письма Word, опубликовать таблицу в документе Word или экспортировать таблицу в электронную таблицу Excel.

Доступ к базе данных

- Чтобы облегчить другим пользователям доступ к вашим данным и работу с ними, а также исключить **случайное** изменение или удаление данных, вы можете создать **кнопочную форму** и установить параметры запуска.
- Можно создать **экран заставки**, который появляется при открытии вашей базы данных. Он может содержать рекламу, графический логотип или анимацию, а также может предоставлять пользователям полезные инструкции.
- В Access 2003 имеется несколько утилит, которые вы можете использовать для поддержания хорошей работоспособности вашей базы данных - **Сжать и восстановить базу данных (Compact and Repair Database)**, **Анализ быстродействия (Performance Analyzer), Архивариус (Documenter), и Найти и восстановить (Detect and Repair)**. Вы можете поддерживать работу вашего приложения, используя возможности этих утилит до того, как Access укажет, что в вашей базе данных возникли проблемы.

Организация безопасности информации

Для организации безопасности информации в базе данных в Access 2003 можно использовать различные средства:

Можно зашифровать базу данных, что не защищает ее от открытия и просмотра в Access, но не дает возможности прочитать ее содержимое тем людям, у которых нет установленной копии Access.

Вы можете задать для своей базы данных пароль, предотвратив ее открытие неавторизованными пользователями.

Можно совместно использовать базу данных в рамках локальной сети (LAN) и с помощью сетевой безопасности, которая используете для защиты информации в сети, ограничить действия пользователей.

Чтобы запретить одновременное обновление одной и той же записи несколькими пользователями (при сетевом доступе к БД), можно реализовать **пессимистическое**

блокирование, которое блокирует запись на все время ее редактирования, или **оптимистическое блокирование**, которое блокирует запись только на короткое время, пока Access сохраняет изменения.

Вы можете преобразовать вашу базу данных в новую версию (Design Master), а затем создать **реплики** этой главной базы данных для распространения в удаленных местах. Затем эти **реплики** могут синхронизироваться и объединять свои изменения с главной базой данных. После того как все изменения будут записаны, все реплики обновляются в соответствии с текущей информацией из Design Master и отправляются обратно в их расположения.

Можно разделить базу данных на «**табличную**» **часть** базы данных, которая содержит таблицы, и **интерфейсную часть** базы данных, которая содержит все остальные объекты базы данных. Вы можете хранить **табличную часть** базы данных на **сервере**, а **интерфейсную часть** распространить среди других пользователей, которые работают с данными. Они могут использовать все созданные вами объекты (за исключением таблиц, которые недоступны для редактирования) или создавать свои собственные.

Если вы добавили в базу данных процедуры VBA, вы можете защитить ваш код VBA с помощью **пароля**, либо сохранив базу данных как файл Microsoft Database Executable (MDE). Если вы устанавливаете для кода пароль, этот код остается доступным для редактирования всеми, кто знает этот пароль. Если вы сохраняете базу данных как файл MDE, другие пользователи могут запускать код, но не могут его просматривать или редактировать.

2.6 Лабораторная работа №6 (2 часа).

Тема: «Создание однотабличной базы данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы

2.6.1 Цель работы: изучить и проанализировать создание однотабличной базы данных

2.6.2 Задачи работы:

1. Изучение функционирования однотабличной базы данных
2. Разработка базы данных однотабличной базы данных

2.6.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.6.4 Описание (ход) работы:

Создание однотабличной базы данных.

Постановка задачи

Предположим, что организация (предприятие) создает информационную систему для автоматизации процессов учета кадров и контроля изменения заработной платы сотрудников. Личная карточка сотрудника содержит анкетные данные, информацию о дате приема на работу, должности, месте работы, общем стаже работы и размере заработной платы.

ИС должна функционировать в двух режимах:

- первичной загрузки данных – ввод информации из личных карточек;
- текущей обработки информации.

Пользователем базы данных может быть сотрудник отдела кадров или бухгалтерии.

ИС должна обеспечивать:

- контроль ввода новой информации;
- получение информации о сотрудниках по различным критериям;
- возможность обновления данных, в частности: перерасчет заработной платы, начисление различных надбавок и премий и т.п;
- возможность добавления в картотеку сведений о новых сотрудниках и удаления устаревшей информации;
- расчет итоговых показателей по различным подразделениям и должностям;
- подготовку кадровых и финансовых отчетов.

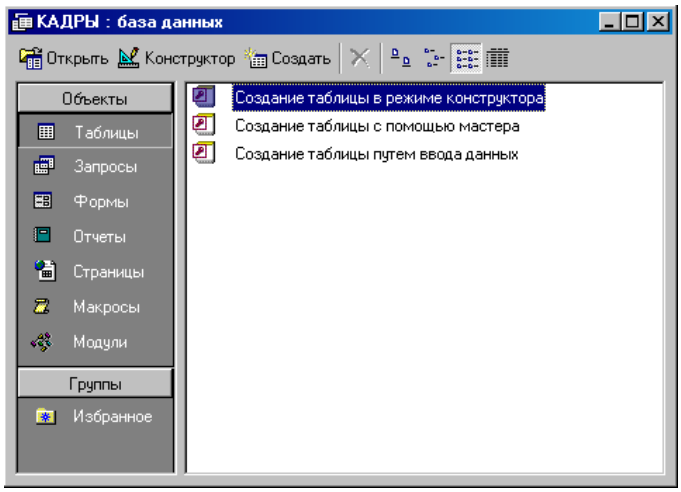
Задание на разработку базы данных

1. Разработать базу данных ***КАДРЫ*** с числом сотрудников 15 человек.
2. База данных содержит одну таблицу ***Картотека*** со следующей структурой записи: *Табельный номер, Фамилия, Имя, Отчество, Дата приема на работу, Подразделение, Должность, Общий стаж, Зарплата*.
3. Предусмотреть контроль ввода данных, задав ограничения на ввод данных по полю *Подразделение*, полю *Должность* и полю *Общий стаж*.
4. Ввод данных в таблицу осуществить посредством формы.
5. Создать запрос на выборку с параметром для получения информации о сотрудниках отдельных *подразделений* (отделы статистики, маркетинга, бухгалтерия).
6. Разработать запрос с параметром для получения информации о любом сотруднике организации по его *фамилии*.
7. Разработать запрос с параметром для получения информации о сотрудниках организации по *должности*.
8. Разработать запрос на сотрудников организации, принятых на работу в 2005 году.
9. Создать запрос о начислении *премии* работникам отдела статистики.
10. Рассчитать среднюю зарплату по должностям сотрудников.
11. Разработать запрос на удаление из базы данных записей, соответствующих уволенным сотрудникам.
12. Предусмотреть возможность увеличения заработной платы на 15% тем работникам, чей *общий стаж* работы превысил 20 лет.
13. Разработать формы для просмотра результатов запросов.
14. Создать отчет ***Кадры***, сгруппировав информацию по *подразделениям*, с расчетом средней, минимальной и максимальной зарплаты для сотрудников каждого подразделения.
15. Разработать отчет ***Премия*** с группировкой по *должности* и сортировкой записей в алфавитном порядке *фамилий* сотрудников.
16. Разработать кнопочную форму для удобства работы с базой данных.

Технология выполнения работы

1. Для открытия файла новой базы данных ***КАДРЫ*** выполните следующее:

- В меню **Пуск** выберите команду **Программы/Microsoft Office/Microsoft Access**, на экране откроется окно программы, предлагающее открыть базу данных в одном из трех режимов.
- Выберите режим **Новая база данных**, <OK>.
- В диалоге **Файл новой базы данных** в списке поля **Папка** выберите папку, в которой будет храниться файл базы данных ***КАДРЫ***;
- В строке **Имя Файла** введите с клавиатуры имя ***КАДРЫ*** вместо заданного по умолчанию имени db1.mdb (каждый следующий новый файл базы данных Access будет именовать соответственно db2, db3 и т.д.);
- В строке **Тип Файла** оставьте заданный программой по умолчанию тип файла **Базы данных Microsoft Access (*.mdb)**, <Создать>. На экране открывается окно базы



данных ***КАДРЫ***.

Для создания структуры таблицы ***Картотека*** воспользуемся режимом **конструктора** таблиц, для этого:

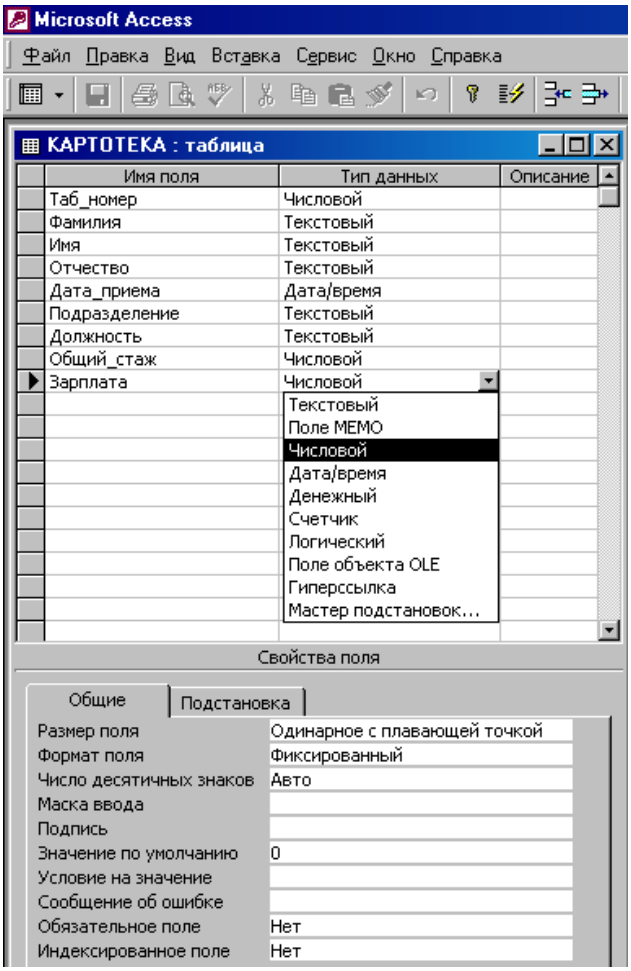
- В окне базы данных слева нужно выбрать объект **Таблицы**, режим **Создание таблицы в режиме конструктора** (можно щелкнуть на кнопке <Создать>, и из предложенного списка способов создания таблицы выбрать конструктор, <OK>);
- В окне конструктора таблиц определите имена полей и их характеристики (тип и размер) в соответствие с Таблицей 2.1:

Таблица 2.1.

Имя поля	**Тип поля**	**Размер поля**
Таб_номер	Числовой	Целое
Фамилия	Текстовый	15
Имя	Текстовый	15
Отчество	Текстовый	15
Дата_приема	Дата/Время	Краткий
Подразделение	Текстовый	35
Должность	Текстовый	35
Общий_стаж	Числовой	Целое
Зарплата	Числовой	Одинарное с плавающей точкой

В именах полей, состоящих из двух и более слов, не рекомендуется использовать в качестве разделителя слов знак пробела. Слова в имени поля можно разделять символом «_», или вообще не разделять, записывая каждое следующее слово с заглавной буквы (напр. *ОбщийСтаж*).

- Тип значений поля выбирается из списка в столбце **Тип поля** конструктора таблиц, размер поля устанавливается на вкладке **Общие** в окне **Свойства поля** в нижней части окна конструктора. Установите курсор в поле *Таб_номер*, на вкладке **Общие** в окне **Свойства поля**, в строке параметра **Размер поля** выберите из списка размер числового значения **Целое**;



- Установите курсор в поле *Фамилия*, в окне свойств поля в строке **Размер поля** введите с клавиатуры значение **15** (максимальное количество символов для записи фамилии, обычно задается по количеству символов в самой длинной фамилии списка, по умолчанию программа задает размер текстового поля 50 символов);
- Аналогично задайте размер для текстовых полей *Имя*, *Отчество*, *Подразделение*, *Должность*;
- Установите курсор в поле *Дата_приема*, в окне свойств поля в строке **Формат поля** откройте список и выберите **Краткий формат даты** (дд.мм.гггг);
- Размер поля *Общий_стаж* задайте аналогично полю *Таб_номер*;
- Установите курсор в поле *Зарплата*, в окне свойств поля в строке **Размер поля** выберите из списка **Одинарное с плавающей точкой** (дробное числовое значение с десятичной точкой), в строке **Формат поля** выберите из списка **фиксированный** (с двумя знаками после запятой).

Для этого поля можно также определить тип **Денежный**. Числовые значения данного типа отображаются в таблице в виде вещественных чисел с символом денежной единицы.

- Структура таблицы в режиме конструктора должна иметь следующий вид:

2. Чтобы ИС могла обеспечивать контроль вводимой информации, нужно задать ограничения на ввод значений в поле *Подразделение*, поле *Должность* и поле *Общий_стаж*, для этого выполните следующее:

- Выберите три подразделения, на работников которых создается база данных, например – **отдел статистики, отдел маркетинга, бухгалтерия.**
- Установите курсор в поле *Подразделение*. В окне свойств поля в строке параметра **Условие на значение** щелкните по кнопке ... **Построитель выражений**. В окне Построителя выражений введите значение «**отдел статистики»**, затем щелкните на кнопке оператора <**Or**> (логический оператор выбора ИЛИ), введите значение «**отдел маркетинга»**, щелкните на кнопке оператора <**Or**>, введите значение «**бухгалтерия»**, вернитесь в окно конструктора таблицы кнопкой <OK>. В строке **Условие на значение** появится выражение:

«отдел статистики» Or «отдел маркетинга» Or «бухгалтерия»

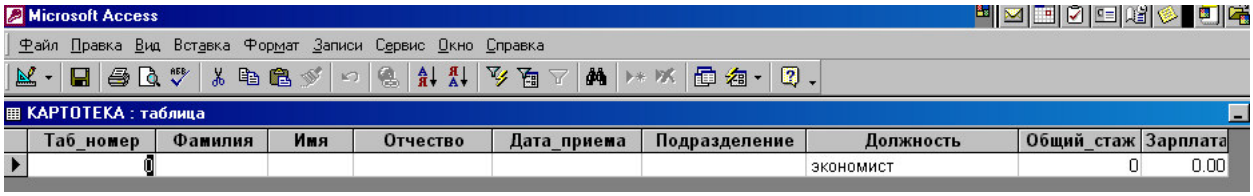
Построитель выражений – специальная программа, помогающая пользователю правильно записать условия и расчетные выражения с использованием синтаксиса допустимых конструкций языка VBA (Visual Basic for Application).

Для удобства ввода значений в поля, на которые наложены ограничения, можно сформировать текст сообщения об ошибке, которое будет выводиться на экран при попытке ввода значения, противоречащего условию. Например, если в поле *Подразделение* ввести значение **отдел кадров**, не предусмотренное условием, Access может вывести на экран следующее сообщение: **«Введено неверное подразделение, повторите ввод!!!»** (текст сообщения может быть произвольным, по усмотрению пользователя).

- Для формирования текста сообщения об ошибке установите курсор в поле *Подразделение*, в окне свойств поля в строке параметра **Сообщение об ошибке**, введите с клавиатуры текст: **ВВЕДЕНО НЕВЕРНОЕ ПОДРАЗДЕЛЕНИЕ, ПОВТОРИТЕ ВВОД!!!**
- Аналогично задайте ограничения на значения поля *Должность*, выбрав должности: **главный специалист, специалист 1й категории, экономист, бухгалтер**. Сформируйте текст сообщения об ошибке для этого поля.

Чтобы облегчить ввод больших объемов информации, для тех полей, в которых часто повторяются значения можно задать значение по умолчанию. Access автоматически будет заполнять указанным значением соответствующее поле таблицы (числовые поля автоматически по умолчанию заполняются нулевыми значениями). Пользователь при необходимости может менять его на другое значение в процессе заполнения таблицы. Например, для поля *Должность* можно определить значение по умолчанию – **экономист**, т.к. оно чаще других встречается в этом поле. Для этого установите курсор в поле *Должность*, в окне свойств поля в строке **Значение по умолчанию** откройте построитель выражений, введите слово **экономист**, вернитесь в конструктор кнопкой <OK>.

- Для задания ограничения на ввод значений в поле *Общий_стаж* в строке параметра **Условие на значение** с помощью построителя выражений запишите условие <**50**, сообщение об ошибке для этого поля сформулируйте аналогично полям *Подразделение* и *Должность*.
- Сохраните созданную структуру таблицы, щелкнув на кнопке , задайте имя таблицы ***Картотека***, в ответ на предложение определить ключевое поле щелкните по кнопке <Нет> (в противном случае программа автоматически создаст дополнительное поле *Код* и определит его как ключевое. В однотабличной базе данных определение ключевого поля - необязательно).
- Перейдите в режим просмотра таблицы, выбрав команду меню **Вид/Режим**



таблицы.

3. Для заполнения таблицы создайте форму:

- В окне базы данных выберите объект **Формы**, щелкните на кнопке <Создать>, из предложенного списка режимов создания формы выберите **Мастер форм**, <OK>.
- В окне мастера форм в поле **Таблицы и запросы** выберите таблицу ***Картотека***. В области **Доступные поля** перечислены все поля таблицы.
- Поочередно выделяя каждое поле перенесите его в область **Выбранные поля** кнопкой (для быстрого выполнения этой рации можно воспользоваться кнопкой , которая переносит сразу все поля из области **Доступные поля** в област **ыбранные поля**), щелкните на кнопке <Далее>.
- Из предложенного списка видов форм выберите **В один столбец**, <Далее>;
- Выберите подходящий вам стиль оформления формы, <Далее>.
- Задайте имя формы ***Кадровый Состав***, оставьте установленный по умолчанию режим **Открыть форму для просмотра**, щелкните на кнопке <Готово>.

Для более быстрого создания формы можно воспользоваться автоматическим режимом, для этого: из окна базы данных для объекта **Формы** нужно щелкнуть на кнопке <Создать>, выбрать из списка **Автоформа: в один столбец**, внизу в поле списка выбрать таблицу ***Картотека***, <OK>. Форма создается программой без участия пользователя и содержит все поля таблицы, параметры оформления задаются автоматически, пользователь задает лишь имя формы.

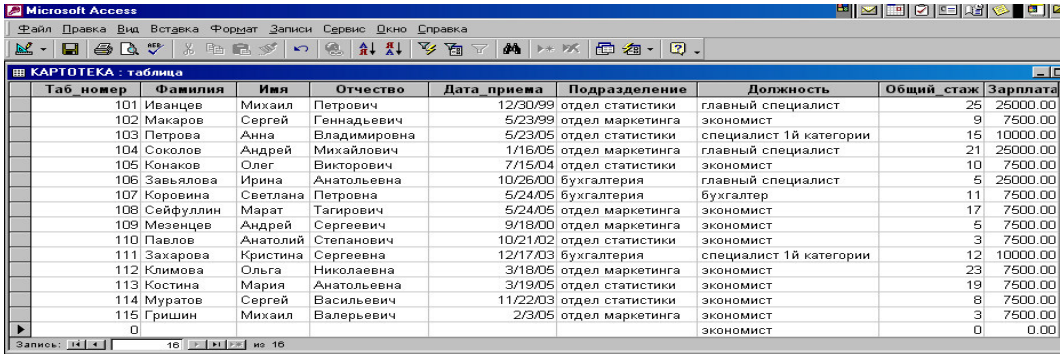
o Форму заполните данными на сотрудников организации в соответствии с Таблицей 2.2. Для перемещения от одного поля к другому можно использовать клавишу **Tab** (в обратном направлении **Shift+Tab**), для перехода к следующей записи воспользуйтесь кнопками прокрутки в строке **Запись** в нижней части окна формы: к следующей записи, к предыдущей записи, к первой записи, к последней записи, добавление пустой строки для новой записи. Проверьте, как будет осуществляться контроль вводимой информации в поля *Подразделение*, *Должность*, *Общий_стаж*. Для этого попробуйте ввести в эти поля значения, противоречащие заданным для них условиям в любой записи: в *Подразделение* введите – **отдел кадров**, в *Должность* введите **консультант**, в поле *Общий_стаж* укажите значение **55**. На каждое недопустимое значение программа реагирует сообщением об ошибке ввода данных.

Таблица 2.2.

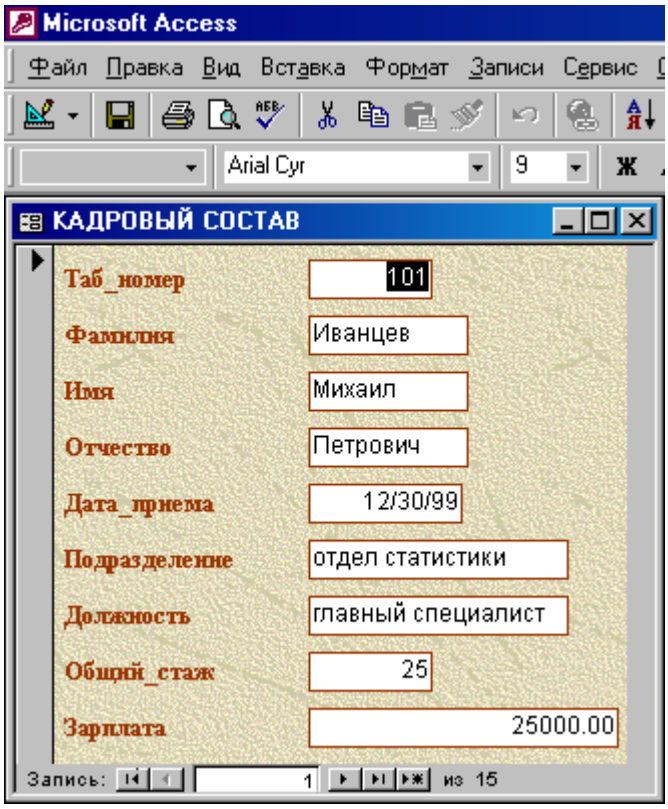
Таб. номер	**Фамилия**	**Имя**	**Отчество**	**Дата приема**	**Подразде-ление**	**Должность**	**Общий Стаж**	**Зар-плата**
101	Иванцев	Михаил	Петрович	10.12.00	отдел статистики	главный специалист	25	25000
102	Макаров	Сергей	Геннадьевич	23.05.99	отдел маркетинга	экономист	9	7500
103	Петрова	Анна	Владимировна	23.05.05	отдел статистики	специалист 1й категории	15	10000
104	Соколов	Андрей	Михайлович	16.01.05	бухгалтерия	главный специалист	21	25000
105	Конаков	Олег	Викторович	15.07.04	отдел статистики	экономист	10	7500
106	Завьялова	Ирина	Анатольевна	26.10.00	бухгалтерия	специалист 1й категории	5	10000
107	Коровина	Светлана	Петровна	24.05.05	бухгалтерия	бухгалтер	11	7500
108	Сейфуллин	Марат	Тагирович	24.05.05	отдел маркетинга	экономист	17	7500
109	Мезенцев	Андрей	Сергеевич	18.09.00	отдел маркетинга	экономист	5	7000
110	Павлов	Анатолий	Степанович	21.10.02	отдел статистики	экономист	3	7000
111	Захарова	Кристина	Сергеевна	17.12.03	бухгалтерия	специалист 1й категории	12	10000
112	Климова	Ольга	Николаевна	18.03.05	отдел маркетинга	экономист	23	7500
113	Костина	Мария	Анатольевна	19.03.05	отдел статистики	экономист	19	7500
114	Муратов	Сергей	Васильевич	22.11.03	отдел статистики	экономист	8	7500
115	Гришин	Михаил	Валерьвич	03.02.05	отдел маркетинга	экономист	3	7000

o Закройте форму ***Кадровый Состав***, предварительно сохранив ее щелчком на кнопке на панели инструментов.

o В окне базы данных выберите объект **Таблицы** и откройте таблицу ***Картотека*** для просмотра кнопкой <Открыть>, или двойным щелчком левой кнопки мыши. Заполненная таблица и соответствующая ей форма должны выглядеть следующим образом:



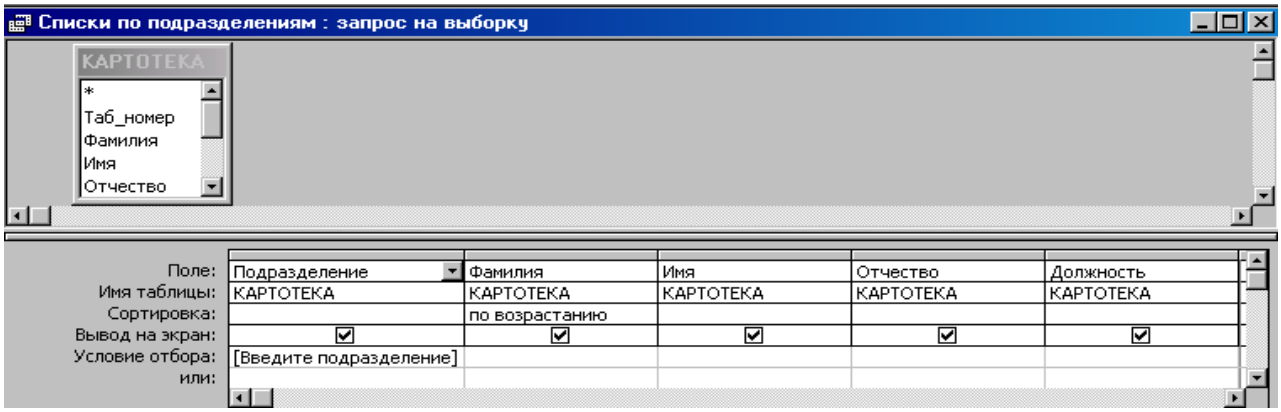
Таб_номер	Фамилия	Имя	Отчество	Дата_приема	Подразделение	Должность	Общий_стаж	Зарплата
101	Иванцев	Михаил	Петрович	12/30/99	отдел статистики	главный специалист	25	25000.00
102	Макаров	Сергей	Геннадьевич	5/23/99	отдел маркетинга	экономист	9	7500.00
103	Петрова	Анна	Владимировна	5/23/05	отдел статистики	специалист 1й категории	15	10000.00
104	Соколов	Андрей	Михайлович	1/16/05	отдел маркетинга	главный специалист	21	25000.00
105	Конаков	Олег	Викторович	7/15/04	отдел статистики	экономист	10	7500.00
106	Завьялова	Ирина	Анатольевна	10/26/00	бухгалтерия	главный специалист	5	25000.00
107	Коровина	Светлана	Петровна	5/24/05	бухгалтерия	бухгалтер	11	7500.00
108	Сейфуллин	Марат	Тагирович	5/24/05	отдел маркетинга	экономист	17	7500.00
109	Мезенцев	Андрей	Сергеевич	9/18/00	отдел маркетинга	экономист	5	7500.00
110	Павлов	Анатолий	Степанович	10/21/02	отдел статистики	экономист	3	7500.00
111	Захарова	Кристина	Сергеевна	12/17/03	бухгалтерия	специалист 1й категории	12	10000.00
112	Климова	Ольга	Николаевна	3/18/05	отдел маркетинга	экономист	23	7500.00
113	Костина	Мария	Анатольевна	3/19/05	отдел статистики	экономист	19	7500.00
114	Муратов	Сергей	Васильевич	11/22/03	отдел статистики	экономист	8	7500.00
115	Гришин	Михаил	Валерьевич	2/3/05	отдел маркетинга	экономист	3	7500.00
0						экономист	0	0.00



- Откройте вновь форму ***Кадровый Состав***, зарегистрируйте новых сотрудников, используя кнопку со звездочкой в строке **Запись**, введите 1-2 новых записи (данные задайте произвольно).
- Произведите сортировку данных по возрастанию *Даты_приема*: установите курсор в это поле и выполните команду **Записи/Сортировка;**
- Посмотрите в таблице ***Картотека*** новые данные командой меню **Вид/Режим таблицы**, вернитесь в окно формы и закройте ее с сохранением.
- Последовательно создайте три **Автоформы** с различным размещением полей: **ленточная**, **выровненная**, **табличная**. Сохраните одну из них по своему усмотрению под именем ***Кадровый Состав 2***.

3. Разработайте **запрос на выборку с параметром** для формирования списков сотрудников по отдельным подразделениям. Для этого выполните следующие действия:

- В окне базы данных выделите объект **Запросы**, щелкните на кнопке <Создать>, из предложенного списка режимов создания запросов выберите **Простой Запрос** (создание запроса с помощью мастера), <ОК>.
- В окне мастера запросов в поле списка **Таблицы и запросы** выберите таблицу ***Картотека***, из списка полей таблицы в области **Доступные поля** выделите и перенесите в область **Выбранные поля** следующие: *Подразделение*, *Фамилия*, *Имя*, *Отчество*, *Должность*, <Далее>.
- Задайте имя запроса ***Списки По Подразделениям***, щелкните на кнопке <Готово>. На экран выводится таблица с данными обо всех сотрудниках в составе пяти выбранных полей. Для того, чтобы отобрать из этого списка данные о сотрудниках одного какого-либо подразделения перейдите в режим **конструктора запросов** используя кнопку  на панели инструментов или команду меню **Вид/Конструктор** (перейти в режим конструктора сразу, не открывая запрос в режиме просмотра, можно установив режим **Изменить макет запроса** до нажатия кнопки <Готово>).



- В поле *Подразделение* в строке **Условие отбора** введите с клавиатуры параметр запроса **[Введите Подразделение]** (это приглашение будет выводиться на экран при каждом запуске запроса на выполнение).
- Определите порядок вывода записей в запросе. Установите курсор в поле *Фамилия* в строку **Сортировка**, откройте список и выберите **по возрастанию**.
- Сохраните запрос с внесенными в его структуру изменениями (кнопка).
- Запустить запрос на выполнение можно непосредственно в режиме конструктора, выберите в меню команду **Запрос/Запуск** или воспользуйтесь кнопкой на панели инструментов. В ответ на приглашение запроса **Введите подразделение** с клавиатуры введите слово **бухгалтерия**, Enter. В результате выполнения запроса на экране будет получен список работников бухгалтерии.

Списки по подразделениям : запрос на выборку

Подразделение	Фамилия	Имя	Отчество	Должность
бухгалтерия	Завьялова	Ирина	Анатольевна	главный специалист
бухгалтерия	Захарова	Кристина	Сергеевна	специалист 1й категории
бухгалтерия	Коровина	Светлана	Петровна	бухгалтер
				экономист

Выполните запрос для других подразделений для получения списков сотрудников отделов статистики и маркетинга.

2.7 Лабораторная работа №7, №8 (4 часа).

Тема: «Создание многотабличной базы данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы

2.7.1 Цель работы: изучить и проанализировать создание многотабличной базы данных

2.7.2 Задачи работы:

1. Изучение функционирования многотабличной базы данных
2. Разработка базы данных многотабличной базы данных

2.7.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.1.4 Описание (ход) работы:

Создание многотабличной базы данных.

Постановка задачи

Разрабатывается фрагмент информационной системы «**Управление персоналом**» для автоматизации работы отдела кадров предприятия с числом сотрудников 15 человек.

Система должна функционировать в двух режимах: первичной загрузки данных и текущей обработки информации.

В режиме загрузки БД система должна обеспечивать контроль вводимой информации, возможность редактирования данных.

Пользователем базы данных может быть сотрудник отдела кадров или бухгалтерии.

В режиме текущей обработки БД должна обеспечивать:

- получение анкетной информации о сотрудниках по данным личной картотеки;
- возможность проведения простейших бухгалтерских расчетов;
- обработку данных по движению кадров: прием, увольнение, перемещение;
- расчет итоговых показателей по различным подразделениям и должностям;
- подготовку кадровых и финансовых отчетов.

Задание на разработку базы данных

1. Разработать базу данных ***УПРАВЛЕНИЕ ПЕРСОНАЛОМ.***
2. База данных содержит четыре таблицы со следующей структурой записи:
   - ***Сотрудники*** (*Табельный номер, Фамилия, Имя, Отчество, Дата рождения, Код подразделения, Код должности, Оклад*),
   - ***Движение кадров*** (*Табельный номер, Дата приема, Движение ,Дата перевода, Дата увольнения)*,
   - ***Подразделение*** (*Код подразделения, Подразделение, Телефон)*,
   - ***Должность*** (*Код должности, Должность)*.
3. В таблице ***Сотрудники*** поля *Код подразделения* и *Код должности* должны быть полями подстановки из соответствующих таблиц ***Подразделение*** и ***Должность***.
4. Разработать схему данных БД, установить связи между таблицами.
5. Ввод данных в таблицы осуществить посредством формы.
6. Используя операции **Поиска** и **Замены** изменить значение оклада некоторым сотрудникам.
7. Разработать запрос ***Картотека*** для просмотра анкетных данных всех сотрудников, выполнить сортировку и фильтрацию данных.
8. Создать запрос ***Личная карточка*** для получения информации о любом сотруднике организации по его фамилии.
9. Создать перекрестный запрос ***Средний оклад*** для расчета среднего оклада для каждого подразделения по должностям сотрудников.
10. В запросе ***Начисление зарплаты*** рассчитать и начислить надбавку к окладу в зависимости от стажа работы.

11. Разработать запросы по движению кадров для получения информации об уволенных и переведенных в другой отдел сотрудниках (***Запрос по движению***, ***Перевод***, ***Увольнение***).
12. Разработать подчиненные формы ***Списки по подразделениям*** (должностям) для просмотра данных о сотрудниках по подразделениям и должностям.
13. На основе подчиненной формы ***Списки по подразделениям*** создать форму с диаграммой, отображающей размер оклада для всех сотрудников подразделения.
14. Разработать отчет ***Картотека*** с группировкой данных по подразделениям и расчетом итоговых значений по полю *Оклад*, отчет разбить на страницы по группам записей.
15. Создать отчет для печати ***Личной карточки*** сотрудника.
16. Разработать отчет ***Движение Кадров***.
17. Создать отчет ***Начисление Зарплаты*** с вычисляемым полем *К_выдаче*.
18. Разработать кнопочную форму-меню.

Технология выполнения работы

1. Создайте новый файл БД по технологии, предложенной для создания однотабличной БД (**пример1**), либо с помощью команды меню **Файл/Создать**, либо используя кнопку  на панели инструментов. Задайте имя файла БД ***УПРАВЛЕНИЕ ПЕРСОНАЛОМ.***

2. Для разработки структуры таблиц используйте конструктор таблиц. Для этого:

o в окне БД выберите объект **Таблицы – Создание таблицы в режиме конструктора.**

o Разработайте структуру таблицы ***Движение кадров*** в соответствии с данными таблицы 2.3.

Таблица 2.3.

Имя поля	**Тип поля**	**Размер поля**
ТабНомер	Числовой	Длинное целое
ДатаПриема	Дата/Время	Средний формат даты
Движение	Текстовый	15
ДатаПеревода	Дата/Время	Средний формат даты
ДатаУвольнения	Дата/Время	Средний формат даты

o Поле *ТабНомер* определите как ключевое, для этого выделите его и щелкните на кнопке  , сохраните таблицу под именем ***Движение кадров.***

o Разработайте структуру таблицы ***Подразделение*** в соответствии с данными таблицы 2.4.

Таблица 2.4.

Имя поля	**Тип поля**	**Размер поля**
КодПодразделения	Числовой	Длинное целое
Подразделение	Текстовый	35
Телефон	Текстовый	9

- Поле *КодПодразделения* определите как ключевое. В строке параметра **Значение по умолчанию** удалите 0, т.к. первичные ключи не могут принимать нулевые значения. Сохраните структуру таблицы, задав имя ***Подразделение***.
- Разработайте структуру таблицы ***Должность*** в соответствии с данными таблицы 2.5.

Таблица 2.5.

Имя поля	**Тип поля**	**Размер поля**
КодДолжности	Числовой	Длинное целое
Должность	Текстовый	35

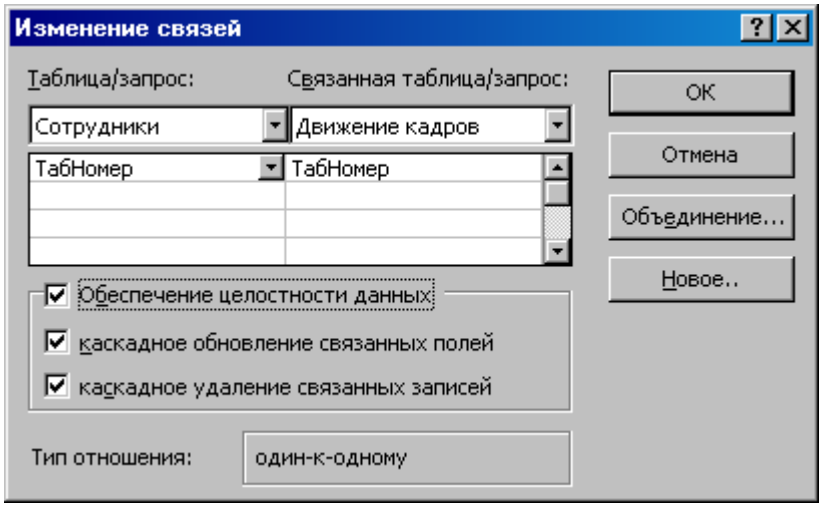
- Поле *КодДолжности* определите как ключевое. В строке параметра **Значение по умолчанию** удалите 0. Сохраните структуру таблицы под именем ***Должность***.

- Для разработки структуры таблицы ***Сотрудники*** воспользуйтесь структурой похожей таблицы ***Картотека*** однотабличной БД ***КАДРЫ***. Откройте БД ***КАДРЫ***, объект **Таблицы**, таблица ***Картотека***. В меню выберите команду **Правка/Копировать** (можно воспользоваться контекстным меню или кнопками на панели инструментов). Вернитесь в окно БД ***УПРАВЛЕНИЕ ПЕРСОНАЛОМ***, выполните команду **Правка/Вставить**, укажите режим вставки – **структуру и данные**, задайте имя таблице ***Сотрудники***.
- Откройте таблицу ***Сотрудники*** в режиме конструктора.
- Внесите изменения в структуру таблицы: удалите поле *Дата_приема* клавишей **Del** или кнопкой  , предварительно его выделив. Добавьте три дополнительных поля *ДатаРождения*, *КодПодразделения*, *КодДолжности* (для вставки строк используйте кнопку ). Поля *КодПодразделения* и *КодДолжности* будут использоваться для связи с таблицами ***Подразделение*** и ***Должность***. Измените имя поля *Зарплата* на *Оклад*. Характеристики полей задайте в соответствии с таблицей 2.6.

Таблица 2.6.

Имя поля	**Тип поля**	**Размер поля**
ТабНомер	Числовой	Длинное целое
Фамилия	Текстовый	15
Имя	Текстовый	15
Отчество	Текстовый	15
ДатаРождения	Дата/Время	Средний формат даты
КодПодразделения	Числовой	Длинное целое
КодДолжности	Числовой	Длинное целое
Оклад	Числовой	Одинарное с плавающей точкой

- Для поля *КодПодразделения* задайте **подстановку** значений из таблицы ***Подразделение***: курсор установите на поле *КодПодразделения*, откройте вкладку **Подстановка** окна **Свойства поля**, в строке **Тип элемента управления** выберите **Поле со списком**, в строке **Тип источника строк** оставьте **Таблица или запрос**, в строке **Источник строк** выберите таблицу ***Подразделение***.
- Аналогично задайте подстановку для поля *КодДолжности*.
- Определите ключевое поле *ТабНомер*, сохраните таблицу с внесенными в ее структуру изменениями, закройте окно конструктора таблиц.
- Просмотрите созданные таблицы в режиме таблицы.



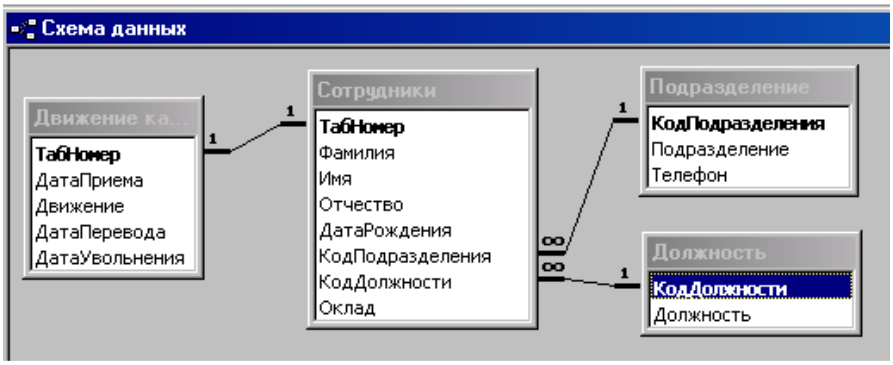
3. Установите связи между таблицами с помощью **Схемы данных**. Для этого выполните следующие действия:

- Из окна БД щелкните на кнопке  . Открывается окно **Схема данных** вместе с диалогом **Добавление таблицы**, в котором в списке **Таблицы и запросы** перечислены четыре таблицы ***Движение кадров***, ***Сотрудники***, ***Подразделение***, ***Должност.***

- Выделите первую таблицу и кнопкой <Добавить> разместите ее макет в окне схемы данных. Выполните ту же операцию с остальными таблицами так, чтобы все четыре таблицы отобразились на схеме данных.
- Закройте диалог **Добавление таблицы**.

Если по ошибке в схему данных была добавлена лишняя таблица, ее можно удалить командой **Скрыть таблицу** в контекстном меню этой таблицы.

- Для установления связи между таблицами ***Сотрудники*** и ***Движение кадров*** выделите в любой из этих таблиц поле *ТабНомер* (оно является ключевым в обеих таблицах) и «перетащите» его на поле *ТабНомер* второй таблицы, т.е. совместите одноименные поля двух таблиц. Открывается диалог **Изменение связей**.
- В диалоге **Изменение связей** обратите внимание на тип устанавливаемой связи **один-к-одному** (так как оба поля связи являются ключевыми) в нижней строке диалога.
- Установите режимы **Обеспечение целостности данных**, **Каскадное обновление связанных полей**, **Каскадное удаление связанных записей**, щелкните на кнопке <OK>.
- Установите связи между таблицами ***Сотрудники*** и ***Подразделение*** по полю *КодПодразделения*. Обратите внимание на тип устанавливаемой связи **один-ко-многим**. Access определяет такой тип отношения потому что поле *КодПодразделения* является ключевым только в одной таблице ***Подразделение*** (**уникальный ключ**), для второй таблицы ***Сотрудники*** это поле неключевое (**внешний ключ**).
- Аналогично установите связи между таблицами ***Сотрудники*** и ***Должность***.



Если при установлении связи программа определяет тип отношения **не определено**, значит связь устанавливается по неключевым полям. В этом случае нужно вернуться в конструктор таблиц и определить ключевое поле, или выбрать другое поле связи. Для удаления неверно установленной связи нужно подвести указатель мыши к линии связи и в контекстном меню выбрать команду **Удалить связь**.

4. Разработайте формы для заполнения таблиц: для таблиц ***Сотрудники*** и ***Движение кадров*** – **в один столбец**, для таблиц ***Подразделение*** и ***Должность*** – **ленточный вариант** (можно воспользоваться режимом Автоформы). Формам задайте имена, соответствующие именам таблиц.
5. Заполните формы ***Подразделение*** и ***Должность*** следующими данными:

КодПодразделения	Подразделение	Телефон
2001	отдел статистики	475432678
2002	отдел маркетинга	475589224
2003	бухгалтерия	476598874

КодДолжности	Должность
1001	главнный специалист
1002	специалист 1й категории
1003	экономист
1004	бухгалтер

- Заполните форму ***Движение кадров*** следующими данными:

ТабНомер	ДатаПриема	Движение	ДатаПеревода	ДатаУвольнения
101	24-апр-97	уволен		10-апр-02
102	23-май-99	переведен	10-апр-00	
103	23-май-05			
104	16-янв-05	переведен	02-ноя-05	
105	15-июл-04			
106	26-окт-00	переведен	08-сен-04	
107	24-май-05	уволен		10-окт-05
108	24-май-05			
109	18-сен-00			
110	21-окт-02	уволен		20-май-04
111	17-дек-03			
112	18-мар-05			
113	19-мар-05			
114	22-ноя-03	переведен	21-ноя-05	
115	03-фев-05			

- Заполните в форме ***Сотрудники*** поля *ДатаРождения*, *КодПодразделения* и *КодДолжности* данными из нижеприведенной таблицы. Значения полей *КодПодразделения* и *КодДолжности* выбирайте из списков этих полей так как это поля подстановки.

ТабНомер	Фамилия	Имя	Отчество	ДатаРожден	КодПодразд	КодДолжн	Оклад
101	Иванцев	Михаил	Петрович	04-мар-56	2001	1001	25000.00
102	Макаров	Сергей	Геннадьевич	04-дек-76	2002	1003	7500.00
103	Петрова	Анна	Владимировна	14-апр-70	2001	1002	10000.00
104	Соколов	Андрей	Михайлович	23-фев-62	2002	1001	25000.00
105	Конаков	Олег	Викторович	12-сен-69	2001	1003	7500.00
106	Завьялова	Ирина	Анатольевна	20-май-78	2003	1001	25000.00
107	Коровина	Светлана	Петровна	27-ноя-73	2003	1004	7500.00
108	Сейфуллин	Марат	Тагирович	14-июн-67	2002	1003	7500.00
109	Мезенцев	Андрей	Сергеевич	02-янв-79	2002	1003	7500.00
110	Павлов	Анатолий	Степанович	10-авг-81	2001	1003	7500.00
111	Захарова	Кристина	Сергеевна	25-апр-72	2003	1002	10000.00
112	Климова	Ольга	Николаевна	28-фев-61	2002	1003	7500.00
113	Костина	Мария	Анатольевна	01-янв-66	2001	1003	7500.00
114	Муратов	Сергей	Васильевич	18-дек-76	2001	1003	7500.00
115	Гришин	Михаил	Валерьевич	03-ноя-80	2002	1003	7500.00

Установленные между таблицами связи обеспечивают контроль ввода данных в поля *КодПодразделения* и *КодДолжности* в таблице Сотрудники. Для проверки попробуйте в любой записи ввести с клавиатуры значение *КодаПодразделения* – 3001(или любое другое). Система выдает сообщение об ошибке – попытка нарушения целостности данных в связанных записях.

6. Используя операции **поиска** и **замены** данных, измените значения оклада сотрудникам – **экономистам** (код должности 1003) с **7500р**. на **7850р**. Для этого выполните следующие действия:
   - откройте таблицу ***Сотрудники***, установите курсор в поле *Оклад*;
   - откройте диалог замены данных командой меню **Правка/Замена** или кнопкой  ;
   - в строку **Образец** введите значение **7500**, в строку **Заменить на** значение **7850**;
   - в строке **Поиск в:** должно быть указано поле *Оклад*;
   - в строке **Совпадение** выберите из списка **С любой частью поля**;
   - для поиска первого совпадения щелкните на кнопке <Найти далее>. Курсор в таблице остановится на первой записи со значением оклада 7500, если в этой записи *КодДолжности* имеет значение 1003 щелкните на кнопке <Заменить>, в противном случае пропустите значение и продолжите поиск кнопкой <Найти далее>;

выполняйте поиск и замену по всем записям таблицы, по окончании поиска на экран выводится сообщение: «Поиск записей в приложении завершен».

2.8 Лабораторная работа №9 (2 часа).

Тема: «Формирование запросов и отчетов для многотабличной базы данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.8.1 Цель работы: изучить и проанализировать формирование запросов и отчетов для многотабличной базы данных

2.8.2 Задачи работы:

1. Изучение запросов для многотабличной базы данных
2. Изучение отчетов для многотабличной базы данных

2.8.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.8.4 Описание (ход) работы:

Формирование запросов и отчетов для многотабличной базы данных.

Для разработки запросов к БД и отчетов используйте технологию, рассмотренную в **примере 1** (согласно данным ЛР-7). Для краткости изложения подробно повторять ее не будем.

1. Для разработки запроса ***Картотека*** (просмотр анкетных данных сотрудников) выполните следующее:

- создайте простой запрос на выборку, выберите в него поля: *Фамилия*, *Имя*, *Отчество* (из таб. ***Сотрудники***), *Подразделение* (из таб. ***Подразделение***), *Должность* (из таб. ***Должность***), *Оклад* (из таб. ***Сотрудники***), *Движение* (из таб. ***Движение кадров***). Поля нужно выбирать в той последовательности, в которой они должны выводится в результате выполнения запроса (поле *Оклад* должно быть последним). Задайте имя запроса ***Картотека***, кнопкой <Готово> запустите его на выполнение. На экран выводится список сотрудников с анкетными данными.
- Выполните фильтрацию данных запроса по полю *Подразделение*: установите курсор в этом поле на значение **бухгалтерия** в любой записи и щелкните на кнопке **фильтр по выделенному**  . В запросе остаются только записи о сотрудниках бухгалтерии. Отмените фильтр кнопкой  . Установите фильтр для сотрудников других подразделений.
- Аналогично выполните фильтрацию по полю *Должность*. Отмените фильтр.
- Отсортируйте данные ***Картотеки*** в алфавитном порядке фамилий: выделите поле *Фамилия* и щелкните на кнопке  . Повторите сортировку по другим полям запроса.

2. Для просмотра личной карточки сотрудника создайте простой запрос с параметром (параметрический) на основе данных запроса ***Картотека***:

- создайте простой запрос, выберите в него все поля запроса ***Картотека***, задайте имя ***Личная карточка***, перейдите в режим конструктора;
- в конструкторе в поле *Фамилия* в строку **Условие отбора** введите параметр **[Введите фамилию сотрудника]**, сохраните запрос, закройте окно конструктора, выполните запрос из окна БД.
- Просмотрите личные карточки нескольких сотрудников.

3. Для создания перекрестного запроса на расчет среднего оклада по подразделениям:

- создайте предварительный простой запрос в составе трех полей: *Подразделение* (из таб. ***Подразделение***), *Должность* (из таб. ***Должность***), *Оклад* (из таб. ***Сотрудники***), задайте ему имя ***Предварительный запрос***, просмотрите его результаты и закройте;
- в окне БД для объекта **Запросы** щелкните на кнопке <Создать>, выберите из списка **Перекрестный запрос**, <ОК>;
- в окне мастера перекрестного запроса в области **Показать** установите переключатель на опцию **Запросы**, выберите из предлагаемого списка запросов **Предварительный запрос**, <Далее>;
- в окне **Создание перекрестных таблиц** в области **Доступные поля** выберите поле *Подразделение* – для обозначения строк перекрестного запроса, перенесите его в область **Выбранные поля**, <Далее>;
- выделите поле *Должность* – для обозначения столбцов запроса, <Далее>;
- в списке статистических функций выберите функцию **Среднее**, <Далее>;
- задайте имя запроса ***Средний оклад***, <Готово>.

4. По технологии создания отчета, описанной в примере 1 (БД Кадры), самостоятельно разработайте отчет Картотека на основе запроса Картотека. Задайте группировку данных по полю Подразделение и расчет минимального, максимального и среднего значений Оклада, отчет разбейте на страницы таким образом, чтобы каждая группа записей, относящихся к одному подразделению, вместе с ее итоговыми значениями выводилась на отдельной странице.

2.9 Лабораторная работа №10 (2 часа).

Тема: «Разработка информационно-логической модели реляционной базы данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.9.1 Цель работы: изучить и проанализировать разработку информационно-логической модели реляционной базы данных

2.9.2 Задачи работы:

1. Изучение видов связей между таблицами
2. Изучение целостность данных

2.9.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.9.4 Описание (ход) работы:

Разработка информационно-логической модели реляционной базы данных.

Слово "реляционная" происходит от английского *relation* - отношение. *Отношение* - тематическое понятие, но в терминологии моделей данных отношения удобно изображать в виде таблицы. При этом строки таблицы соответствуют кортежам отношения, а столбцы - атрибутам. Ключом называют любую функцию от атрибутов кортежа, которая может быть использована для идентификации кортежа. Такая функция может быть значением одного, из атрибутов (простой ключ), задаваться алгебраическим выражением, включающим значения нескольких атрибутов (составной ключ). Это означает, что данные в строках каждого из столбцов составного ключа могут повторяться, но комбинация данных каждой строки этих столбцов является уникальной. Например, в таблице Студенты есть столбцы *Фамилии* и *Год рождения.* В каждом из столбцов есть некоторые повторяющиеся данные, т.е. одинаковые фамилии и одинаковые года рождения. Но если студенты, имеющие одинаковые фамилии, имеют разные года рождения, то эти столбцы можно использовать в качестве составного ключа. Как правило, ключ является уникальным, т.е. каждый кортеж определяется значением ключа однозначно, но иногда используют и неуникальные ключи (ключи с повторениями). В локализованной (русифицированной) версии Access вводится термин *ключевое поле,* которое можно трактовать как *первичный ключ.*

В Access можно выделить три типа ключевых полей: простой ключ, составной ключ и внешний ключ.

Одно из важнейших достоинств реляционных баз данных состоит в том, что вы можете хранить логически сгруппированные данные в разных таблицах и задавать связи между ними, объединяя их в единую базу. Для задания связи таблицы должны иметь поля с одинаковыми именами или хотя бы с одинаковыми форматами данных. Связь между

таблицами устанавливает отношения между совпадающими значениями в этих полях. Такая организация данных позволяет уменьшить избыточность хранимых данных, упрощает их ввод и организацию запросов и отчетов. Поясним это на примере. Допустим, вам в базе надо хранить, данные о студентах (фамилия, изучаемая дисциплина) и преподавателях (фамилия, номер кафедры, ученая степень, преподаваемая дисциплина). Если хранить данные в одной таблице, то в строке с фамилией студента, изучающего конкретную дисциплину, будут храниться все атрибуты преподавателя, читающего эту дисциплину. Это же огромная избыточность данных. А если хранить данные о студенте в одной таблице, о преподавателе - в другой и установить связь между полями "Читаемая дисциплина" - "Изучаемая дисциплина" (фактически это одинаковые поля), то избыточность хранимых данных многократно уменьшится без ущерба для логической организации информации.

В Access можно задать три вида связей между таблицами; *Один-ко-многим,, Многие-ко-многим* и *Один-к-одному.*

Связь *Один-ко-многим* - наиболее часто используемый тип связи между таблицами. В такой связи каждой записи в таблице А может соответствовать несколько записей в таблице В (поля с этими записями называют*внешними ключами),* а запись в таблице В не может иметь более одной соответствующей ей записи в таблице А.

При связи *Многие-ко-многим* одной записи в таблице А может соответствовать несколько записей в таблице В, а одной записи в таблице В - несколько записей в таблице А. Такая схема реализуется только с помощью третьей (связующей) таблицы, ключ которой состоит по крайней мере из двух полей, одно из которых является общим с таблицей А, а другое - общим с таблицей В.

При связи *Один-к-одному* запись в таблице А может иметь не более одной связанной записи в таблице В и наоборот. Этот тип связи используют не очень часто, поскольку такие данные могут быть помещены в одну таблицу. Связь с отношением *Один-к-одному* применяют для разделения очень широких таблиц, для отделения части таблицы в целях ее защиты, а также для сохранения сведений, относящихся к подмножеству записей в главной таблице.

Тип создаваемой связи зависит от полей, для которых определяется связь:

- связь *Один-ко-многим* создается в том случае, когда только одно из полей является ключевым или имеет уникальный индекс, т.е. значения в нем не повторяются;

- связь *Один-к-одному* создается в том случае, когда оба связываемых поля являются ключевыми или имеют уникальные индексы;

- связь *Многие-ко-многим* фактически представляет две связи типа *один-ко-многим* через третью таблицу, ключ которой состоит, по крайней мере, из двух полей, общих для двух других таблиц.

Целостность данных

Целостность данных означает систему правил, используемых в СУБД Access для поддержания связей между записями в связанных таблицах (таблиц, объединенных с помощью связи), а также обеспечивает защиту от случайного удаления или изменения связанных данных. Контролировать целостность данных можно, если выполнены следующие условия:

связанное поле (поле, посредством которого осуществляется связь) одной таблицы является ключевым полем или имеет уникальный индекс;

связанные поля имеют один тип данных. Здесь существует исключение. Поле счетчика может быть связано с числовым полем, если оно имеет тип *Длинное целое,*

обе таблицы принадлежат одной базе данных Access. Если таблицы являются связанными, то они должны быть таблицами Access. Для установки целостности данных база данных, в которой находятся таблицы, должна быть открыта. Для связанных таблиц из баз данных других форматов установить целостность данных невозможно.

ЗАДАНИЕ 1

Создание инфологической и логической моделей базы данных.

1. Разработайте информационно-логическую модель реляционной базы данных.
2. Разработайте логическую модель реляционной базы данных

ТЕХНОЛОГИЯ РАБОТЫ

1. Перед разработкой информационно-логической модели реляционной базы данных рассмотрим, из каких информационных объектов должна состоять эта база данных. Можно выделить три объекта, которые не будут обладать избыточностью, - *Студенты, Дисциплины* и *Преподаватели.* Представим состав реквизитов этих объектов в виде "название объекта (перечень реквизитов)": *Студенты* (код студента, фамилия, имя, отчество, номер группы, дата рождения, стипендия, оценки). *Дисциплины* (код дисциплины, название дисциплины), *Преподаватели* (код преподавателя, фамилия, имя, отчество, дата рождения, телефон, заработная плата).

Рассмотрим связь между объектами *Студенты* и *Дисциплины*. Студент изучает несколько дисциплин, что соответствует многозначной связи и отражено на рис. 1 двойной стрелкой. Понятно, что каждая дисциплина изучается множеством студентов. Это тоже многозначная связь, обозначаемая двойной стрелкой (связь "один" обозначена одинарной стрелкой). Таким образом, связь между объектами *Студенты* и *Дисциплины -Многие-ко-многим* (M : N).

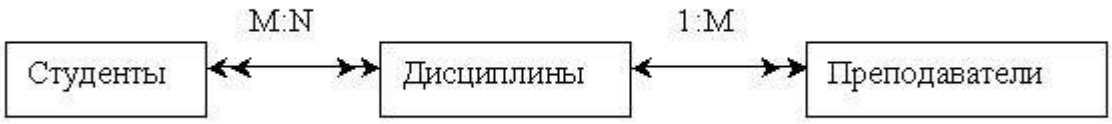


Рис. 1. Типы связей между объектами Студенты, Дисциплины и Преподаватели.

Множественные связи усложняют управление базой данных, например, в СУБД Access при множественных связях нельзя использовать механизм каскадного обновления. Поэтому использовать такие связи нежелательно и нужно строить реляционную модель, не содержащую связей типа Многие-ко-многим. В Access для контроля целостности данных с возможностью каскадного обновления и удаления данных необходимо создать вспомогательный объект связи, который состоит из ключевых реквизитов связываемых объектов и который может быть дополнен описательными реквизитами. В нашем случае таким новым объектом для связи служит объект Оценки, реквизитами которого являются код студента, код дисциплины и оценки. Каждый студент имеет оценки по нескольким дисциплинам, поэтому связь между объектами Студенты и Оценки будет Один-ко-многим (1:M). Каждую дисциплину сдает множество студентов, поэтому связь между объектами Дисциплины и Оценки также будет Один-ко-многим (1:M). В результате получаем информационно-логическую модель базы данных, приведенную на рис. 2.

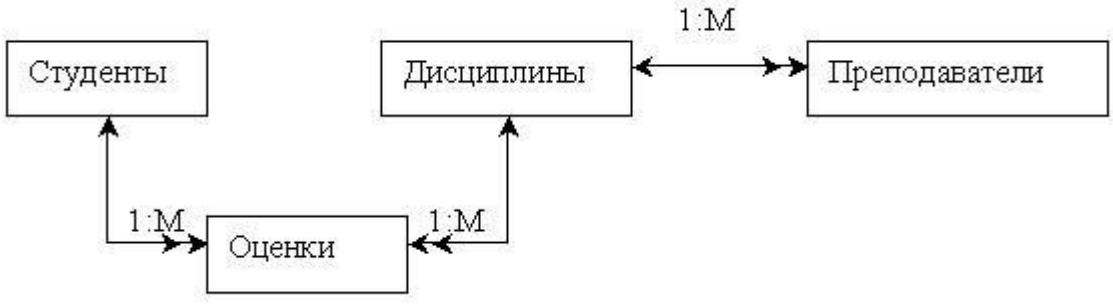


Рис. 2. Информационно-логическая модель реляционной базы данных

2. В реляционной базе данных в качестве объектов рассматриваются отношения, которые можно представить в виде таблиц. Таблицы между собой связываются посредством

общих полей, т.е. одинаковых по форматам и, как правило, по названию, имеющихся в обеих таблицах. Рассмотрим, какие общие поля надо ввести в таблицы для обеспечения связности данных. В таблицах *Студенты* и *Оценки* таким полем будет "Код студента", в таблицах *Дисциплины* и *Оценки* - "Код дисциплины", в таблицах *Преподаватели* и *Дисциплины* - "Код дисциплины". Выбор цифровых кодов вместо фамилий или названий дисциплин обусловлен меньшим объемом информации в таких полях: например, число "2". по количеству символов значительно меньше слова "математика". В соответствии с этим логическая модель базы данных представлена на рис. 3, где жирными буквами выделены ключевые поля.

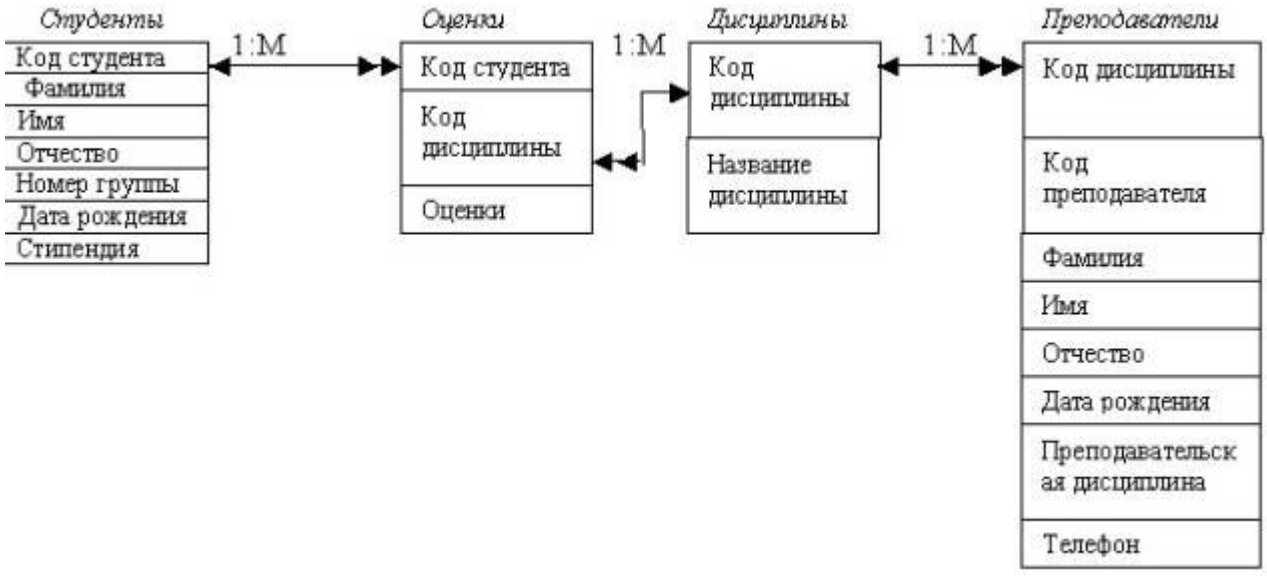


Рис. 3. Логическая модель базы данных

ЗАДАНИЕ 2

Создание реляционной базы данных.

1.Создайте базуданных *Деканат.*

2.Создайте структуру таблицы *Студенты.*

3.Создайте структуру таблицы *Дисциплины.*

4. Измените структуру таблицы *Преподаватели.*

5. Создайте структуру таблицы *Оценки.*

6. Разработайте схему данных, т.е. создайте связи между таблицами.

ТЕХНОЛОГИЯ РАБОТЫ

1. Создайте базу *данных Деканат,* выполнив следующие действия:

- загрузите Access, в появившемся окне выберите пункт *Новая база данных,* затем щелкните по кнопке <ОК>;

- в окне <Файл новой базы данных> задайте имя (пункт *Имя файла)* и выберите папку (пункт *Папка),* где ваша база будет находиться. По умолчанию Access предлагает имя базы *dbl,* а тип файла - *Базы данных Access.*Имя задайте *Деканат,* а тип файла оставьте прежним, так как другие типы файлов нужны в специальных случаях;

- щелкните по кнопке <Создать>

2. Создайте структуру таблицы *Студенты.* Для этого:

- в окне базы данных выберите вкладку *Таблицы,* а затем щелкните по кнопке <Создать>;

- в окне "Новая таблица" выберите пункт *Конструктор* и щелкните по кнопке <ОК>. В результате проделанных операций открывается окно таблицы в режиме конструктора, в котором следует определить поля таблицы;

- определите поля таблицы в соответствии с табл. 1;

Таблица 1

Имя поля	Тип данных	Размер поля
Код студента	Числовой	Целое
Фамилия	Текстовый	15
Имя	Текстовый	12
Отчество	Текстовый	15
Номер группы	Числовой	Целое
Телефон	Текстовый	9
Стипендия	Логический	Да/Нет

- в качестве ключевого поля задайте "Код студента". Для этого щелкните по полю "Код студента" и по кнопке на панели инструментов или выполните команду **Правка, Ключевое поле**;

- закройте таблицу, задав ей имя *Студенты*.

Примечание. Заполнять таблицу данными пока не требуется, это будет сделано в режиме формы.

3. Создайте структуру *таблицы Дисциплины* аналогично п. 2 в соответствии с табл. 2.

Таблица 2

Имя поля	Тип данных	Размер поля
Код дисциплины	Числовой	Целое
Название дисциплины	Текстовый	30

В качестве ключевого поля задайте "Код дисциплины". Заполняться эта таблица будет также в режиме формы.

4. Структура таблицы *Преподаватели* уже создана в работе 1 и заполнена данными, этому для работы используйте эту таблицу с одним лишь изменением - в соответствии с рис. 4.11 в структуру таблицы надо добавить поле "Код дисциплины" и заполнить его в соответствии с данными табл. 4.4.

5. Создайте структуру таблицы *Оценки* аналогично п. 2 в соответствии с табл. 3.

Таблица 3

Имя поля	Тип данных	Размер поля
Код студента	Числовой	Целое
Код дисциплины	Числовой	Целое
Оценки	Числовой	Байт

В этой таблице задавать ключевое поле не надо, так как данные во всех полях могут повторяться. Эта таблица, аналогично предыдущим, будет заполняться в режиме формы.

6. Разработайте схему данных, т.е. создайте связи между таблицами. Для этого:

- щелкните по кнопке на панели инструментов или выполните команду **Сервис, Схема данных**. На экране появится окно "Схема данных";

- щелкните по кнопке на панели инструментов или выполните команду **Связи, Добавить таблицу**;

- в появившемся окне будет выделено название одной таблицы. Щелкните по кнопке <Добавить>;

- переведите выделение на имя следующей таблицы и щелкните по кнопке <Добавить>. Аналогично добавьте оставшиеся две таблицы;

- закройте окно, щелкнув по кнопке <Закрыть>;

- создайте связь между таблицами *Дисциплины* и *Оценки.* Для этого подведите курсор мыши к полю "Код дисциплины" в таблице *Дисциплины* щелкните левой кнопкой мыши и, не отпуская ее, перетащите курсор на поле "Код дисциплины" в таблицу *Оценки,* а затем отпустите кнопку мыши. На экране откроется окно "Связи";

- установите флажок ("галочку") в свойстве *Обеспечение целостности данных,* щелкнув по нему;

- установите флажок в свойстве *Каскадное обновление связанных полей* и *Каскадное удаление связанных записей;*

Примечание. Задание каскадного обновления связанных полей и каскадного удаления связанных записей позволит вам отредактировать записи только в *таблице Дисциплины,* а в таблице *Оценки* эти действия будут со связанными записями выполняться автоматически. Например, если вы удалите из таблицы *Дисциплины* один предмет, то в таблице *Оценки* удалятся все строки, связанные с этим предметом.

- щелкните по кнопке <Создать>. Связь будет создана;

- аналогично создайте связи между полем "Код дисциплины" в *таблице Дисциплины* и полем "Код дисциплины" в таблице *Преподаватели,* а также между полем "Код студента" в таблице *Студенты* и полем "Код студента" в таблице *Оценки.* Результат представлен на рис. 4;

- закройте окно схемы данных, ответив *ДА* на вопрос о сохранении макета.

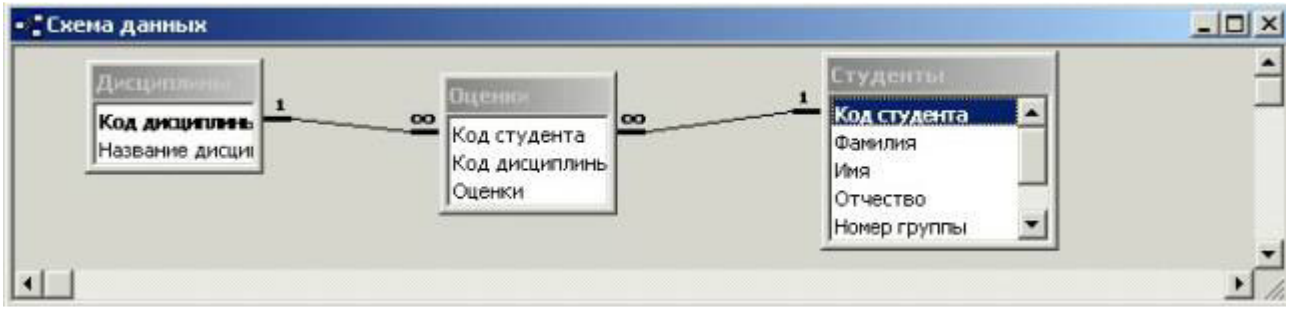


Рис. 4. Структура таблицы *Студенты*

2.10 Лабораторная работа №11 (2 часа).

Тема: «Основы языка SQL»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.10.1 Цель работы: изучить и проанализировать основы языка программирования SQL

2.10.2 Задачи работы:

1. Изучение термина «Структурированный Язык Запросов»
2. Изучение использование SQL Management Studio

2.10.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер

2. Microsoft Office Access

2.10.4 Описание (ход) работы:

Основы языка SQL.

Постепенная унификация баз данных привела к необходимости создания стандартного языка доступа к данным БД, который мог бы использоваться для функционирования в большом количестве различных видов компьютерных сред. Стандартный язык позволит пользователям, знающим один набор команд, применять их, чтобы создавать, отыскивать, изменять и передавать информацию независимо от используемого сервера баз данных и его местоположения.

SQL (Structured Query Language), или Структурированный Язык Запросов, - это язык, который дает возможность работать с данными в реляционных базах данных. Стандарт SQL определяется ANSI (Американским Национальным Институтом Стандартов), а также ISO (Международной организацией по стандартизации). Однако большинство коммерческих программ баз данных расширяют SQL, добавляя разные особенности в этот язык, которые, как они считают, будут полезны. Эти дополнения являются не стандартизированными и часто приводят к сложностям при переходе от одного сервера данных к другому.

Для обращения к базе данных используются запросы, написанные на языке SQL. Запросом называется команда, которая передается серверу базы данных, и которая сообщает ему, что нужно вывести определенную информацию из таблиц в память. Эта информация обычно посылается непосредственно на экран компьютера или терминала, хотя в большинстве случаев ее можно также передать на принтер, сохранить в файле (как объект в памяти компьютера) или представить как вводную информацию для другой команды или процесса.

Несмотря на большое количество разновидностей этого языка, существующих в настоящее время, логика его работы проста. Достаточно освоить основные команды хотя бы в одной из его версий, чтобы впоследствии без труда разобраться в любой другой его реализации. Для выполнения SQL-запросов будем использовать SQL Management Studio. При запуске среды Management Studio появляется следующее окно (рис. 1).

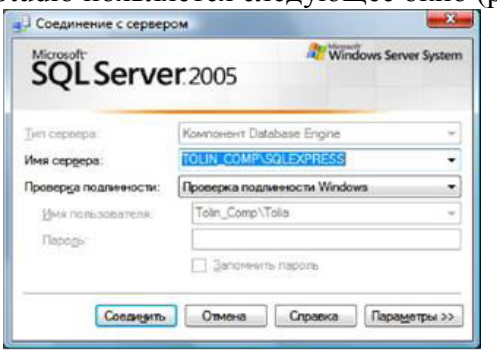


Рис. 1. Задание параметров подключения

Задействуем подключение к серверу, находящемуся на локальном компьютере. Параметр «Проверка подлинности» задает аутентификацию при подключении к серверу - при выбранном значении «Проверка подлинности Windows» (Windows authentification) в качестве имени пользователя и пароля будут использованы системные параметры.

Если все сделано правильно, то появляется главное окно программы. Для перехода в режим запросов необходимо нажать кнопку «Создать запрос» (рис. 2).

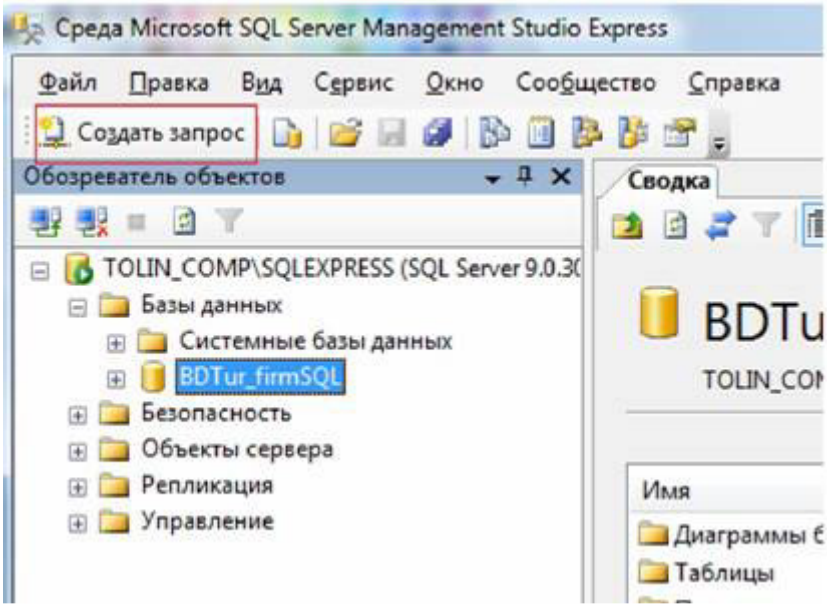


Рис. 2. Переход в режим создания запросов

Отметим, что будут создаваться запросы, работающие с выделенной базой данных. После нажатия кнопки «Создать запрос» среда SQL Management Studio принимает вид, как показано на рисунке (рис. 3). Обратите внимание на кнопку «Выполнить», которая выполняет запросы, введенные в правом текстовом поле, и выводит результат их выполнения.

Основной операцией для описания запроса к БД в языке SQL является конструкция вида:

Select <список атрибутов>
From <список отношений>
Where <условие>

Эта операция представляет собой композицию реляционных операторов проекции, соединения и выбора. Проекция берется для указанного списка атрибутов, соединение выполняется для указанного списка отношений, выбор определяется условием отбора записей where.

В результате выполнения операции соединения данные из указанных в списке отношений представляются одной таблицей. В этой таблице из всех имеющихся столбцов исходных отношений списка отношений остаются только те столбцы, которые указаны в списке атрибутов, и только те строки, которые удовлетворяют условию where.

Итак, напишем первый запрос и нажмем клавишу F5 (пункт меню Запрос - Выполнить):

select * from Туристы;

В результате возвращаются все записи из таблицы «Туристы» базы данных BDTur_firmSQL.

Главное окно программы принимает вид (рис. 3).

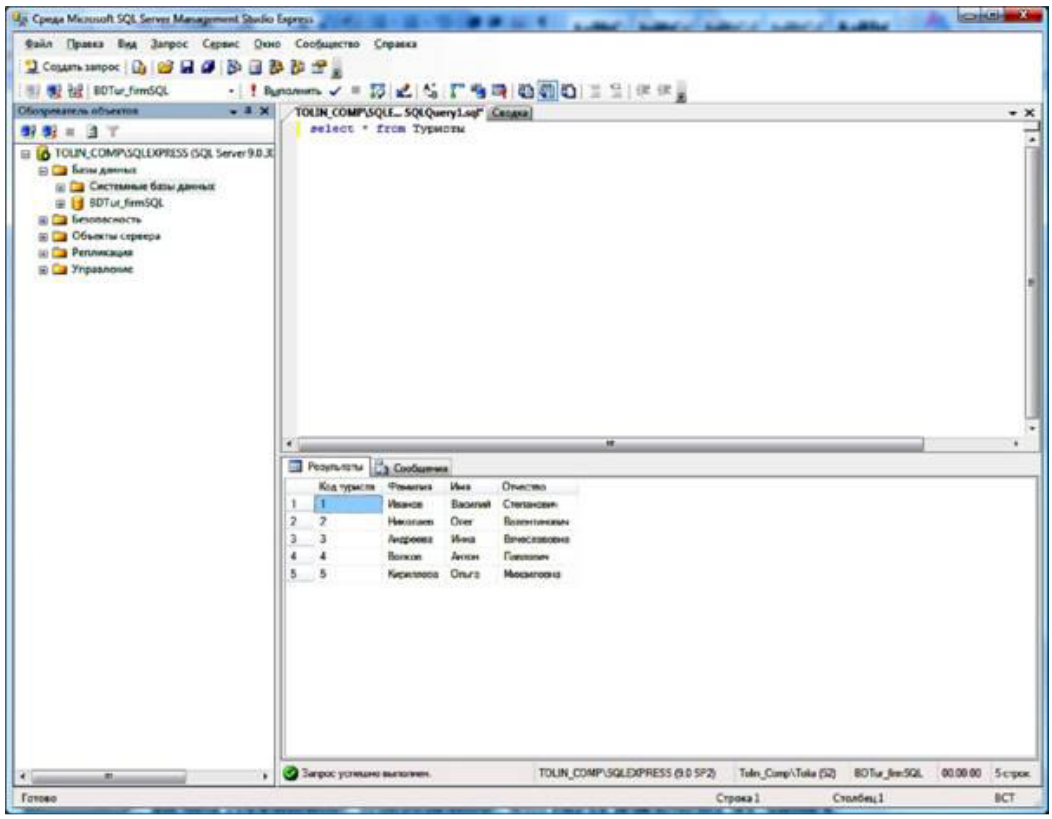


Рис. 3. Извлечение строк из таблицы «Туристы»

Данный запрос извлекал все столбцы таблицы. Если необходимо выбрать только столбец «Фамилия», запрос необходимо модифицировать следующим образом:

select Фамилия from Туристы;

Результат выполнения данного запроса представлен на рисунке 4.

Базовая конструкция SQL-запроса

Для вывода определенного количества записей используем следующий запрос (рис. 5):

select top 3 Фамилия from Туристы;

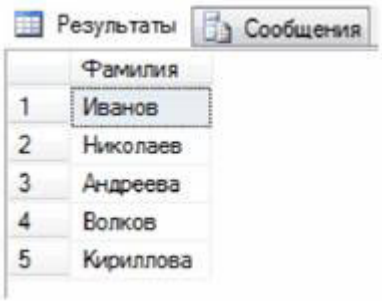


Рис. 4. Извлечение столбца «Фамилия»

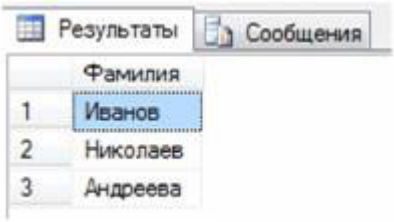


Рис. 5. Извлечение заданного количества записей

Извлекаются первые три записи поля «Фамилия», расположенные в самой таблице «Туристы». Обратим внимание на то, что фамилии расположены не в алфавитном порядке, а в порядке, в котором они были сохранены в базе данных.

Добиться алфавитного порядка можно с помощью предложения order by, содержащего список атрибутов, после каждого из которых стоит либо ключевое слово asc

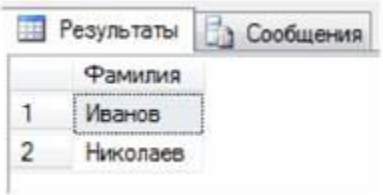
(сортировка по возрастанию), либо ключевое слово dec (сортировка по убыванию). Теперь предыдущий запрос может выглядеть так:

select top 3 Фамилия from Туристы order by Фамилия asc;

Вводя оператор percent, можем получить указанный процент записей от общего числа:

select top 25 percent Фамилия from Туристы;

Результат выполнения запроса представлен на рисунке 6.


Результаты | Сообщения

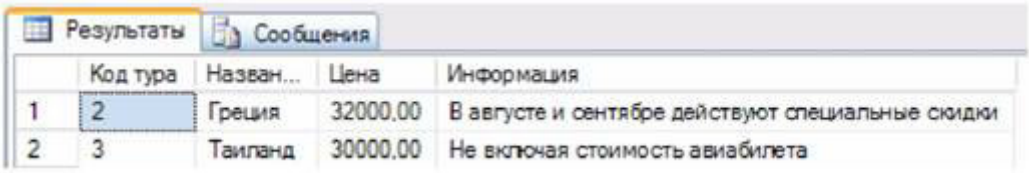
	Фамилия
1	Иванов
2	Николаев

Рис. 6. Извлечение нескольких записей

Для отбора записей, отвечающих заданному условию, используем оператор where:

select * from Туры where Цена > 27000;

Этот запрос возвращает все записи из таблицы «Туры», в которых поле «Цена» имеет значение, большее 27000 (рис. 7).


Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	2	Греция	32000,00	В августе и сентябре действуют специальные скидки
2	3	Таиланд	30000,00	Не включая стоимость авиабилета

Рис. 7. Отбор записей со всеми полями по заданному значению

Оператор where поддерживает работу со знаками сравнения <, >, >=, <=. Точную выборку только из заданного множества значений осуществляет оператор in, в следующем примере извлекаются лишь те записи, в которых значение поля «Цена» в точности равно либо 10 000, либо 20 000, либо 30 000 (рис. 8):

select * from Туры where Цена in (10000, 20000, 30000);

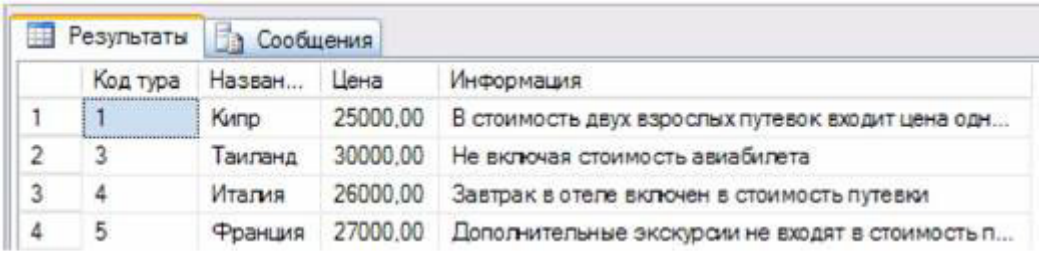

Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	3	Таиланд	30000,00	Не включая стоимость авиабилета

Рис. 8. Отбор записей по точному совпадению значений поля Цена

Выборка значений, лежащих в указанном интервале, осуществляется оператором between <первое_значение> and <второе_значение> (рис. 9):

Select * from Туры where Цена between 10000 and 30000;


Результаты | Сообщения

	Код тура	Назван...	Цена	Информация
1	1	Кипр	25000,00	В стоимость двух взрослых путевок входит цена одн...
2	3	Таиланд	30000,00	Не включая стоимость авиабилета
3	4	Италия	26000,00	Завтрак в отеле включен в стоимость путевки
4	5	Франция	27000,00	Дополнительные экскурсии не входят в стоимость п...

Рис. 9. Отбор записей по значениям в указанном интервале поля Цена

Агрегирующие функции языка SQL

При статистическом анализе баз данных необходимо получать такую информацию, как общее количество записей, наибольшее и наименьшее значения заданного поля записи, усредненное значение поля и т. д. Данная задача выполняется с помощью запросов, содержащих так называемые агрегирующие функции.

Агрегирующие функции производят вычисление одного «собирающего» значения (суммы, среднего, максимального, минимального значения и т. п.) для заданных групп строк таблицы. Группы строк определяются различными значениями заданного поля (полей) таблицы. Разбиение на группы выполняется с помощью предложения group by.

Рассмотрим перечень агрегирующих функций.

- count определяет количество записей данного поля в группе строк.
- sum вычисляет арифметическую сумму всех выбранных значений данного поля.
- avg рассчитывает арифметическое среднее (усреднение) всех выбранных значений данного поля.
- max находит наибольшее из всех выбранных значений данного поля.
- min находит наименьшее из всех выбранных значений данного поля.

Для определения общего числа записей в таблице Туристы используем запрос

select count (*) from Туристы;

Результат выполнения запроса представлен на рисунке 10.

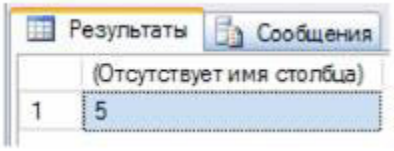


Рис. 10. Результат запроса с функцией count

Отметим, что результатом запроса является одно число, содержащееся в поле с отсутствующим именем.

А если мы захотим посчитать однофамильцев (то есть разбить набор записей-результатов запроса на группы с одинаковыми фамилиями), то запрос будет выглядеть так:

select Фамилия, count (Фамилия) from Туристы group by Фамилия;

Синтаксис использования других операторов одинаков - следующие запросы извлекают сумму, арифметическое среднее, наибольшее и наименьшее значения поля «Цена» таблицы «Туры» (здесь заданной группой записей, как и в первом примере с функцией count, являются все записи таблицы).

select sum(Цена) from Туры
select avg(Цена) from Туры
select max(Цена) from Туры
select min(Цена) from Туры

Если значение поля может быть незаполненным, то для обращения к таким полям необходимо использовать оператор null. Отметим, что величина null не означает, что в поле стоит число 0 (нуль) или пустая текстовая строка. Существует два способа образования таких значений:

1) Microsoft SQL Server автоматически подставляет значение null, если в значение поля не было введено никаких значений и тип данных для этого поля не препятствует присвоению значения null;

2) или если пользователь явным образом вводит значение null.

Оператор сравнения записей like

Оператор сравнения like нужен для поиска записей по заданному шаблону. Эта задача является одной из наиболее часто встречаемых задач - например, поиск клиента с известной фамилией в базе данных.

Предположим, что в таблице «Туристы», содержащей поля «Фамилия», «Имя» и «Отчество», требуется найти записи клиентов с фамилиями, начинающимися на букву «И».

select Фамилия, Имя, Отчество from Туристы
where Фамилия Like 'И%'

Результатом этого запроса будет таблица, представленная на рисунке 11. Оператор like содержит шаблоны, позволяющие получать различные результаты (таблица 1).

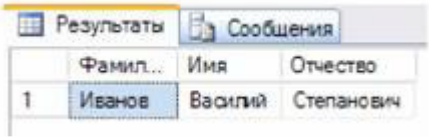


Рис. 11. Запрос с оператором like

Таблица 1 Шаблоны оператора like

Шаблон	Значение
like '5[%]'	5%
like '[ ]n'	n
like '[a-cdf]'	a, b, c, d, или f
like '[-acdf]'	-, a, c, d, или f
like '[[ ]'	[
like ']'	]
like 'abc[ ]d%'	abc d и abc de
like 'abc[def]'	abcd, abce, и abcf

Команды определения данных языка SQL

Пока мы познакомились только с работой некоторых команд языка SQL по извлечению таблиц и данных из таблиц, предполагая, что сами таблицы были созданы кем-то ранее. Это наиболее реальная ситуация, когда небольшая группа людей (проектировщики баз данных) создает таблицы, которые затем используются другими людьми. Эти команды относятся к области SQL, называемой DML (Data Manipulation Language, или Язык Манипулирования Данными). Тем не менее существует специальная область SQL, называемая DDL (Data Definition Language, или Язык Определения Данных), которая специально работает над созданием объектов данных.

Таблицы создаются командой create table. Эта команда создает пустую таблицу. Команда create table задает имя таблицы, столбцы таблицы в виде описания набора имен столбцов, указанных в определенном порядке, а также она может определять главный и вторичные ключи таблицы. Кроме того, она указывает типы данных и размеры столбцов. Каждая таблица должна содержать, по крайней мере, один столбец.

Пример команды create table:

create table ClientInfo (FirstName varchar(20), LastName varchar(20), Address varchar(20), Phone varchar(15)

Тип varchar предназначен для хранения символов не в кодировке Unicode. Число, указываемое в скобках, определяет максимальный размер поля и может принимать значение от 1 до 8000. Если введенное значение поля меньше зарезервированного, при сохранении будет выделяться количество памяти, равное длине значения. После выполнения этого запроса в окне «Сообщения» появляется сообщение: Команды выполнены успешно.

После перезапуска Management Studio в списке таблиц появилась новая таблица (рис. 12).

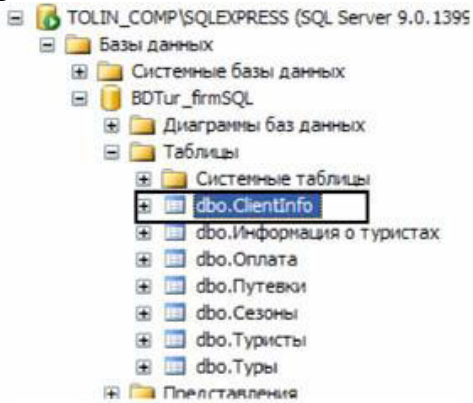


Рис. 12. Созданная таблица в базе данных

Итак, была создана таблица, состоящая из четырех полей типа varchar, причем для трех полей была определена максимальная длина 20 байт, а для одного - 15. Значения полей не заполнены - на это указывает величина Null.

Можно удалить созданную таблицу непосредственно в интерфейсе Management Studio, щелкнув правой кнопкой и выбрав «Удалить».

Команды изменения данных языка DML

Значения могут быть помещены и удалены из полей тремя командами языка DML (Язык Манипулирования Данными):

- insert (вставить),
- update (изменить),
- delete (удалить).

Команда insert имеет свои особенности.

- При указании значений конкретных полей вместо использования каких- либо значений можно применить ключевое слово DEFAULT.
- Вставка пустой строки приводит к добавлению пробела, а не значения NULL.
- Строки и даты задаются в апострофах.
- Можно задавать NULL явно, а можно задавать DEFAULT.

Например:

insert into ClientInfo (FirstName, LastName, Address, Phone) values('Petr','Petrov','Chehova 13','1234567');

Однократное выполнение этого запроса (нажатие клавиши F5 один раз) приводит к добавлению одной записи. Добавим еще несколько записей, изменяя значения values:

insert into Clientlnfo (FirstName, LastName, Address, Phone) values('Ivan','Ivanov','Naberejnaya 13,,,1234568');
insert into Clientlnfo (FirstName, LastName, Address, Phone) values(null,'Sidorov','Naberejnaya 25','1234569');

Извлечем все записи созданной таблицы (рис. 13):

select * from Clientlnfo;

Результаты | Сообщения

	FirstName	LastName	Address	Phone
1	Petr	Petrov	Chehova 13	1234567
2	Ivan	Ivanov	Naberejnaya 13	1234568
3	NULL	Sidorov	Naberejnaya 25	1234569

Рис. 13. Все записи таблицы Clientlnfo

Отметим, что третья строка содержит значение null, а не текстовую строку «null».

Команда update позволяет изменять заданные значения записей:

update Clientlnfo set FirstName = 'Andrey' where FirstName = 'Petr';

В этом случае в первой записи поля FirstName значение Petr изменится на Andrey (рис. 14).

Результаты | Сообщения

	FirstName	LastName	Address	Phone
1	Andrey	Petrov	Chehova 13	1234567
2	Ivan	Ivanov	Naberejnaya 13	1234568
3	NULL	Sidorov	Naberejnaya 25	1234569

Рис. 14. Изменение одной записи

Отметим, что если не указывать условие, определяющее значение, которое необходимо изменить, команда update затронет все записи.

Команда delete удаляет записи из таблицы.

delete from Clientlnfo where LastName like 'Petrov';

Результатом этого запроса будет удаление первой записи из таблицы ClientInfo.

Если не задавать условие, определяющее данные, которые необходимо удалить, то будут удалены все данные таблицы.

Запросы с командами insert, update и delete могут содержать в себе все прочие конструкции языка SQL.

2.11 Лабораторная работа №12 (2 часа).

Тема: «Создание сложных запросов. Подзапросы, теоретико-множественные операции, операции соединения»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.11.1 Цель работы: изучить и проанализировать создание сложных запросов, подзапросов, теоретико-множественных операций, операций соединения

2.11.2 Задачи работы:

1. Изучение создания сложных запросов
2. Изучение создания подзапросов, теоретико-множественных операций, операций соединения
3. Изучение создания теоретико-множественных операций
4. Изучение создания операций соединения

2.11.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.11.4 Описание (ход) работы:

Создание сложных запросов. Подзапросы, теоретико-множественные операции, операции соединения.

1. Разработайте запрос для расчета ведомости зарплаты (расчет надбавок к окладу и удержаний) с вычисляемыми полями *Стаж*, *Надбавка*, *Налог* (согласно данным ЛР-7). Стаж работы сотрудника в данной организации рассчитывается на основе **текущей даты** и **даты приема на работу**. Текущая дата определяется функцией **Date**. В зависимости от стажа работнику начисляется надбавка к окладу в размере **10%** - если стаж менее 5 лет и **20%** - если стаж не менее 5лет. Налог рассчитывается в размере **13%** от суммы оклада и надбавки. Для создания запроса ***Начисление зарплаты*** выполните следующие действия:
    - создайте простой запрос с полями: *ТабНомер, Фамилия, КодПодразделения, КодДолжности* (из таб. ***Сотрудники***), *ДатаПриема* (из таб. ***Движение кадров***), *Оклад* (из таб. ***Сотрудники***);
    - задайте имя запроса ***Начисление зарплаты***, перейдите в режим конструктора;
    - в поле *Фамилия* в строке **Сортировка** установите – **По возрастанию**;
    - в конструкторе выделите столбец, соответствующий полю *Оклад*, выберите в меню команду **Вставка/Столбцы** (слева от поля *Оклад* появляется пустой столбец);

- в строке **Поле** пустого столбца откройте **Построитель выражений**, в окне построителя сформируйте формулу для расчета стажа работы:

(Date()-[Движение кадров]![ДатаПриема])/365

формулу можно ввести с клавиатуры, можно воспользоваться средствами построителя выражений - для этого: щелкните на кнопке (, в списке папок откройте папку **Функции** – **Встроенные функции**, в категориях функций выберите **Дата/Время**, выберите функцию **Date**, <Вставить>,щелкните на кнопке -, в списке папок откройте папку **Таблицы** – **Движение кадров**, из списка полей таблицы выберите поле **ДатаПриема**, <Вставить>, щелкните на кнопке), затем на кнопке /, введите с клавиатуры значение **365**. В результате в окне построителя выражений должна быть записана формула:

Выражение1:(Date()-[Движение кадров]![ДатаПриема])/365

- вернитесь в окно конструктора запросов кнопкой <ОК>, запустите запрос на выполнение, проверьте, как рассчитывается стаж работы;
- снова перейдите в режим конструктора, в свободных столбцах справа от поля *Оклад* аналогично сформируйте еще два вычисляемых поля для расчета **надбавки** к окладу и **налога**, для расчета используйте следующие выражения:

IIf ([Стаж] < 5; [Оклад]*0,1; [Оклад]*0,2) **-** надбавка

([Надбавка] + [Оклад]) * 0,13 **-** налог;

- в режиме конструктора замените имена вычисляемых полей **Выражение1, Выражение2, Выражение3** на имена *Стаж, Надбавка, Налог* соответственно.

Начисление зарплаты : запрос на выборку

ТабНомер	Фамилия	ДатаПриема	КодПодразд	КодДолжности	Стаж	Оклад	Надбавка	Налог
115	Гришин	03-фев-05	2002	1003	0.96438356164	7850.00	785	1122.55
106	Завьялова	26-окт-00	2003	1001	5.24109589041	25000.00	5000	3900
111	Захарова	17-дек-03	2003	1002	2.09863013699	10000.00	1000	1430
101	Иванцев	24-апр-97	2001	1001	8.75068493151	25000.00	5000	3900

2. Для создания запроса по движению кадров выполните следующие действия:
   - создайте простой запрос, в который выберите поля: *ТабНомер, Фамилия, Имя, Отчество* (из таб. ***Сотрудники***), *ДатаПриема, Движение, ДатаПеревода, ДатаУвольнения* (из таб. ***Движение кадров***);
   - задайте имя запроса ***Запрос по движению***, перейдите в режим конструктора;
   - в конструкторе запроса в поле *Движение* в строке **Условие отбора** введите значение **уволен**, в строке **Или** введите значение **переведен** (такая запись условия аналогична выражению **«уволен» Or «переведен»**;
   - сохраните запрос, закройте окно конструктора и запустите запрос из окна БД.
   - Самостоятельно разработайте **параметрический** запрос на получение данных об **уволенных** или **переведенных** сотрудниках.

Самостоятельно разработайте запрос на удаление из БД записей об уволенных сотрудниках.

2.12 Лабораторная работа №13 (2 часа).

Тема: «Создание сложной формы, создание подчиненной формы»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.12.1 Цель работы: изучить и проанализировать создание сложной формы, создание подчиненной формы

2.12.2 Задачи работы:

1. Изучение создания сложной формы
2. Изучение создания подчиненной формы

2.12.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

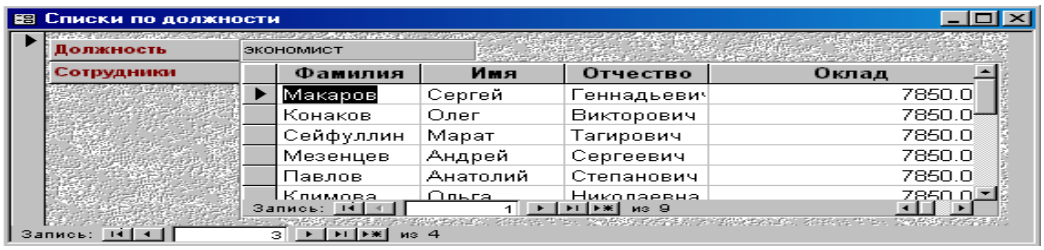
2.12.4 Описание (ход) работы:

Создание сложной формы, создание подчиненной формы.

1. Для разработки подчиненной формы, отображающей информацию о сотрудниках по наименованиям подразделений (или должностям) воспользуемся одним из двух способов создания сложных форм (согласно данным ЛР-7):
   - создайте главную форму для таблицы ***Подразделение*** в режиме **Автоформа в один столбец**, задайте ей имя ***Списки по подразделениям***;
   - создайте с помощью мастера форму (**ленточный** вариант) со следующим составом полей: *Фамилия*, *Имя*, *Отчество*, *Оклад* (из таб. ***Сотрудники***), задайте ей имя ***Список***;
   - откройте главную форму ***Списки по подразделениям*** в режиме конструктора;
   - расположите окно БД и окно главной формы на экране так, чтобы они не перекрывали друг друга;
   - в окне БД выделите объект **Формы**, выберите подчиненную форму ***Список*** и мышью «перетащите» ее в главную форму;
   - в режиме конструктора настройте размеры подчиненной формы так, чтобы все ее поля отображались на экране;
   - перейдите в режим формы, проверьте, как работает подчиненная форма: при изменении значения **подразделения** (строка **Запись** внизу главной формы) изменяется **список** сотрудников в подчиненной форме;
   - сохраните изменения, внесенные в макет формы.
   - Разработайте подчиненную форму ***Списки по должности***, используя второй способ создания сложных форм: создайте с помощью мастера форму со

следующим составом полей: *Фамилия*, *Имя*, *Отчество*, *Оклад* (из таб. ***Сотрудники***), *Должность* (из таб. ***Должность***), <Далее>;

- определите вид представления данных - **Должность**, автоматически включается режим **Подчиненные формы**, <Далее>;



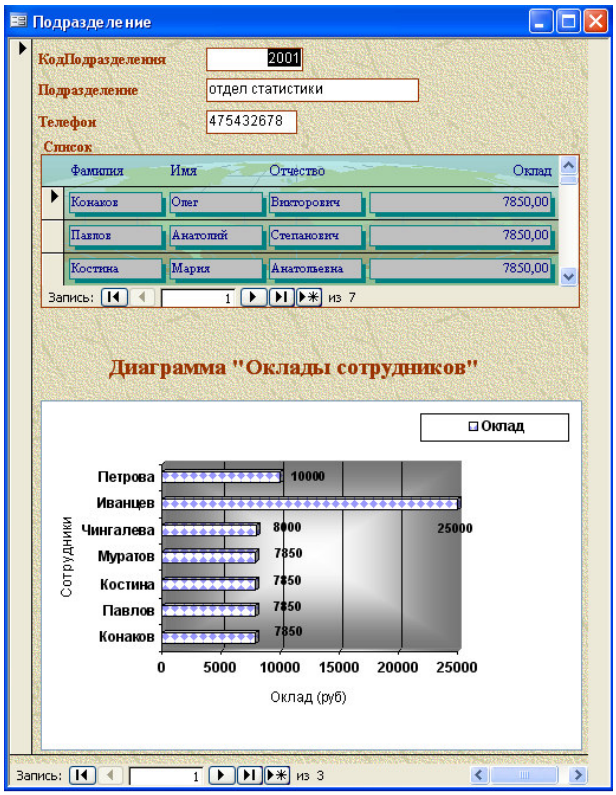
- выберите тип формы – **табличный**, <Далее>;
- определите стиль оформления формы, <Далее>;
- задайте имя главной формы ***Списки по должности***, имя подчиненной формы оставьте без изменения;
- откройте форму в просмотр кнопкой <Готово>;
- при необходимости настройте размер подчиненной формы в режиме конструктора.

2. В сложную форму ***Списки по подразделениям*** вставьте диаграмму окладов сотрудников:

- откройте форму ***Списки по подразделениям*** в режиме конструктора;
- отобразите на экране панель элементов (**Вид/Панель элементов**);
- увеличьте с помощью мыши размер окна формы и размер области данных по ширине и высоте для освобождения места под диаграмму;
- под встроенной формой ***Список*** с помощью кнопки 

вставьте текстовый элемент с названием диаграммы **Диаграмма «Оклады сотрудников»;**
- активизируйте мастер диаграмм «Microsoft Graph» с помощью соответствующей кнопки на ПИ или команды меню (**Вставка/Диаграмма**);
- на свободном месте формы под названием диаграммы с помощью мыши «растяните» прямоугольник под диаграмму от левого верхнего угла к правому нижнему, на экране откроется диалоговое окно, предлагающее выбрать источник данных для диаграммы;
- выберите таблицу ***Сотрудники***, <Далее>;
- в следующем окне выберите поля *Фамилия* и *Оклад* для отображения данных на диаграмме, <Далее>;
- в следующем окне мастера диаграмм выберите тип диаграммы – **Линейчатая**, объемный вид, <Далее>;
- в окне, где представлен образец диаграммы, щелкните двойным щелчком по надписи *Сумма_Оклад*, в открывшемся списке функций выберите значение **Отсутствует**, <Ок>, <Далее>;
- вновь щелкнуть на кнопке <Далее>, так как в строке **Поля формы** и в строке **Поля диаграммы** по умолчанию уже установлено поле *КодПодразделения* (что и требуется);
- в окне названия диаграммы удалите название **Сотрудники** (так как вы уже задали надпись для диаграммы ранее), <Готово>;
- перейдите в режим просмотра формы, оцените результат;
- отредактируйте диаграмму: вновь перейдите в режим конструктора, двойным щелчком мыши активизируйте диаграмму, настройте все элементы диаграммы (оси, легенду, ряды, стенки диаграммы, надписи) используя диалог **Параметры**

диаграммы и контекстное меню каждого элемента, добейтесь максимального сходства с рисунком.



2.13 Лабораторная работа №14 (2 часа).

Тема: «Создание отчетов для вывода данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.13.1 Цель работы: изучить и проанализировать создание отчетов для вывода данных

2.13.2 Задачи работы:

1. Изучение создания отчетов для вывода данных
2. Изучение создания отчетов режиме автоотчет в один столбец

2.13.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер

2. Microsoft Office Access

2.13.4 Описание (ход) работы:

Создание отчетов для вывода данных.

1. Для печати бланка личной карточки сотрудника (согласно данным ЛР-7):
   - создайте отчет ***Личная карточка*** в режиме **Автоотчет в один столбец** на основе запроса ***Личная карточка***;
   - откройте отчет в режиме просмотра печати, задав в параметре любую фамилию из списка сотрудников;
   - перейдите в режим конструктора отчетов;
   - в **области данных** выделите щелчком мыши элемент данных **Фамилия**;
   - с помощью кнопок на ПИ, измените размер шрифта для этого элемента - 14 пунктов, сохраните внесенные изменения;
   - вернитесь в окно просмотра и закройте отчет.

2. С помощью мастера отчетов создайте отчет ***Движение кадров*** (согласно данным ЛР-7). Поля для отчета выберите из запроса ***Запрос по движению***: *Фамилия*, *Имя*, *Отчество*, *ДатаПриема*, *Движение*, *ДатаПеревода*, *ДатаУвольнения*. Задайте группировку записей в отчете по полю *Движение*, сортировку по полю *ДатаПриема*. При необходимости откорректируйте элементы отчета в режиме конструктора.

3. С помощью мастера отчетов разработайте отчет ***Начисление зарплаты***, в который выберите поля из запроса ***Начисление зарплаты***: *ТабНомер, Фамилия, КодПодразделения, Оклад, Надбавка, Налог*. Задайте группировку записей в отчете по полю *КодПодразделения* с расчетом итоговых (**Sum**) значений *Оклада*, *Надбавки* и *Налога*. Задайте сортировку по полю *ТабНомер*. Откройте отчет для просмотра. Для добавления дополнительного поля *К_выдаче* перейдите в режим конструктора отчетов:
   - выделите строку **Область данных**;
   - с помощью кнопки  на панели элементов вставьте дополнительное поле справа от поля *Налог* в **области данных**, новое поле состоит из двух элементов **ПолеN:** (имя поля) и **Свободный** (содержимое поля);
   - элемент **ПолеN:** с помощью команд контекстного меню **Вырезать** и **Вставить** перенесите в верхнюю область и разместите его над элементом **Свободный** справа от имени поля *Налог*;
   - щелчком мыши выделите элемент данных **Свободный**, правой кнопкой мыши откройте контекстное меню, выберите команду **Свойства**;
   - в окне **свойств поля** откройте вкладку **Данные** и в строке **Данные** введите формулу для расчета суммы к выдаче: **=[Оклад]+[Надбавка]-[Налог]**;
   - откройте вкладку **Макет**, установите **Формат поля – фиксированный**, **Число десятичных знаков - 2**, закройте окно свойств поля;
   - выделите элемент **ПолеN:**, откройте для него окно **свойств надписи**;
   - на вкладке **Макет** в строке **Подпись** введите - **К_выдаче**, закройте окно свойств;
   - перейдите в режим просмотра, проверьте, верно, ли рассчитывается значение суммы *К_выдаче* для всех сотрудников, вернитесь в режим конструктора;
   - для расчета итоговых значений поля *К_выдаче* для каждой группы записей в область **Примечание группы** добавьте еще одно дополнительное поле, определив для него подпись **Общий Итог:** и расчет данных по формуле:

= Sum ([Оклад]) + Sum ([Надбавка]) – Sum ([Налог]);

   - для расчета **конечных итоговых** значений поля *К_выдаче* по всем записям отчета в область **Примечание отчета** справа добавьте аналогичное поле.
   - для элемента **ПолеN:** в окне **свойств надписи** на вкладке **Макет** в строке **Вывод на экран** установите значение **Нет** (т.к. общая подпись **ИТОГ** уже на бланке отчета есть);

- для элемента **Свободный** данного поля в **окне свойств** на вкладке **Данные** введите аналогичную формулу для расчета итоговой суммы к выдаче:

= Sum ([Оклад]) + Sum ([Надбавка]) – Sum ([Налог]);

- переходя в режим **просмотра** и обратно в режим **конструктора,** добейтесь корректного расположения элементов отчета. В конструкторе бланк отчета должен выглядеть примерно, так, как показано на рисунке.

Сохраните окончательный вариант макета отчета.

2.14 Лабораторная работа №15 (2 часа).

Тема: «Использование языка VBA при работе с основными объектами базы данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.14.1 Цель работы: изучить и проанализировать использование языка VBA при работе с основными объектами базы данных

2.14.2 Задачи работы:

1. Изучение возможностей языка Visual Basic for Applications (VBA) при создании и работе с базами данных
2. Изучение основных объектов базы данных

2.14.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.14.4 Описание (ход) работы:

Использование языка VBA при работе с основными объектами базы данных.

Ознакомление с возможностями языка Visual Basic for Applications (VBA) при создании и работе с базами данных, а также с основными объектами базы данных — таблицами, полями, формами.

Общие сведения

VBA является полнофункциональным объектно-ориентированным языком программирования, позволяющим создавать приложения пользователя в среде СУБД Access. VBA — это общее средство программирования для всего семейства Microsoft Office, включая Word, Excel, Access, Outlook, Project.

VBA целесообразно использовать для создания нестандартных процедур обработки событий и функций, выполняющих сложные вычисления, которые невозможно записать в виде выражений.

Основой программ на VBA являются процедуры, состоящие из инструкций, выполняющих необходимые операции и вычисления. Процедуры хранятся в модулях, из

которых они запрашиваются на выполнение. Собственно модуль не исполняется, а служит для объединения процедур по их функциональному назначению или привязке к форме или отчету.

Различают процедуры-подпрограммы и процедуры-функции.

Процедуры-подпрограммы (Sub) не возвращают значения вызывающей процедуре.

Процедура-подпрограмма может выполнять любые действия, например корректировать данные базы, выполнять вычисления в полях, открывать формы, печатать отчеты.

Формат процедуры-подпрограммы имеет следующий вид:

[Private!Public] [Static] Sub <Имя процедуры>[([список аргументов>)]

[<описание переменных>] [<инструкции>] [Exit Sub] [<инструкции>] End Sub

Здесь Sub, End — отмечают соответственно начало и конец тела процедуры; Public — указывает, что процедура Sub является общей, т.е. доступной для всех других процедур во всех модулях; Private — указывает, что процедура Sub доступна только для процедур того модуля, в котором она описана; Static — указывает, что значения локальных переменных процедуры сохраняются между вызовами этой процедуры; Exit Sub — приводит к немедленному завершению процедуры Sub.

Процедура-функция (Function) возвращает значение, которое присваивается ее имени внутри процедуры.

Формат процедуры-функции имеет следующий вид:

[Private!Public] [Static] Function <Имя процедуры>[(<список аргументов>)] [<описание переменных>] [<инструкции>]

[<имя процедуры>=<выражение>] [Exit Function] [<инструкции>]

[<имя процедуры>=<выражение>] End Function

В теле процедуры-функции (в отличие от процедуры-подпрограммы) присутствует инструкция присваивания имени процедуры значения, вычисляемого выражением. Эта инструкция позволяет возвратить значение из процедуры-функции в место ее вызова. При описании переменных обычно используется инструкция присвоения Dim, которая присваивает выражение переменной или константе. Инструкции присвоения всегда включают в себя знак равенства (=). Для присвоения значения переменной, описанной как объект, применяется инструкция Set.

Задание на лабораторную работу

Написать процедуру VBA, создающую новую базу данных.

1. Написать процедуру, создающую в текущей базе данных таблицу СТУДЕНТЫ с полями Номер студента, ФИО, Предмет 1, Предмет2, Предмет3, Предмет4, Средний балл.

2. Внести в созданную таблицу пять записей во все поля (кроме поля Средний балл).

3. Создать процедуру, подсчитывающую средний балл всех студентов и заносящую рассчитанные значения в поле Средний балл.

4. Создать форму, отображающую данные таблицы СТУДЕНТЫ и содержащую кнопку, запускающую процедуру расчета среднего балла.

Технология выполнения задания

1. Открыть новую базу данных и вкладку Модули, написать процедуру, создающую новую базу данных в соответствий с приведенным примером:

Создание новой базы данных Sub CreateDatabaseX()

'Описание переменных

Dim myWs As Workspace

Dim myDb As Database

'Определяем стандартный объект Workspace (рабо-'чее пространство)

Set myWs = DBEngine.Workspaces (0)

'Создаем новую базу данных 'с указанным используемым порядком символов

'dbLangGeneral

Set myDb = myWs.CreateDatabase("C:\NewDB.mdb", dbLangGeneral) myDb.Close End Sub

В результате выполнения процедуры на диске С должна появиться новая база данных с названием NewDB.mdb.

2. Написать процедуру, создающую в текущей базе данных таблицу СТУДЕНТЫ с полями Номер студента, ФИО, Предмет 1, Предмет2, Предмет3, Предмет4, Средний балл в соответствии с приведенным примером:

Создание новой таблицы СТУДЕНТЫ в текущей базе данных Sub CreateTableDefX() 'Определяем переменные Dim myDb As Database Dim myTab As TableDef Dim myF As Field Set myDb = CurrentDb()

'Создаем новый объект TableDef - таблицу СТУДЕНТЫ Set myTab = myDb.CreateTableDef("Студенты")

'Создаем новый объект Field — текстовое поле Номер 'студента и добавляем его к семейству полей объекта 'таблицы СТУДЕНТЫ

Set myF = myTab.CreateField("Номерстудента", dblnteger)

myTab.Fields.AppendmyF

1 Создаем новый объект Field — текстовое поле *ФИО* 'и добавляем его к семейству полей объекта таблицы • СТУДЕНТЫ

Set myF = myTab.CreateField(ФИО, dbText) myTab.Fields.Append myF

'Создаем новый объект Field — поле *Предмет1* и 'добавляем его к семейству полей объекта таблицы 'СТУДЕНТЫ

Set myF = myTab.CreateField("Предмет1", dblnteger) myTab.Fields.Append myF

'Аналогичнопоступаемсдругимиполямитаблицы Set myF = myTab.CreateField("Предмет2", dblnteger) myTab.Fields.Append myF

Set myF = myTab.CreateField("Предмет3", dblnteger) myTab.Fields.Append myF

Set myF = myTab.CreateField("Предмет4", dblnteger) myTab.Fields.Append myF

Set myF = myTab.CreateField("Среднийбалл", dbDouble) myTab.Fields.Append myF

'Добавляем объект таблицу СТУДЕНТЫ к семейству таблиц базы данных

myDb.TableDefs.Append myTab

End Sub

3. Открыть созданную таблицу и внести пять записей во все поля (кроме поля ***Средний балл).***

4. Создать процедуру, подсчитывающую средний балл всех студентов и заносящую рассчитанные значения в поле ***Средний балл*** в соответствии с приведенным примером:

Private Sub SB()

Dim myDb As Database 'объектная переменная типа 'базы данных

DimmyRecAsRecordset 'объектная переменная типа 'набора записей

DimsbAsDouble 'переменная для вычисления 'среднего балла

DimiAsInteger 'переменная цикла

DimmaxAsInteger 'переменная для хранения числа записей в таблице

SetmyDb = CurrentDb() 'Работаем с текущей базой данных

'Открываем набор записей таблицы СТУДЕНТЫ и присваива-'ем ссылку на него объектной переменной myRecSetmyRec = myDb.OpenRecordset("Студенты") i = 0

myRec.MoveLast 'Идем к последней записи таблицы

max = myRec.RecordCount 'При этом RecordCount содер'жит число записей в 'таблице, которое нужно нам 'для вычисления 'среднего бала в каждой строке таблицы

myRec.MoveFirst 'Переходим к первой записи таблицы и вычисляем средний ' балл

DoWhilei < max

sb = (myRec!Предмет1 + myRec!Предмет2 + myRec!Предмет3 + + myRec!Предмет4)/4

myRec.Edit 'Заносим значение среднего балла в одноименное поле

myRec![Средний балл] = sb

myRec.Update 'Для внесения данных в поля таблицы обязательно исполь-'зуются команды Edit и 'Update.

myRec .MoveNext 'Переходим к следующей записи таблицы и повторяем все

i = i + 1 'пока не достигнем последней записи.

Loop

'Закрываем набор записей. myRec.CloseEndSub

5. Создать форму, отображающую данные таблицы СТУДЕНТЫ и содержащую кнопку, запускающую процедуру расчета среднего балла.

2.15 Лабораторная работа №16 (2 часа).

Тема: «Администрирование баз данных»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.15.1 Цель работы: изучить и проанализировать администрирование баз данных

2.15.2 Задачи работы:

1. Изучение обязанностей администратора базы данных (АБД)
2. Изучение утилит АБД (Import, Export, Loader)

2.15.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.15.4 Описание (ход) работы:

Администрирование баз данных.

Обязанности администратора базы данных (АБД)

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

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

В обязанности администратора могут входить:

-инсталляция и обновление версий сервера ORACLE и прикладных инструментов

-распределение дисковой памяти и планирование будущих требований системы к памяти

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

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

-модификация структуры базы данных в соответствии с потребностями приложений

-зачисление пользователей и поддержание защиты системы

-соблюдение лицензионного соглашения ORACLE

-управление и отслеживание доступа пользователей к базе данных

-отслеживание и оптимизация производительности базы данных

-планирование резервного копирования и восстановления

-поддержание архивных данных на устройствах хранения информации

-осуществление резервного копирования и восстановления

-обращение в корпорацию Oracle за техническим сопровождением

Подключение в режиме INTERNAL

Запуск и останов базы данных - это мощные административные возможности. В угоду поддержания корректной работоспособности базы данных, функции(команды STARTUP или SHUTDOWN) остановки и запуска разрешены, только для администратора подключенного к ORACLE в режиме NTERNAL(CONNECT INTERNAL), а для возможности подключиться в режиме NTERNAL, вы должны соотвествовать одному из ниже следующих условий:

Ваше учетное имя в операционной системе имеет привилегии операционной системы, позволяющие вам соединяться в режиме INTERNAL.

Вы имеете полномочия соединяться в режиме INTERNAL.

Ваша база данных имеет пароль для INTERNAL, и вы знаете этот пароль.

Все эти требования образуют дополнительный слой защиты, препятствующий несанкционированному запуску или остановке баз данных ORACLE. Для систем, имеющих пароль для INTERNAL, существуют дополнительные, ниже описываемые соображения.

Использование пароля для INTERNAL

Некоторые операционные системы позволяют устанавливать пароль для соединений в режиме INTERNAL. Можно установить пароль для INTERNAL во время инсталляции сервера ORACLE, Oracle предоставляет утилиту для управления этим паролем (создания, изменения и удаления его).

INTERNAL и незащищенные соединения

Если используется незащищенное соединение(как большинство сетевых соединений), то ДОЛЖНО использовать пароль для INTERNAL, для последующего подключения в режиме INTERNAL; это требование подразумевает, что в системе должен быть установлен пароль для INTERNAL.

В некоторых О.С. можно либо включить, либо полностью отключить возможность соединений CONNECT INTERNAL для незащищенных соединений. Выбор делается во время инсталляции ORACLE, и может быть изменен позднее.

Утилиты АБД (Import, Export, Loader)

*SQL*Loader*

Одной из многих проблем, с которыми часто сталкиваются администраторы базы данных, является перемещение данных из внешних источников в базу данных Oracle. Сложность этой задачи возрастает с появлением хранилищ данных, приходится перемещать уже не мегабайты данных, а гигабайты, а в некоторых случаях √ терабайты. Oracle предусматривает для решения этой задачи SQL*Loader √ универсальное инструментальное средство, которое загружает внешние данные в таблицы базы данных Oracle. Утилита SQL*Loader является гибкой и настраиваемой до такой степени, что часто удается обойтись без процедур на языке третьего поколения с внедренными операторами SQL. Каждый раз, сталкиваясь с задачей преобразования инородных данных в формат Oracle, вначале рассмотрите возможность применения SQL*Loader, прежде чем обращаться к другим средствам.

Основные компоненты SQL*Loader

Для утилиты SQL*Loader необходимы входные данные 2-ух типов: внешние данные, которые могут находиться на диске или ленте, и управляющая информация (содержащаяся в управляющем файле), которая описывает характеристики входных данных. Выходные данные, часть которых является необязательной, включает таблицы Oracle, журналы, файлы некорректных записей и файлы отвергнутых записей.

Входные данные

Утилита SQL*Loader может обрабатывать файлы данных практически любого типа и поддерживает собственные типы данных почти любой платформы. Данные обычно считываются из одного или нескольких файлов данных, однако они могут быть также внесены в управляющий файл после управляющей информации. Файл данных может находиться:

В файлах с переменным форматом данные находятся в записях, которые могут изменяться по длине, в зависимости от размеров данных в полях. Поля имеют длину, необходимую для размещения данных. Поля в файлах с переменным форматом могут быть разделены завершающими символами (такими как запятые и пробелы), а так же заключены в ограничительные символы.

Управляющий файл

Прежде чем утилита SQL*Loader сможет обработать данные в файлах данных, необходимо знать орпеделения даных для SQL*Loader. Используйте управляющий файл для указания физических определений файла данных, а также формата данных в файлах. Упраляющий файл √ это файл произвольного формата, который также содержит дополнительные управляющие данные, указывающие SQL*Loader, как обрабатывать эти данные.

Журнал

После выполнения утилита SQL*Loader создает журнал, содержащий подробную информацию о загрузке, включая, следующие сведения:

Имена фалов входных данных, управляющего файла, файлов некорректных записей и файлов отвергнутых записей.

Входные данные и связанные с ними определения таблиц

-Ошибки SQL*Loader

-Результаты работы SQL*Loader

-Итоговую статистику

-Import и Export

Import и Export ≈ две дополнительные утилиты, поставляемые корпорацией Oracle. Они в основном применяются для копирования и восстановления данных и для перемещения данных либо в другую базу данных Oracle, либо из более старой версии Oracle в более новую. Ниже приведены другие возможности утилит Import и Export :

· Хранение данных в файлах операционной системы для архивирования

· Выборочное резервное копирование частей базы данных

· Перемещение данных из одной пользовательской схемы Oracle в другую

· Перемещение данных с одной аппаратной платформы или операционной системы в другую

· Экономия пространства и повышение производительности за счет уменьшения фрагментации

Работа с утилитами Import и Export весьма проста. Утилита Export записывает информацию о таблицах или объектах базы данных, такую как операторы создания таблицы, операторы создания индекса, разрешения на таблицу, информация о размерах и т.д., а также данные из самих таблиц Oracle. Затем утилита Export сохраняет эту информацию в именованных файлах операционной системы. Файлы операционной системы, создаваемые утилитой Export, известны как файлы дампа. Файлы дампа, которые представлены в двоичном формате Oracle, применяются главным образом только в утилите Import. Можно назвать Файл дампа любым именем, допустимым в операционной системе. Если вы не

укажите имя выходного файла для утилиты Export, то по умолчанию будет принято имя EXPDAT.DMP.

Затем можно сохранить выходные файлы, созданные Export, на диске или записать на съемный носитель для дальнейшего хранения, либо воспользоваться утилитой Import для воссоздания экспортируемых данных в целях восстановления или ведения базы данных.

Export

Иногда можно обнаружить, что вы не сделали в свое время то, в чем сейчас остро нуждаетесь. Возьмем, например, утилиты Import и Export. Утилита Export ≈ это самый удобный способ застраховаться от возможных неприятностей. Экспорт ≈ это универсальная утилита, поставляемая корпорацией Oracle. При всей заложенной в ней гибкости ею довольно легко пользоваться на основе обширного списка параметров. Разнообразие параметров дает возможность воспользоваться утилитой Export для решения сложных проблем управления данными. Утилита Export может записывать файлы операционной системы, которые затем можно перемещать в другую операционную систему или версию Oracle.

Проверьте, чтобы для хранения экспортного файла на запоминающем устройстве было достаточно свободного пространства. Можно использовать представление user_segments для оценки необходимого дискового пространства.

В следующем коде показан пример использования утилиты Export:

exp userid=system/manager OWNER=scott... [прочие опции}

Использование файла параметров

Можно использовать файл параметров как для утилиты Export, так и для утилиты Import. Файл параметров помогает выполнять операции импорта и экспорта, обеспечивая при этом непротиворечивость и простоту. Его удобно использовать для экспорта в ночное время. Файлы параметров гарантируют целостный экспорт для полной уверенности в том, что все необходимые таблицы действительно будут экспортированы. Можно вызывать эти утилиты, задавая параметры в командной строки или в сценарии операционной системы, но использование командной строки может не позволить задать все необходимые параметры. Сценарий export_ts записывает необходимый ему файл параметров. Имена таблиц в кавычках являются чувствительными к регистру.

Режим экспорта таблиц

Режим экспорта таблиц используется для экспорта одной таблицы или перечня таблиц, а не всей базы данных. По умолчанию он экспортирует все таблицы, которые принадлежат пользователю, выполняющему экспорт. Пользователи, имеющие доступ к другой схеме, могут экспортировать таблицы из этой схемы, указав имя схемы.

Режим экспорта пользователя

Режим экспорта пользователя в основном используется для экспорта всех таблиц и индексов конкретного пользователя или перечня пользователей. Этот режим работает хорошо при создании пользователя, который является владельцем всех объектов приложения. Например, если существует пользователь с именем sales, который является владельцем всех таблиц и индексов и других объектов в приложении sales, экспорт приложения может выглядеть следующим образом:

exp VSERlD=system/manager OWNER=sales

Режим экспорта всей базы данных

Режим экспорта всей базы данных используется для экспорта всех объектов базы данных, за исключением объектов, которые обычно создаются и поддерживаются учетной записью SYS. Эту опцию могут применять только пользователи, которым назначена роль EXP_FULL_DATABASE. Здесь можно упомянуть несколько других интересных возможностей. По умолчанию Oracle выполняет полный экспорт при указании режима экспорта всей базы данных (INCTYPE= COMPLETE). Если указана опция INCTYPE= INCREMENTAL, Oracle будет экспортировать только таблицы, содержащие какие-либо изменившиеся строки, начиная с последнего полного экспорта любого типа. Если указана

опция INCTYPE=CUMULATIVE, Oracle будет экспортировать только таблицы, содержащие какие-либо измененные строки, начиная с последнего полного или кумулятивного экспорта.

Типы экспорта:

-Полный экспорт

-Инкрементный экспорт

-Кумулятивный экспорт

Import

Утилита Import противоположна утилите Export. Она отвечает за чтение экспортных файлов в целях воссоздания объектов базы данных, а также любого состояния, в котором они экспортировались первоначально. Утилита Import может также преобразовывать данные предоставленные с разных платформ к примеру с UNIX машины в ASCII кодах, на мейнфрем с кодировкой EBCDIC и наоборот, что позволяет перемещать данные с одной платформы на другую. Утилита Import может работать в интерактивном режиме или в режиме командной строки. При использовании интерактивного режима утилита Import запрашивает у пользователя параметры, необходимые для выполнения импорта. Обычно проще задать параметры в командной строке или в файле параметров. Утилита Import, как и Export, использует файлы параметров.

Пользователи базы данных и схемы

Каждая база данных ORACLE имеет список имен пользователей. Чтобы получить доступ к базе данных, пользователь должен использовать приложение базы данных, и попытаться соединиться с базой данных, предоставив действительное имя пользователя. С каждым именем пользователя связан пароль, чтобы предотвратить несанкционированный доступ.

С каждым именем пользователя ассоциирована СХЕМА(SCHEMA), имеющая такое же имя. SCHEMA(схема) - это логическая коллекция объектов базы данных (таблиц, представлений, последовательностей, синонимов, индексов, кластеров, процедур, функций, пакетов и связей баз данных). По умолчанию, каждый пользователь базы данных создает и имеет доступ ко всем объектам в соответствующей схеме. Команда CREATE SCHEMA полезна для гарантирования успешного создания всех нужных объектов и грантов за одну операцию; если индивидуальный объект внутри этой операции не может быть создан, все предложение откатывается и аннулируются все его результаты.

Домен защиты

Каждый пользователь имеет ДОМЕН ЗАЩИТЫ - набор свойств, которые определяют такие вещи, как:

· действия (привилегии и роли), доступные пользователю

· квоты табличных пространств (доступной дисковой памяти)

· лимиты на системные ресурсы (например, время процессора) для данного пользователя

Каждое свойство, вносящее вклад в домен защиты пользователя, обсуждается ниже.

Привилегии

ПРИВИЛЕГИЯ(PRIVILEGE) - это право выполнять определенный тип предложений SQL. Некоторые примеры привилегий включают:

· право соединяться с базой данных (создавать сессию)

· право создавать таблицу в вашей схеме

· право выбирать строки из чьей-либо таблицы

· право выполнять чью-либо хранимую процедуру

Привилегии в s базе данных ORACLE могут быть разделены на две различные категории: системные привилегии и объектные привилегии.

Системные привилегии

СИСТЕМНЫЕ ПРИВИЛЕГИИ(SYSTEM_PRIVILEGE) позволяют пользователям выполнять конкретное действие на уровне системы, или конкретное действие над конкретным типом объектов. Таковы, например, привилегия создавать табличное

пространство или привилегия удалять строки любой таблицы в базе данных. Большинство системных привилегий доступны только администраторам и разработчикам приложений, поскольку такие привилегии весьма мощны.

Объектные привилегии

ОБЪЕКТНЫЕ ПРИВИЛЕГИИ позволяют пользователям выполнять конкретные действия на конкретном объекте. Такова, например, привилегия удалять строки в указанной таблице. Объектные привилегии назначаются пользователям, так что они могут использовать приложения базы данных для выполнения конкретных задач.

Назначение привилегий

Привилегии назначаются пользователям с тем, чтобы эти пользователи могли обращаться к данным и модифицировать эти данные в базе данных. Пользователь может получить привилегию двумя различными способами:

Привилегии могут назначаться пользователям явно. Например, привилегия вставлять записи в таблицу EMP может быть явно назначена пользователю SCOTT.

Привилегии могут назначаться РОЛЯМ (именованным группам привилегий), а затем эта роль может быть назначена одному или нескольким пользователям. Например, привилегия вставлять записи в таблицу EMP может быть назначена роли с именем CLERK, а эта роль, в свою очередь, может быть назначена пользователям SCOTT и BRIAN.

Роли (ROLE)

ORACLE предоставляет легкое и контролируемое управление привилегиями через использование ролей. РОЛИ (ROLE) - это поименованные группы взаимосвязанных привилегий, которые назначаются пользователям или другим ролям.

Табличные пространства и файлы данных

Используемые данные базы данных ORACLE логически хранятся в ТАБЛИЧНЫХ ПРОСТРАНСТВАХ (TABLESPACE), а физически располагаются в ФАЙЛАХ ДАННЫХ, ассоциированных с соответствующим табличным пространством. Эту взаимосвязь иллюстрирует следующий рисунок:

Файлы данных и табличные пространства

Файлы данных - физические структуры, каждая из которых связана с одним табличным пространством

Объекты - Хранятся в табличных пространствах, и могут располагаться в нескольких файлах данных

Табличные пространства

Каждая база данных ORACLE подразделяется на логические элементы, называемые ТАБЛИЧНЫМИ ПРОСТРАНСТВАМИ(TABLESPACE). Администратор базы данных может использовать табличные пространства для:

-управления распределением памяти для объектов базы данных

-установления квот памяти для пользователей базы данных

-управления доступностью данных, путем перевода отдельных табличных пространств в состояния online или offline

-копирования и восстановления данных

-распределения данных по устройствам для повышения производительности

Табличное пространство SYSTEM

Каждая база данных ORACLE содержит табличное пространство SYSTEM, которое создается автоматически при создании базы данных. Табличное пространство SYSTEM всегда содержит таблицы словаря данных для всей базы данных. Небольшой базе данных может оказаться достаточным одного табличного пространства SYSTEM; однако рекомендуется создать по меньшей мере одно дополнительное пространство, чтобы хранить данные пользователей отдельно от информации словаря данных. Это позволит более гибко осуществлять разнообразные операции администрирования.

Размер табличного пространства - это размер его файла данных или суммарный размер всех файлов данных, составляющих это табличное пространство.

Онлайновые и офлайновые табличные пространства

АБД может перевести любое табличное пространство в базе данных ORACLE в состояние ОНЛАЙН (т.е. доступно) или ОФЛАЙН (недоступно), если база данных открыта. Единственным исключением является то, что табличное пространство SYSTEM всегда находится в онлайне, ибо словарь данных должен быть всегда доступен ORACLE. Обычное состояние табличного пространства - онлайн, так что данные, содержащиеся в нем, доступны пользователям базы данных. Однако администратору может понадобиться перевод табличного пространства в офлайн по одной из следующих причин:

-чтобы сделать часть базы данных недоступной, сохраняя в то же время нормальный доступ к остальной части

-чтобы выполнить резервное копирование офлайнового табличного пространства (хотя такое копирование можно осуществлять и в онлайне, одновременно с использованием табличного пространства)

-чтобы сделать приложение вместе с его группой таблиц временно недоступным на время обновления или сопровождения этого приложения

Когда табличное пространство переводится в офлайн, ORACLE не позволяет последующим предложениям SQL обращаться к объектам этого табличного пространства.

Табличное пространство не может быть переведено в офлайн, если оно содержит активные сегменты отката. Табличное пространство может быть переведено в офлайн лишь в том случае, если все сегменты отката, содержащиеся в нем, не используются.

Файлы данных

Табличное пространство в базе данных ORACLE состоит из одного или нескольких физических ФАЙЛОВ ДАННЫХ. Файлы данных, ассоциированные с табличным пространством, хранят все данные этого табличного пространства. Любой файл данных может ассоциироваться только с одним табличным пространством и только с одной базой данных. При создании файла данных для табличного пространства ORACLE распределяет ему указанное количество дисковой памяти. Когда файл данных создается, операционная система несет ответственность за очистку старой информации и за установку должных режимов доступа к файлу, прежде чем он будет распределен ORACLE. Если файл велик, этот процесс может потребовать значительного времени. Поскольку первым табличным пространством в любой базе данных всегда является SYSTEM, первые файлы данных любой базы данных автоматически распределяются табличному пространству SYSTEM во время создания базы данных.

Содержимое файла данных

После первоначального создания файла данных соответствующее дисковое пространство еще не содержит никаких данных; однако это пространство ЗАРЕЗЕРВИРОВАНО за будущими сегментами ассоциированного табличного пространства - оно не может содержать каких-либо иных данных. Когда сегмент (например, сегмент данных таблицы) будет создан и начнет увеличиваться в размерах, ORACLE использует свободное место в соответствующих файлах данных, чтобы распределять экстенты для этого сегмента. Данные в сегментах объектов (сегментах данных, сегментах индексов, сегментах отката и т.д.) в табличном пространстве физически хранятся в одном или нескольких файлах данных, составляющих это табличное пространство.

Офлайновые файлы данных

Табличные пространства в любой момент можно переводить в ОФЛАЙН (т.е. делать недоступными) или в ОНЛАЙН (делать доступными). Поэтому все файлы данных, составляющие табличное пространство, переводятся в офлайн или онлайн одновременно, всей группой. Индивидуальные файлы данных также могут быть переведены в офлайн; однако это обычно делается лишь при некоторых процедурах восстановления базы данных.

Схемы и объекты схемы

СХЕМА(SCHEMA) - это коллекция объектов. ОБЪЕКТЫ СХЕМЫ - это логические структуры, непосредственно относящиеся к данным базы данных. Объекты схемы включают

такие структуры как: таблицы, представления, последовательности, хранимые процедуры, синонимы, индексы, кластеры и связи баз данных. (Не существует взаимосвязи между табличным пространством и схемой; объекты одной и той же схемы могут находиться в разных табличных пространствах, и одно и то же табличное пространство может содержать объекты из разных схем.)

ТАБЛИЦА(TABLE) - это основная единица хранения данных в базе данных ORACLE. Таблицы базы данных хранят все данные, доступные пользователям.

Данные таблицы хранятся в виде СТРОК и СТОЛБЦОВ. Каждая таблица определяется с ИМЕНЕМ ТАБЛИЦЫ и набором столбцов. Каждому столбцу дается ИМЯ СТОЛБЦА, ТИП ДАННЫХ (такой как CHAR, DATE или NUMBER), а также ШИРИНА (которая может быть предопределена типом данных, как в случае DATE) или МАСШТАБ и ТОЧНОСТЬ (только для типа данных NUMBER). После того, как таблица создана, в нее могут быть вставлены строки действительных данных. После этого строки таблицы можно опрашивать, удалять или обновлять. Чтобы учредить организационные правила для данных таблицы, для таблицы можно также определить ограничения целостности и триггеры.

ПРЕДСТАВЛЕНИЯ(VIEW) - это настроенное представление данных из одной или нескольких таблиц. представлений можно рассматривать как "хранимый запрос". Представления в действительности не содержат данных; вместо этого они доставляют данные из тех таблиц, на которых они основаны (так называемых БАЗОВЫХ ТАБЛИЦ представлений). Базовые таблицы, в свою очередь, могут быть как таблицами, так и представлениями.

Как и таблицы, представления можно опрашивать, обновлять и осуществлять в них вставки и удаления, с некоторыми ограничениями. Все операции, осуществляемые над представлениями, в действительности затрагивают базовые таблицы этого представления.

ПОСЛЕДОВАТЕЛЬНОСТЬ(SEQUENCE) - генерирует уникальные порядковые номера, которые могут использоваться как значения числовых столбцов таблиц базы данных. Последовательности упрощают прикладное программирование, автоматически генерируя уникальные числовые значения для строк одной или нескольких таблиц. Номера, генерируемые последовательностью, независимы от таблиц, так что одну и ту же последовательность можно использовать для нескольких таблиц. После ее создания, к последовательности могут обращаться различные пользователи, чтобы получать действительные порядковые номера.

Термином "программная единица" обозначаются хранимые процедуры, функции и пакеты.

ПРОЦЕДУРА(PROCEDURE) или ФУНКЦИЯ(FUNCTION) - это совокупность предложений SQL и PL/SQL, сгруппированных вместе как выполнимая единица, исполняющая специфическую задачу.

Процедуры и функции сочетают легкость и гибкость SQL с процедурными возможностями языка структурного программирования. С помощью PL/SQL такие процедуры и функции можно определять и сохранять в базе данных для продолжительного использования.

Процедуры и функции похожи друг на друга, с той разницей, что функция всегда возвращает вызывающей программе единственное значение, тогда как процедура в общем случае не возвращает значения, однако существуют специфические методики для возвращения значения процедуры.

ПАКЕТЫ(PACKAGE) дают метод инкапсулирования и хранения взаимосвязанных процедур, функций и других конструктов пакета как единицы в базе данных. Предоставляя администратору базы данных или разработчику приложений организационные преимущества, пакеты в то же время расширяют функциональные возможности, и увеличивают производительность базы данных.

СИНОНИМ(SYNONYM) - это алиас (дополнительное имя) для таблицы, представлений, последовательности или программной единицы. Синоним не есть объект, но он является прямой ссылкой на объект. Синонимы используются для:

-маскировки действительного имени и владельца объекта;

-обеспечения общего доступа к объекту

-достижения прозрачности местоположения для таблиц, представлений или программных единиц удаленной базы данных

-упрощения кодирования предложений SQL для пользователей базы данных

Синоним может быть общим или личным. Индивидуальный пользователь может создать ЛИЧНЫЙ СИНОНИМ, который доступен только этому пользователю. Администраторы баз данных чаще всего создают ОБЩИЕ СИНОНИМЫ, благодаря которым объекты базовых схем становятся доступными для общего пользования всем пользователям базы данных.

Индексы, кластеры и хешированные кластеры - это необязательные структуры, ассоциированные с таблицами, которые можно создавать для повышения производительности операций извлечения данных. Так же, как индекс в книге позволяет вам быстрее отыскивать нужную информацию, индекс ORACLE предоставляет быстрый путь доступа к данным таблицы. При обработке запроса ORACLE может использовать некоторые или все имеющиеся индексы для эффективного отыскания запрашиваемых строк. Индексы полезны, когда приложения часто опрашивают интервалы строк таблицы либо отдельные строки.

Индексы создаются по одному или нескольким столбцам таблицы. Однажды созданный, индекс автоматически поддерживается и используется ORACLE. Изменения в данных таблицы автоматически отражаются во всех соответствующих индексах при полной прозрачности для пользователей.

КЛАСТЕРЫ(CLUSTER) предоставляют необязательный способ хранения данных таблиц. Кластер - это группа из одной или нескольких таблиц, физически хранящихся вместе.

Взаимосвязанные столбцы таблиц в кластере называются КЛЮЧОМ КЛАСТЕРА. Ключ кластера индексируется, так что строки кластера могут извлекаться с минимальными затратами на ввод-вывод. Поскольку данные ключа кластера в индексированном (не кэшированном) кластере хранятся в одном экземпляре для всех таблиц кластера, достигается экономия пространства по сравнению с обычными (некласте- ризованными) таблицами.

Кластеризованные таблицы: Связанные данные хранятся вместе, более эффективно Некластеризованные таблицы: Связанные данные хранятся отдельно, занимая больше места.

Кластеры могут также повысить эффективность извлечения данных, в зависимости от распределения данных и от того, какие операции SQL наиболее часто выполняются на кластеризованных данных. В частности, кластеризованные таблицы, опрашиваемые через соединения, выигрывают за счет кластеров, потому что строки, общие для объединяемых таблиц, извлекаются за одну операцию ввода-вывода. Как и индексы, кластеры не влияют на проектирование приложений. Является ли таблица частью кластера или нет, остается прозрачным для пользователей и приложений. Данные, хранящиеся в кластеризованной таблице, доступны через те же операции SQL, как если бы они не были кластеризованы.

ХЭШИРОВАННЫЕ КЛАСТЕРЫ похожи на обычные, индексированные, кластеры. Однако в хэшированных кластерах строки записываются не на основе ключа кластера, а на основе значения ФУНКЦИИ ХЭШИРОВАНИЯ, применяемой к ключу кластера. Все строки с одинаковым значением такого хэш-ключа хранятся на диске вместе. Хэшированные кластеры выигрывают по сравнению с индексированной таблицей и индексированным кластером, когда таблица часто опрашивается на равенство. Для таких запросов значения указанного ключа кластера хэшируются, и результирующие значения хэш-ключа прямо указывают на участок диска, в котором хранятся соответствующие строки.

СВЯЗЬ БАЗ ДАННЫХ - это именованный объект, который описывает "путь" от одной базы данных к другой. Связи баз данных неявно используются при обращении к ГЛОБАЛЬНОМУ ИМЕНИ ОБЪЕКТА в распределенной базе данных.

2.16 Лабораторная работа №17, интерактивная форма (2 часа).

Тема: «Обзорное итоговое занятие»

На занятии происходит привитие навыков решения задач с применением специальных методов и алгоритмов на основе специально подобранных примеров нарастающей сложности, выстроенных по типу **практикума** (особый вид проведения учебных занятий, покрывающий определенную тему образовательной области, имеющий целью практическое усвоение основных положений предмета), а также применяется метод ПОПС-формулы. На занятии студенты высказывают свою точку зрения, отношение к предложенной проблеме. ПОПС-формула применяется для закрепления изученного материала на лекционном занятии. Каждый студент выбирает одну из предложенных задач или создает свою. В случае необходимости студент может усложнить выбранную задачу, повторно решив ее, используя компьютерную симуляцию. После решения задачи студент защищает решение по методу ПОПС-формулы, высказывая: П-позицию (объясняет, в чем заключена точка зрения студента); О-обоснование (не просто объясняет свою позицию, но и доказывает); П-пример (при разъяснении сути позиции пользуется конкретными примерами); С-следствие (делает вывод в результате обсуждения определенной проблемы).

Оценка результатов практического занятия происходит согласно, предусмотренных баллов в рамках каждого пункта ПОПС-формулы.

2.16.1 Цель работы: повторение всех вопросов, которые решались на предыдущих занятиях

2.16.2 Задачи работы:

1. Повторить этапы создания базы данных
2. Повторить этапы и способы программирования баз данных
3. Повторить администрирование баз данных

2.16.3 Перечень приборов, материалов, используемых в лабораторной работе:

1. Персональный компьютер
2. Microsoft Office Access

2.16.4 Описание (ход) работы:

Студенты на обзорном итоговом занятии повторяют пройденный материал дисциплины, выполняют пропущенные лабораторные работы и иные виды деятельности для увеличения количества полученных за семестр обучения баллов рейтингового контроля.

Разработал(и): _____ Д.А. Андриенко