ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Вычислительные системы и информационная безопасность»

СБОРНИК УПРАЖНЕНИЙ

по дисциплине

«Базы знаний и базы данных»

Автор
Чуйкова Е.Н.

Ростов-на-Дону, 2014

Аннотация

Сборник упражнений предназначен для студентов направления 090900 очной формы обучения.

Автор

к.т.н., доцент Чуйкова Е.Н.

Оглавление

ЛАБОРАТОРНАЯ РАБОТА НА ТЕМУ: ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ МОДЕЛИ ДАННЫХ (МОДЕЛИ «СУЩНОСТЬ-СВЯЗЬ»)

Цель работы: освоить приемы создания инфологической модели данных.

Руководство к выполнению

Концептуальное проектирование базы данных состоит в построении инфологической модели данных (модели **«сущность - связь»** (или ER-модели)).

Основные понятия модели «сущность - связь» включают **сущности, связи** и **атрибуты.**

Сущности – объекты, информацию о которых нужно хранить в базе данных. Сущность представляет множество объектов реального мира с одинаковыми свойствами. Сущность может представлять физические или абстрактные объекты.

Каждый уникально идентифицируемый экземпляр сущности соответствует конкретному объекту данного вида. Например, отдельные сотрудники являются экземплярами сущности EMPLOYEE (сотрудник).

Каждая сущность идентифицируется именем и набором свойств. База данных обычно содержит много разных сущностей.

Отдельные свойства сущностей называются **атрибутами**. Разные сущности могут быть **связаны** друг с другом или находиться между собой в некоторых отношениях.

Примеры введенных понятий приведены ниже.

Понятие	Неформальное определение	Примеры
СУЩНОСТЬ	Различимые объекты	Поставщик, товар, поставка, сотрудник, отдел, человек, произведение, концерт, оркестр, заказ.
АТРИБУТ	Информация, описывающая некоторое свойство объекта	Номер поставщика, объем поставок, название отдела, ФИО сотрудника, дата заказа.

СВЯЗЬ	Объект, который служит для организации взаимодействия двух или нескольких других объектов	Поставка (поставщик-товар), назначение (сотрудник-отдел)

Модель данных «сущность - связь» представляется в виде диаграмм. Пример диаграммы приведен на рис. 1.

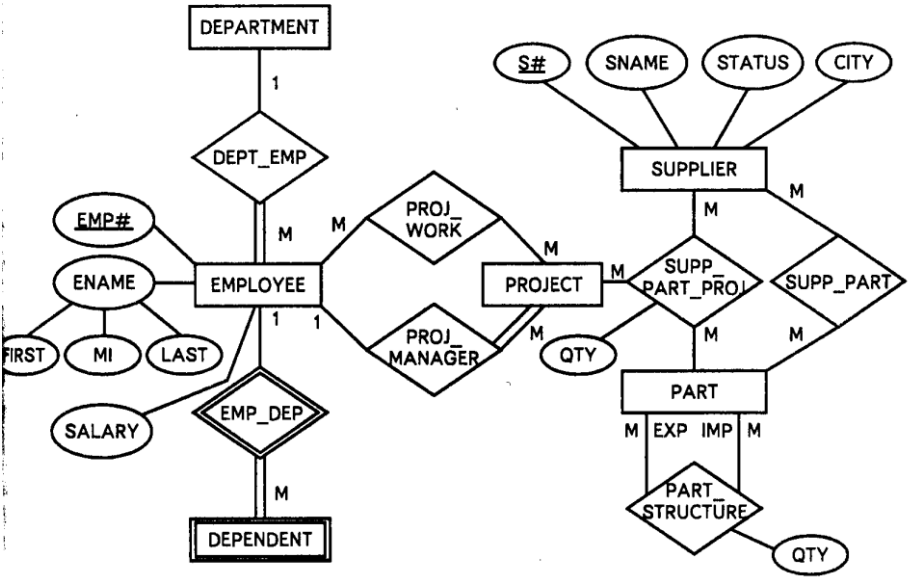


Рис. 1. Диаграмма модели «сущность - связь».

Сущности

Сущности можно классифицировать как сильные и слабые.

Слабая сущность – сущность, существование которой зависит от какой-то другой сущности.

Сильная сущность – сущность, существование которой *не* зависит от какой-то другой сущности.

Сущности показаны на диаграмме в виде прямоугольников с именами сущностей внутри. *Слабые* сущности изображаются в *двойной* рамке.

Примеры (см. рис. 1).

- Сильные сущности:

DEPARTMENT (отдел)

EMPLOYEE (сотрудник)
SUPPLIER (поставщик)
PART (деталь)
PROJECT (проект)

- Слабые сущности:

DEPENDENT (подчиненный)

Слабые сущности иногда называют **дочерними, зависимыми** или **подчиненными,** а сильные – **родительскими, сущностями-владельцами.** Слабая сущность *не может* существовать, если не существует соответствующая сильная сущность. Например, сущность DEPENDENT (подчиненный сотрудник) является слабой, поскольку она не может существовать (в контексте данной базы данных), если не существует соответствующая сущность EMPLOYEE (сотрудник). В частности, если данный сотрудник (EMPLOYEE) будет удален, то все зависящие от него экземпляры сущности (DEPENDENT) также будут удалены.

Атрибуты

Атрибут – свойство сущности или связи. Все экземпляры сущности обладают некоторыми общими атрибутами: например, все сотрудники имеют табельный номер, имя, зарплату и т.д. Значения каждого атрибута извлекаются из соответствующего множества значений, называемого доменом.

Связь, которая соединяет две сущности, также может иметь атрибуты, аналогичные атрибутам сущности.

Атрибуты делятся на простые и составные, однозначные и многозначные, а также производные.

Простой атрибут – атрибут, состоящий из одного компонента. Примерами простых атрибутов являются табельный номер сотрудника (EMP#) или зарплата (SALARY).

Составной атрибут – атрибут, состоящий из нескольких компонентов (например, составной атрибут "имя сотрудника" может складываться из простых атрибутов "имя", "отчество" и "фамилия").

Однозначный атрибут – атрибут, который содержит одно значение для одного экземпляра сущности. Например, экземпляр сущности EMPLOYEE (сотрудник) имеет единственное значение атрибута "Номер сотрудника"-- (EMP#).

Многозначный атрибут – атрибут, который содержит несколько значений для одного экземпляра сущности. Например, экземпляр сущности "отдел" может содержать несколько значений для атрибута номера телефона отдела.

Производный атрибут – атрибут, который представляет значение, производное от значения связанного с ним атрибута или некоторого множества атрибутов, принадлежащих некоторой сущности (например, общее количество определенного товара может быть выведено на основе суммирования отдельных объемов поставок данного товара).

Ключи

Под ключом подразумевается элемент данных, который позволяет уникально идентифицировать отдельные экземпляры некоторой сущности.

Потенциальный ключ – атрибут или набор атрибутов, который уникально идентифицирует отдельные экземпляры сущности. Например, номер отдела является потенциальным ключом сущности "отдел". Потенциальный ключ должен содержать значения, которые уникальны для каждого отдельного экземпляра сущности. Например, каждый отдел организации обладает уникальным номером, и не существует отделов с одинаковыми номерами.

Первичный ключ – потенциальный ключ, который выбран в качестве первичного ключа. Сущность может иметь несколько потенциальных ключей. Например, каждый сотрудник может иметь уникальный номер социального страхования, а также уникальный личный (табельный) номер, присваиваемый сотрудникам этой организации. Таким образом, сущность "сотрудник" обладает двумя потенциальными ключами, каждый из которых может быть выбран в качестве первичного ключа.

Выбор первичного ключа сущности осуществляется из соображений суммарной длины атрибутов, минимального количества атрибутов в ключе, а также наличия гарантии уникальности его значений в текущий момент времени и в обозримом будущем. В частности, личный номер сотрудника меньше по размеру, а потому предпочтительнее, чем номер социального страхования. Следовательно, первичным ключом сущности "сотрудник" целесообразно выбрать именно личный номер сотрудника, номер социального страхования будет называться в этом случае **альтернативным ключом**.

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

Представление атрибутов на диаграмме

На диаграмме атрибуты изображаются в виде эллипсов, соединенных сплошной линией с соответствующей сущностью (или

связью), и помеченных именем атрибута. Эллипс обведен штриховой линией, если атрибут производный, и двойной линией, если атрибут многозначный. Если атрибут составной, то составляющие его атрибуты показаны в виде других эллипсов, соединенных с эллипсом составного атрибута с помощью дополнительных линий. Ключевые атрибуты подчеркиваются, а множества значений не показываются вовсе (см. рис.1).

Примеры атрибутов (см. рис. 1).

- Для EMPLOYEE:
  EMP# (ключевой атрибут)
  ENAME (составной, состоящий из атрибутов FIRST, MI и LAST)
  SALARY (простой атрибут)
- Для SUPPLIER:
  S# (ключевой атрибут)
  SNAME
  STATUS
  CITY
    - Для SUPP_PART_PROJ:

  QTY (количество)
- Для PART_STRUCTURE:
  QTY

Для экономии места другие свойства на рис. 1 не показаны.

Связи

Связь - осмысленная ассоциация между разными сущностями.

Связь является набором ассоциаций между двумя (или больше) сущностями-участниками. Каждой связи присваивается имя, которое должно описывать ее функцию. Например, между отделами и сотрудниками определена связь (DEPT_EMP), которая означает, что в данном отделе работает заданное множество сотрудников.

Как и в случае сущностей, здесь следует различать понятия "связь" и "экземпляр связи".

Связь – ассоциация между сущностями, включающая по одной сущности из каждой участвующей в связи сущности.

Каждый уникально идентифицируемый экземпляр связи указывает на отдельные экземпляры сущностей, объединяемые им.

Связь PROJ_WORK (занят в проекте) указывает на взаимосвязь между сущностями EMPLOYEE и PROJECT, каждый экземпляр связи PROJ_WORK связывает один экземпляр сущности EMPLOYEE

с одним экземпляром сущности PROJECT.

Представление связей на диаграммах

Каждая связь изображается в виде ромба с указанным на нем именем связи (см. рис.1). Ромб окружен двойной линией, если связь соединяет слабую сущность с сильной сущностью, от которой эта слабая сущность зависит.

Охваченные некоторой связью сущности называются **участниками** этой связи. Участники каждой связи присоединены к соответствующему ромбу сплошными линиями. Количество участников некоторой связи называется **степенью** этой связи. Следовательно, степень связи указывает на количество сущностей, охваченных данной связью. Связь со степенью два называется **бинарной.** Связь со степенью три называется **тернарной.**

Связь, в которой одни и те же сущности участвуют несколько раз и в разных ролях, называется **рекурсивной связью.**

Связям могут приписываться атрибуты.

Существует два варианта участия сущности в связи: **полное** и **частичное.**

Пусть *R* является связью, которая содержит сущность *E* в качестве участника. Если каждый экземпляр сущности *E* находится по крайней мере в одном экземпляре связи *R,* то участие *E* в связи *R* называется полным, в противном случае - частичным. Например, если каждый товар должен поставляться по крайней мере одним поставщиком, то участие товаров в отношении между поставщиками и товарами (SUPP_PART) является полным. Но если допустима такая ситуация, когда некоторый товар не поставляется ни одним из поставщиков, то участие товаров в отношении SUPP_PART является частичным.

Каждая связь характеризуется показателем кардинальности.

Показатель кардинальности описывает количество возможных связей для каждой из сущностей-участниц.

Наиболее распространенными являются бинарные связи с показателями кардинальности "один к одному" (1:1), "один ко многим" (1:M), "многие ко многим" (M:N).

Участники каждой связи на ER-диаграмме соединяются линиями с метками 1, M или N, определяющими показатель кардинальности этой связи. Двойная линия обозначает полное участие.

Связь **"один к одному"** означает, что один экземпляр сущности A связан связью C с единственным экземпляром сущности B. Например, один экземпляр сущности "менеджер" связан

связью "управляет" с одним экземпляром сущности "отделение компании".

Связь **"один ко многим"** означает, что один экземпляр сущности A связан с несколькими экземплярами сущности B связью C. Например, один экземпляр сущности DEPARTMENT (отдел) связан связью DEPT_EMP с несколькими экземплярами сущности EMPLOYEE (сотрудник). Поэтому, с точки зрения сущности DEPARTMENT связь DEPT_EMP является связью типа 1:M. Если рассмотреть связь с противоположной стороны, то можно заметить, что один экземпляр сущности EMPLOYEE связан связью DEPT_EM с единственным экземпляром сущности DEPARTMENT (каждый сотрудник работает в одном отделе). Следовательно, с точки зрения сущности EMPLOYEE связь DEPT_EM является связью типа 1:1. Однако на ER-диаграммах ее следует представлять с наиболее высоким из всех существующих показателем кардинальности, т.е. для связи DEPT_EM показатель кардинальности мы принимаем равным 1:M.

Связь **"многие ко многим"** рассмотрим на примере связи PROJ_WORK (занят в проекте) между сущностями EMPLOYEE и PROJECT. Один экземпляр сущности EMPLOYEE (сотрудник) может быть связан с одним или больше экземплярами сущности PROJECT (т.е. отдельный сотрудник может быть занят в одном или нескольких проектах). Следовательно, связь PROJ_WORK, с точки зрения сущности EMPLOYEE, является связью типа "один ко многим" (1:M).

Если рассмотреть связь с противоположной стороны, то можно заметить, что в каждом проекте занято несколько сотрудников. Отсюда следует, что один экземпляр сущности PROJECT связан с несколькими экземплярами сущности EMPLOYEE. Поэтому связь PROJ_WORK, с точки зрения сущности PROJECT, также является связью "один ко многим" (1:M).

Итак, связь PROJ_WORK является связью типа 1:M и с точки зрения сущности EMPLOYEE, и с точки зрения сущности PROJECT. Таким образом, связь PROJ_WORK представлена в виде двух связей типа "один ко многим" (1:M), которые вместе образуют связь "многие ко многим" (M:N).

Примеры (см. рис. 1).

- DEPT_EMP (связь "один ко многим" между сущностями DEPARTMENT и EMPLOYEE)
- EMP_DEP (связь "один ко многим" между сущностью EMPLOYEE и сущностью слабого типа DEPENDENT)
- PROJ_WORK и PROJ_MANAGER (обе связи между сущ-

ностями EMPLOYEE и PROJECT, первая имеет тип "многие ко многим", а вторая - "один ко многим")

• SUPP_PART_PROJ (тернарная связь многие-ко-многим-ко-многим, включающая сущности SUPPLIER, PART и PROJECT)

• SUPP_PART (связь "многие ко многим" между сущностями SUPPLIER и PART)

• PART_STRUCTURE (связь "многие ко многим" между сущностями PART и PART)

Обратите внимание, что в последнем случае две линии от PART к PART_STRUCTURE отличаются своими надписями с указанием различных выполняемых "ролей" (EXP и IMP, которые обозначают соответственно "раскладывание товара по компонентам" и "складывание товара по компонентам"). Связь PART_STRUCTURE является типичным примером рекурсивной связи.

Суперклассы и подклассы сущностей

Суперкласс – сущность, включающая разные подклассы, которые необходимо представить в модели данных.

Подкласс является сущностью, которая исполняет отдельную роль, а также является членом суперкласса.

В некоторых случаях сущность может иметь несколько разных подклассов. Например, для сущности "сотрудник" отдельные экземпляры этой сущности можно классифицировать как подтипы "менеджер", "секретарь" и "торговый агент". Иначе говоря, сущность "сотрудник" можно рассматривать как суперкласс для подклассов "менеджер", "секретарь" и "торговый агент". Связь между суперклассом и любым его подклассом называется связью "суперкласс/подкласс".

Каждый член подкласса является членом суперкласса. Другими словами, член подкласса является сущностью суперкласса и в то же время играет собственную отдельную роль. Связь между суперклассом и подклассом относится к типу "один ко многим" (1:M). Некоторые суперклассы могут содержать перекрывающиеся подклассы. Например, сотрудник может быть одновременно менеджером и торговым агентом. В этом примере подклассы "менеджер" и "торговый агент" являются перекрывающимися подклассами суперкласса "сотрудник". Однако не каждый член суперкласса обязательно должен быть членом какого-либо подкласса — например, это могут быть рядовые сотрудники, не играющие какой-либо особой роли в организации.

Суперклассы и подклассы могут использоваться с целью ис-

ключения описания различных типов персонала с (возможно) разными атрибутами внутри одной сущности. Например, торговые агенты могут иметь особые атрибуты "Компенсация транспортных расходов", "Район сбыта" и т.д. Если все атрибуты сотрудников и особые атрибуты для выполнения отдельных работ будут описаны в одной сущности "сотрудник", то это может привести к появлению большого количества неопределенных значений атрибутов, описывающих отдельные виды работ. Очевидно, что подкласс "торговый агент" имеет указанные общие атрибуты со сведениями о других сотрудниках — например, таких как табельный номер, имя, адрес и зарплата. Можно также показать связи, которые имеются только для отдельных групп работников (подклассов), но не для всех сотрудников в целом.

Экземпляр сущности в подклассе обладает всеми атрибутами суперкласса, а также специфическими атрибутами подкласса.

Подкласс также является сущностью, а потому может иметь свои собственные подклассы.

Суперкласс и его подклассы, которые также являются сущностями, на ER-диаграмме обозначаются прямоугольниками. Подклассы соединяются линиями с кружком, который, в свою очередь, соединяется с суперклассом. Символ принадлежности множеству ($\subset$) на каждой линии, соединяющей подкласс с кружком, указывает направление связи "подкласс/суперкласс" (например, "менеджер" с "сотрудник"). Символ "o" в кружке обозначает, что подклассы пересекаются, т.е. сущность может быть членом сразу нескольких подклассов. Например, показанные на рис. 2 подклассы служебных ролей (менеджер, секретарь, торговый агент) являются пересекающимися. Это значит, что сотрудник может быть одновременно и менеджером (т.е. членом подкласса "менеджер"), и торговым агентом (т.е. членом подкласса "торговый агент").

Если подклассы не пересекаются, то каждая отдельная сущность может быть членом только одного из подклассов данного суперкласса. Для представления непересекающихся подклассов используется символ "d", который располагается в центре кружка, соединяющего подклассы данного суперкласса. Например, показанные на рис. 2 подклассы видов соглашений о найме ("постоянный работник" и "временный работник") являются непересекающимися. Это значит, что сотрудник может установить с компанией либо соглашение о полной постоянной занятости, либо соглашение о частичной временной занятости.

На связь "подкласс/суперкласс" может накладываться ограничение участия, которое может быть полным или частичным.

Полное участие означает, что каждый экземпляр суперкласса должен быть членом подкласса. Для обозначения полного участия между суперклассом и кружком, связывающим с подклассами, проводят двойную линию. Например, на рис. 2 показано полное участие, при котором каждый сотрудник компании должен установить с ней соглашение о полной постоянной или частичной временной занятости.

Частичное участие означает, что экземпляр суперкласса не обязательно должен быть членом любого подкласса. Для обозначения частичного участия между суперклассом и кружком, связывающим с подклассами, проводят одинарную линию. Например, на рис. 2 показано частичное участие суперкласса "сотрудник" в подклассах служебных ролей, при котором сотрудник не обязательно должен выполнять одну из дополнительных служебных ролей – менеджера (т.е. быть членом подкласса "менеджер"), секретаря (т.е. быть членом подкласса "секретарь") или торгового агента (т.е. быть членом подкласса "торговый агент").

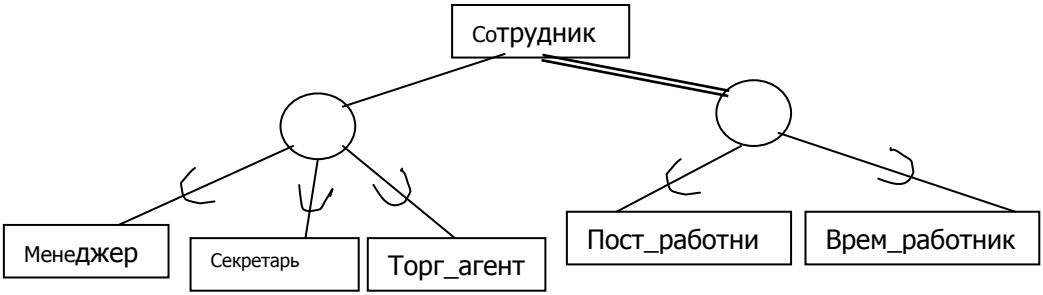


Рис.2

Задание на лабораторную работу

Изучите методические указания.

Используя методические указания, выполните следующие действия:

1. Создайте инфологическую модель данных в виде диаграммы «сущность-связь» для указанного варианта задания.
2. Определите атрибуты каждой сущности в модели, указав для них тип данных, размер, ограничение на значение.
3. Оформите отчет о выполненной работе.

Варианты заданий

№ 1

Строительная компания «Премьер» возводит различные здания. Для всех зданий требуются разнообразные материалы в различных количествах. На разных этапах проекта работают разные бригады. Например, есть бригады арматурщиков, каменщиков, штукатуров и т.д. Составляя график работ, фирма «Премьер» варьирует состав бригад. Рабочие назначаются в разные бригады в соответствии с квалификацией. Так, Петров может выполнят работу как плотника, так и каменщика, поэтому его иногда включают в бригаду арматурщиков, иногда каменщиков. Численность бригад меняется в зависимости от размера здания и предъявляемых к нему требований. Для каждой бригады выбираются дни работы. Например, бригаде штукатуров требуется несколько дней для того, чтобы оштукатурить здание. Для каждой бригады, работающей на строительстве данного здания, назначается бригадир. Рабочий может возглавлять одну бригаду и работать в другой простым рабочим. Владелец компании «Премьер» хочет знать, кто из его рабочих в каждую бригаду назначен на разных зданиях, какие материалы используются при возведении разных зданий, а также график работ по каждому зданию.

№ 2

Фирма «Парус» имеет конструкторские бюро, заводы и склады. В них проектируются, производятся и хранятся различные детали. Деталь проектируется только в одном конструкторском бюро, но может производиться на нескольких заводах и храниться на нескольких складах. Модель данных должна давать ответ на вопросы, подобные следующим:

Какие детали где спроектированы?

Где спроектирована и где произведена деталь, в которой обнаружен брак?

Какое количество деталей А235 находится на складе № 3?

№ 3

Консультационная фирма «Феникс» оказывает услуги по анализу, проектированию, программированию различных прикладных систем для клиентов. Проработав с множеством разных клиентов, работники фирмы обнаружили, что у клиентов часто возникают одни и те же потребности, и что для их нужд можно использовать одно и то же базовое ПО. Например, Иванову требуется система инвентарного учета и система расчета стоимости. Петрову нужна система расчета стоимости и система начисления

зарплаты. Создав обобщенные системы расчета стоимости, инвентарного учета, начисления зарплаты и т.д., фирма может удовлетворить потребности многих клиентов за меньшую цену. Отсюда возникла идея создания базовых систем в каждой из этих областей. Базовая система может иметь несколько версий. Каждая клиентская система создается на основе некоторых базовых систем определенных версий.

Когда проект завершен, фирма посылает клиенту счет на выполненную работу и затраченные материалы. Образец счета представлен ниже.

Консультационная служба «Феникс»
195000, Москва, Шаболовка,57

СЧЕТ

Дата	Номер счета	Проект
27.12	349	Система подсчета стоимости

Консультант	Вид деятельности	Часы	Ставка	Цена
Цветков	Системный анализ	30	60/час	1800.0
Цветков	Системное проектирование	30	60/час	1800.0
Цветков	Программирование	20	60/час	1200.0
Рыжов	Программирование	60	40/час	2400.0
Итого оплата консультантов				7200.0

ДРУГИЕ РАСХОДЫ

Описание	Цена
Материалы (бумага, ксерокопирование и т.д.)	35.00
Итого другие расходы	35.00
Итого к оплате	7235.00

Клиент: фирма «Парус»
195111, Москва, Петровка,18

Сколько клиентов?

У кого из клиентов есть и текущие счета, и сберегательные счета?

Каков процент сберегательных счетов, баланс которых не превышает 1000 руб?

Какой тип клиентов имеет самый высокий средний баланс текущих счетов?

Какой процент обладателей текущих счетов банка составляют его служащие?

Сколько кассиров имеют в банке сберегательные счета? Сколько менеджеров?

Кто из менеджеров, имеющих в банке сберегательные счета, руководит служащими, имеющими в банке сберегательные счета?

№ 5

Некто занимается выращиванием фруктов и владеет садами в различных областях. Он желает иметь базу данных, хранящую обширную информацию о его хозяйстве, которая позволит получить ответы на вопросы, подобные следующим:

Сколько сортов персиков в саду в селе Красное?

Сколько деревьев в год в среднем погибает в саду в селе Самарское?

Каков средний возраст яблонь?

Модель данных должна включать информацию о годе посадки каждого дерева. Если дерево погибло, то эта информация также должна быть зафиксирована. Деревья бывают разных видов, а внутри каждого вида существуют сорта. Например, яблоня – это вид, а Джонатан и Гольден – сорта. Поскольку дерево можно прививать, на данном дереве может быть более одного сорта. Так, к яблоне, которая исходно была сорта Гольден, могли также привить Джонатан и Айдаред. Каждое дерево относится к одному виду, но может нести несколько сортов. Разумеется, существует множество деревьев каждого вида и сорта.

Деревья в садах посажены продольными и поперечными рядами. Таким образом, расположение каждого дерева в саду можно определить номерами продольного и поперечного ряда. Когда дерево погибает, его выкорчевывают и впоследствии сажают на его место новое дерево. В зависимости от погодных условий весной разные сорта цветут в разное время. Сбор урожая начинается через определенное (для каждого сорта) количество дней после полного цветения. Модель данных должна давать ответы на вопросы, подобные следующим:

Сколько яблок сорта Гольден собрано в саду в селе Самарское в последнем сезоне?

Какой средний срок начала сбора урожая персиков сорта Клинг во всех наших садах за последние 10 лет?

Когда созреет Джонатан в саду в селе Самарском в этом году?

Сколько мест в каждом саду для новых деревьев?

Сколько этих мест будет, если мы выкорчуем деревья, чья средняя урожайность за последние 5 лет не превышала одного центнера?

№ 6

Модель данных должна давать ответы на следующие вопросы по истории Европы:

Сколько королей Пруссии носили имя Фредерик? В какие годы они жили и в какие – правили? Управляли ли они на протяжении своей жизни какими-либо еще странами? Управлялись ли в XVII веке какие-либо европейские страны женщинами? Если да, то какие?

Правил ли дед Марии-Антуанетты какой-либо страной? Какой и когда? Кто была его мать? Были ли случаи, когда правители двух разных стран женились между собой? Сколько детей Генриха VIII стали королями Англии? Кто были их матери?

№ 7

Некоторая телевизионная компания владеет несколькими телевизионными станциями. Эти станции транслируют программы телеагентств, коммерческую информацию и спортивные репортажи. Модель данных должна давать ответы на следующие вопросы:

Репортажи о скольких футбольных матчах телекомпания показала за последний год? Когда они транслировали встречи между командами «Динамо» и «Спартак»? Матчи какой команды показывались больше всего? Как насчет хоккейных матчей? Баскетбольных? Теннисных? Других видов спорта? Был ли показан хоть один теннисный матч с участием Кафельникова? Когда и какой станцией?

Какие коммерческие объявления телекомпания показала более трех раз в течение 1 часа на одной станции? Когда это было? В течение какого часа, какого числа и на какой станции? Какую плату телекомпания назначила за трансляцию каждого из этих коммерческих сообщений?

№ 8

Юридической конторе требуется модель данных, отвечаю-

щая на следующие юридические вопросы:

В каких делах высказывались мнения по разделу 411.3с федерального кодекса? В каких судах? Были ли они отвергнуты? Какой раздел федерального кодекса был затронут в процессе Романова А.И. против Коваленко В.П.?

Какие юридические фирмы представляли компанию «Аэрофлот» в судах в течение последних 10 лет? Какие дела разбирались, какая сторона выиграла, каков был размер вознаграждения? Какие фирмы представляли противную сторону? Какие еще крупные компании представляли эти юридические фирмы в процессах в то же самое время?

№ 9

Модель данных должны давать ответы на вопросы, подобные следующим:

Какие товары имеют продажную цену более 200 руб.? Какие из них имеют закупочную цену менее 150 руб.? Какие товары произведены в Белоруссии? Кто их изготовители?

Кто из продавцов продал товары ценой более 200 руб.? Даты этих продаж? Какова базовая зарплата этих продавцов?

У какого поставщика закупили данный товар и в каком объеме? Сколько данного товара находится на складе?

Кто работает в отделе маркетинга? Кто руководит отделом? Когда назначен новый заведующий отделом? Сколько продавцов в отделе продаж?

№ 10

Авиакомпания хочет иметь информацию о своих самолетах, чтобы получать ответы на вопросы, подобные следующим:

Сколько посадочных мест в ТУ-154?

Сколько у него двигателей?

Какой средний возраст ТУ-154 нашего авиапарка?

Кто главный механик, ответственный за обслуживание самолета № 1388?

Какая компания создала этот самолет?

На какие рейсы назначен данный самолет?

Когда производился последний ремонт данного самолета, и каков характер ремонта?

Какие самолеты находятся в ремонте?

№ 11

Фирма «Феникс» для осуществления своей деятельности заказывает и закупает у поставщиков некоторые расходные материалы. Информацию о заказах, а также об оплате этих заказов с указанием даты оплаты и номера чека нужно сохранять в базе

данных. Форма заказа фирмы представлена ниже.

Фирма «Феникс»
344078, Ростов-на-Дону, Текучева, 95

ЗАКАЗ

Дата	Номер заказа	Номер поставщика
29.03	388	23

Инвентарный номер	Описание товара	Количество	Цена	Сумма
3821	Карандаши № 2	3	4.00	12.00
4919	Блокноты	4	8.90	35.60
	Налог			2.86
			Итого	50.46

Поставщик: Объединенные Канцелярские Поставки,
344082, Ростов-на-Дону, Большая Садовая, 26

– таблицу атрибутов с указанием типов данных, размера, ограничений на значение, первичных ключей;

– описание установленных в модели связей между сущностями.

Контрольные вопросы

1. Как называется графическое представление концептуальной или инфологической модели данных?

2. Какие объекты включает инфологическая модель данных?

3. Для чего нужен первичный ключ сущности?

4. Какие виды бинарных отношений устанавливаются между сущностями?

ЛАБОРАТОРНАЯ РАБОТА НА ТЕМУ: ПРОЕКТИРОВАНИЕ СТРУКТУРЫ БД В MS ACCESS

Цель работы: изучить способы создания базы данных, таблиц базы данных и навыки проектирования схемы базы данных средствами СУБД **MS Access**.

Руководство к выполнению

Общее описание базы данных MS Access

MS Access – это реляционная система управления базой данных (СУБД). В реляционных базах данных информация хранится в таблицах. Таблица содержит информацию об объектах определенного типа конкретной предметной области. Каждая таблица состоит из строк и столбцов, которые называются записями и полями. Каждую строку можно рассматривать как единичную запись. Информация внутри записи состоит из полей. Например, таблица «Заказы» имеет следующую структуру:

Номер заказа	Код покупателя	Код товара	Дата заказа	Заказано	Дата продажи	Продано	Сумма
1	2	3	1.01.06	10	12.01.06	8	160
2	2	1	2.01.06	2	12.01.06	2	260
3	6	2	5.01.06	20	23.01.06	20	6200

Связь между таблицами осуществляется посредством значений одного или нескольких совпадающих полей. Например, можем построить таблицу «Товары»:

Код товара	Наименование	Стоимость	Цена
1	Шампунь	50	60
2	Настольная лампа	190	230
3	Свитер	350	

Тогда указанные две таблицы можно связать с помощью поля «Код товара».

База данных MS Access состоит из отдельных компонентов, которые используются для хранения и представления информации. Этими компонентами являются таблицы, формы, отчеты, запросы, макросы, модули. Для создания форм и отчетов используются конструкторы, поэтому эти компоненты часто называют конструкторскими объектами. Конструкторские объекты являются составными объектами, то есть состоят из более мелких объектов (таких как поля, кнопки, диаграммы, рамки и т.д.), которые назы-

ваются объектами интерфейса.

Таблица является основой базы данных. В MS Access вся информация содержится в таблицах.

Формы используются для ввода и просмотра таблиц в окне формы. Формы позволяют ограничить объем информации, отображаемой на экране, и представить ее в требуемом виде. С помощью конструктора форм можно создавать формы любой степени сложности.

Отчеты используются для отображения информации, содержащейся в базе данных. С помощью конструктора отчетов можно разработать собственный отчет.

Запрос является средством извлечения информации из базы данных, причем данные могут извлекаться из нескольких таблиц.

Макросы предназначены для автоматизации часто выполняемых операций. Каждый макрос содержит одну или несколько макрокоманд, каждая из которых выполняет определенное действие, например, открывает форму или печатает отчет.

Для запуска MS Access можно использовать меню Пуск Windows. Для этого с помощью мыши выберите команду Программы меню Пуск, а затем – Microsoft Access.

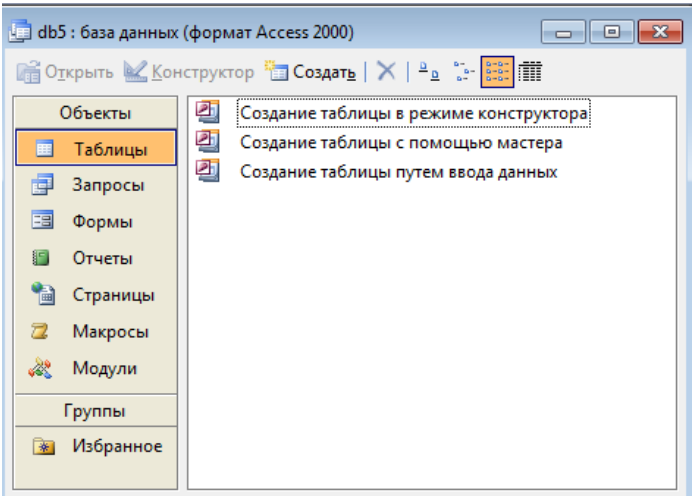
Альтернативный способ запуска – выбор Microsoft Access с помощью проводника Windows или выбор ярлыка, созданного для запуска MS Access.

После запуска Access на экране появится окно приглашения Access. В этом окне диалога имеются 2 основные опции, предназначенные для создания новой и открытия ранее созданной базы данных.

После открытия существующей базы данных на экране появится главное окно Access, содержащее окно базы данных, строку меню, панель инструментов и строку состояния.

MS Access хранит все таблицы, отчеты, формы, макросы и модули в базе данных.

Окно базы данных состоит из 6 вкладок: «Таблицы», «Запросы», «Формы», «Отчеты», «Макросы», «Модули». Каждая вкладка содержит объекты текущей базы данных, тип которых указан ярлыком вкладки:



В верхней части окна базы данных находятся кнопки **Открыть**, **Конструктор** и **Создать**. Кнопка **Открыть** служит для открытия выбранного объекта. Этим объектом может быть таблица, запрос или форма. При открытии таблицы открывается окно таблицы, и можно просматривать и редактировать данные таблицы. Открытие запроса приводит к выполнению выбранного запроса и просмотра результатов выборки. При открытии формы можно просматривать и редактировать данные, но уже не в окне таблицы, а в окне формы.

При переходе на вкладку «Отчеты» наименование кнопки поменяется на кнопку «Запуск».

Кнопка **Конструктор** предназначена для модификации выбранного объекта, а кнопка **Создать** – для его создания.

Меню «Файл» в Access

Меню Файл включает ряд команд.

Команды Создать и Открыть относятся к базе данных. Кроме команд печати, просмотра, закрытия окон, меню содержит команды экспорта и импорта данных, а также установления связи с внешними таблицами.

В нижней части меню Файл находится команда Выход. Она означает выход из Access вообще, а не из текущего окна. Чтобы просто закрыть окно, используется команда Закрыть.

СОЗДАНИЕ БАЗЫ ДАННЫХ

В Access можно создать пустую базу данных, а затем добавить в нее таблицы, формы, отчеты и другие объекты.

Для создания новой базы данных выполните следующие действия:

1. Выполните команду **Файл/Создать**.
2. В открывшемся окне диалога «Создание» выберите «Новая база данных».
3. В открывшемся окне диалога «Файл новой базы данных» из раскрывающегося списка Папка выберите папку, в которой хотите сохранить создаваемую базу данных, а в поле Имя файла введите ее имя. Затем нажмите кнопку **Создать**.

Создание таблиц

Создание таблицы в Access осуществляется в окне базы данных. Рассмотрим последовательность действий при создании таблицы в новой базе данных:

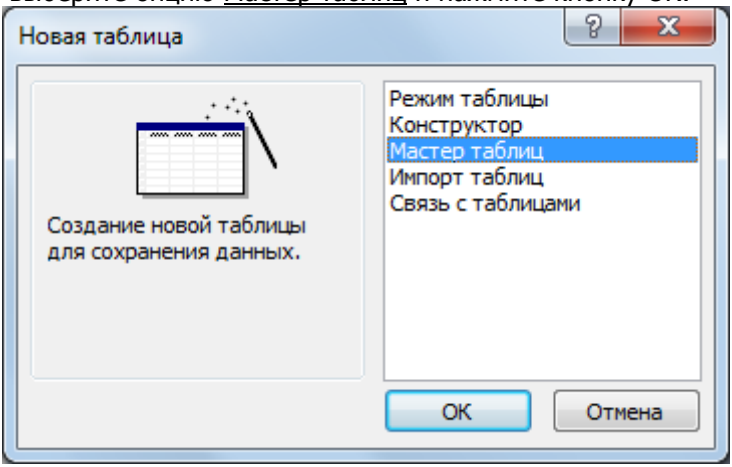
1. Откройте окно созданной вами базы данных и перейдите на вкладку «Таблицы».
2. Нажмите кнопку **Создать** в окне базы данных.
3. Откроется окно диалога «Новая таблица», в правой части которого находится список вариантов дальнейшей работы:
   - Режим таблицы – позволяет создать новую таблицу в режиме таблицы
   - Конструктор - позволяет создать новую таблицу в конструкторе таблиц
   - Мастер таблиц - позволяет создать новую таблицу с помощью мастера
   - Импорт таблиц – позволяет осуществить импорт таблиц из внешнего файла в текущую базу данных
   - Связь с таблицами – позволяет осуществить создание таблиц, связанных с таблицами из внешних файлов
4. Выберите подходящий вам вариант создания таблицы и нажмите кнопку ОК.
5. Создайте структуру таблицы с помощью выбранного средства.
6. Чтобы связать таблицу с содержащейся в ней информацией, каждой таблице присваивается имя. Задайте имя таблицы в окне диалога «Сохранение» и нажмите кнопку ОК.

Рассмотрим отдельные способы создания таблиц.

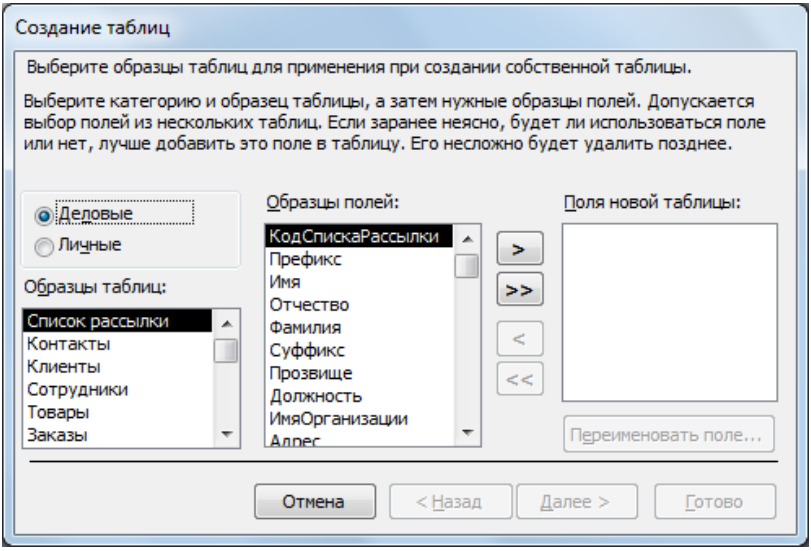
Создание таблицы с помощью мастера

Access содержит целый ряд таблиц, которые можно использовать в качестве прототипов требуемых вам таблиц.

Чтобы вызвать мастера таблиц, в окне диалога «Новая таблица» выберите опцию Мастер таблиц и нажмите кнопку ОК:



На экране откроется окно диалога, в левой части которого находится список Образцы таблиц, рядом список Образцы полей, содержащий предлагаемые образцы полей для выбранной таблицы:



Сначала надо из списка Образцы таблиц выбрать прототип таблицы, которая похожа на создаваемую вами таблицу. Затем из списка Образцы полей выберите поля таблицы и разместите их в списке Поля новой таблицы.

Для выбора полей используйте кнопки со стрелками, кото-

рые расположены правее списка Образцы полей.

Назначение этих кнопок следующее:

>	Выделенное в списке Образцы полей поле добавляется в список полей создаваемой таблицы
>>	Все поля из списка Образцы полей добавляются в список полей создаваемой таблицы
<	Удаляет из списка полей создаваемой таблицы поле, выделенное в списке Поля новой таблицы
<<	Удаляет все элементы из списка Поля новой таблицы

Сформировав список полей создаваемой таблицы, нажмите кнопку **Далее**.

!!! Мастер таблиц сам устанавливает типы данных для полей таблиц. После того как таблица создана, можно перейти в режим конструктора таблиц и посмотреть, какие типы данных выбрал мастер для полей.

На следующем шаге создания таблицы задается имя таблицы и определяется ключевое поле.

Мастер предлагает свой вариант имени, который можно принять, нажав клавишу **Tab**. Если хотите присвоить таблице другое имя, введите его в текстовое поле.

Теперь можно указать мастеру, чтобы он автоматически подобрал для таблицы первичный ключ, и нажать кнопку **Далее** для перехода в следующее окно диалога.

Если в базе данных уже существуют ранее созданные таблицы, на третьем шаге мастер поможет связать создаваемую таблицу с уже созданными.

Для связывания создаваемой таблицы с другими таблицами базы данных выберите из списка уже существующих в базе данных таблиц таблицу, с которой хотите ее связать, и нажмите кнопку **Связи**. На экране откроется окно диалога «Связи». Установите один из 2 возможных типов создаваемых связей и нажмите кнопку ОК.

Вы указали всю необходимую информацию для создания таблицы. На следующем шаге можете указать режим вашей дальнейшей работы:

- Изменение структуры таблицы – после завершения работы мастера на экране открывается режим конструктора для возможной модификации созданной структуры таблицы

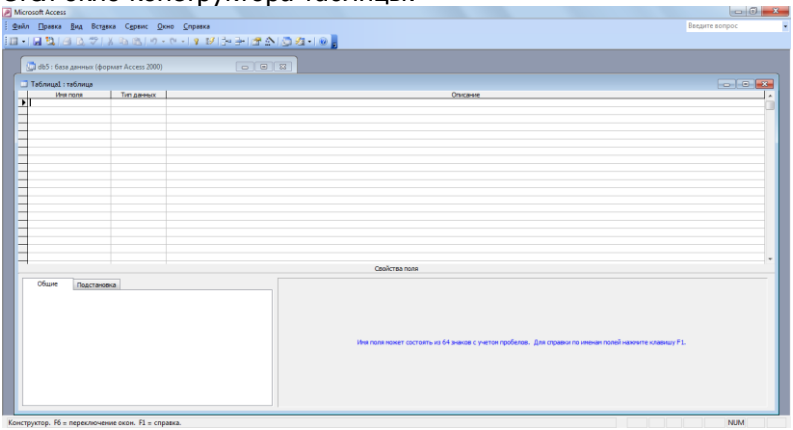
– Непосредственный ввод данных в таблицу - после завершения работы мастера на экране открывается режим таблицы для непосредственного ввода данных в нее

– Ввод данных в таблицу с помощью формы, создаваемой мастером – после завершения работы мастера на экране открывается экранная форма ввода информации в созданную таблицу.

Установив требуемый режим, нажмите кнопку **Готово**. Этим вы запустите мастер на формирование таблицы.

Создание таблицы в окне конструктора

Перейти в конструктор таблиц можно из мастера по созданию таблиц или же непосредственно из окна диалога «Новая таблица», выбрав из списка вариантов значение Конструктор и нажав кнопку ОК. В результате выполнения этих действий откроется окно конструктора таблицы:



В верхней части окна диалога находится таблица, которая содержит следующие атрибуты создаваемой таблицы: имя поля, тип данных и описание. Кроме этих основных атрибутов, каждое поле таблицы обладает дополнительными свойствами, отображаемыми в нижней части конструктора и определяющими условия ввода данных.

Имя поля используется для ссылки на данные таблицы. Для определения типа хранимых данных используется тип данных. В Access допустимыми являются данные следующих типов:

– Текстовый
– Числовой
– Денежный
– Счетчик

- Дата/время
- Логический
- Поле MEMO (текстовое поле произвольной длины – до 64000 символов)
- Поле объекта OLE
- Мастер подстановок.

При вводе имени поля по умолчанию Access присваивает ему текстовый тип данных с шириной поля, равной 50. Текстовые поля могут содержать буквы, цифры и специальные символы. Максимальная ширина поля составляет 255 символов.

Поле объекта OLE – тип данных, который служит для хранения в таблицах изображения и других двоичных данных (например, электронной таблицы Excel, документа Word, рисунка, звукозаписи). Фактический объем данных, который можно ввести в поле данного типа, определяется объемом жесткого диска компьютера.

Для создания структуры таблицы выполните следующие действия:

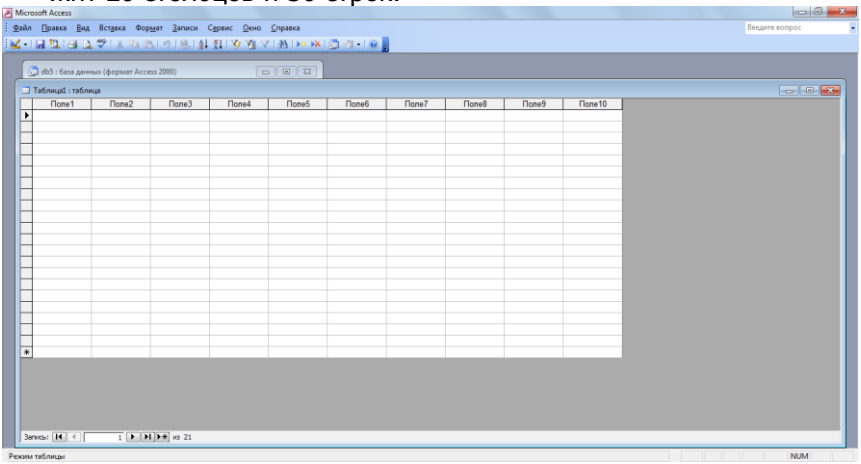
1. В окне конструктора таблицы в столбце Имя поля введите ваше имя поля.
2. Нажмите **Tab** или **Enter**, чтобы перейти в столбец Тип данных.
3. В столбце Тип данных появилось значение Текстовый. Нажмите на кнопку раскрытия списка в правой части прямоугольника, и увидите список, содержащий все типы данных. Из этого списка выберите значение и нажмите на клавишу **Tab** для перехода в столбец Описание. В этом столбце можно дать пояснение своим полям.
4. Введите поясняющий текст в столбец Описание и нажмите клавишу **Tab** или **Enter**, чтобы перейти к вводу информации о следующем поле.
5. Аналогично введите описание всех полей таблицы.
6. Завершив ввод структуры таблицы, сохраните ее, выполнив команду **Файл/Сохранить**.

Создание таблицы в режиме таблицы

Это третий и самый простой способ создания таблицы. Выполните следующие действия:

1. Перейдите на вкладку «Таблицы» окна базы данных и нажмите кнопку **Создать**.
2. В окне диалога «Новая таблица» выберите из списка вариантов значение Режим таблицы и нажмите кнопку ОК. В результате откроется окно диалога «Таблица», содер-

жащее созданную по умолчанию таблицу. Эта таблица содержит 20 столбцов и 30 строк:



После сохранения этой таблицы можно добавить столько строк и столбцов, сколько вам понадобится.

3. Наименования полей таблицы определены по умолчанию. Access позволяет присвоить полям новые имена. Для этого нажмите дважды кнопкой мыши на область выбора первого поля (заголовок которого содержит Поле1). Имя поля выделяется и появляется мигающий курсор. Введите имя первого поля и нажмите клавишу **Tab**. Аналогично введите остальные имена полей таблицы.

4. Теперь заполните несколько строк таблицы, вводя информацию в том виде, в каком она будет вводиться и в будущем.

5. Сохраните таблицу, выполнив команду **Файл/Сохранить макет** или нажав кнопку **Сохранить** на панели инструментов. В открывшемся окне диалога «Сохранение» присвойте таблице имя и нажмите кнопку ОК.

6. На запрос о необходимости создания для таблицы первичного ключа нажмите кнопку **Да**, и Access создаст таблицу, удалив лишние строки и столбцы.

7. Теперь убедитесь, что Access выбрал для каждого поля правильные типы данных. Для этого перейдите в окно конструктора таблицы, выполнив команду **Вид/Конструктор таблиц**. Если вас что-то не устраивает в структуре таблицы, внесите необходимые изменения.

Модификация структуры таблицы

Для модификации структуры таблицы установите в окне ба-

зы данных указатель на модифицируемую таблицу и нажмите кнопку **Конструктор**. На экране откроется окно конструктора, содержащее структуру выбранной таблицы.

Изменение наименования поля и его типа

Установите указатель на наименование поля или тип, который требуется изменить. Для удаления неправильных символов в имени поля используйте клавишу Backspace или Delete. После этого введите правильное имя поля.

Изменение типа поля осуществляется аналогично его присвоению.

Изменение порядка следования полей

1. Слева от имен полей находится область выбора поля. Перейдите на строку с описанием поля, которое хотите переместить, и нажмите на эту область. При этом вся строка будет выделена.
2. Теперь нажмите еще раз на область выбора поля и не отпускайте кнопку мыши. Под курсором мыши появился маленький пунктирный прямоугольник.
3. Перенесите всю строку так, чтобы она оказалась прямо над тем полем, перед которым вы хотите ее расположить.
4. Отпустите кнопку мыши. Поле теперь находится на новом месте.
5. С помощью команды **Правка/Отменить сдвиг** можно вернуть перемещенное поле на прежнее место.

Удаление полей из таблицы

В конструкторе таблиц нажмите на область выбора поля, которое хотите удалить, и нажмите клавишу **Delete**.

Чтобы удалить несколько соседних полей одновременно:

1. Нажмите мышью на область выбора первого поля из тех, которые хотите удалить.
2. Не отпуская кнопку мыши, нажмите и удерживайте клавишу **Shift**.
3. Нажмите мышью последнее поле из тех, которые хотите удалить. После этого первый, последний и все строки между ними будут выделены.
4. Отпустите клавишу **Shift**.
5. Нажмите клавишу **Delete** для удаления всех выделенных полей.

Аналогично можно использовать клавишу **Ctrl** для выбора и удаления полей, которые расположены не подряд.

Подобные действия можно выполнить, чтобы одновременно

перенести несколько полей. Выберите их одним из описанных способов и перенесите в нужное место.

Добавление нового поля

Выполните команду **Вставка/Поле**. Новая строка будет вставлена над текущей строкой, то есть той строкой, в которой находится курсор (она отмечена стрелкой в зоне области выбора поля).

Другой способ:

Нажмите правую кнопку мыши на строке, находящейся ниже того места, куда вы собираетесь вставить новое поле. Выберите пункт всплывающего меню Вставить поле для добавления нового поля. Этот способ выбора можно использовать и для удаления полей.

Индексы

Одним из основных требований, предъявляемых к СУБД, является возможность быстрого поиска требуемых записей среди большого объема информации. Индексы – наиболее эффективное средство, которое позволяет ускорить поиск данных в таблицах по сравнению с таблицами, не содержащими индексов. В зависимости от количества полей, используемых в индексе, различают простые и составные индексы.

В MS Access допускается создание произвольного количества индексов. Индексы создаются при сохранении макета таблицы и автоматически обновляются при вводе и изменении записей. Можно добавить новые или удалить ненужные индексы в окне конструктора таблиц.

Индексы могут быть уникальными и неуникальными. Если индекс уникальный, то для таблицы, содержащей только одно индексное поле, уникальными должны быть значения этого поля. Для составных индексов значения в каждом из индексных полей могут повторяться. Однако индексное выражение (комбинация значений всех полей, входящих в индекс) должно быть уникальным.

Создание простого индекса

Для создания простого индекса используется свойство поля Индексированное поле, позволяющее ускорить выполнение поиска и сортировки записей по одному полю таблицы. Индексированное поле может содержать как уникальные, так и повторяющиеся значения.

Данное свойство поля Индексированное поле может принимать следующие значения:

Нет

Значение по умолчанию. Индекс не создается.

Да (Допускаются совпадения) — В индексе допускаются повторяющиеся значения.

Да (Совпадения не допускаются) — Повторяющиеся значения в индексе не допускаются.

Для создания простого индекса выполните следующие действия:

1. В окне конструктора таблицы выберите в верхней половине окна поле, для которого создается индекс.
2. В нижней половине окна для свойства Индексированное поле выберите одно из двух последних значений: «Да (Допускаются совпадения)» или «Да (Не допускаются совпадения)».

!!! Не допускается создание индексов для полей MEMO и полей объектов OLE.

Создание составного индекса

Индексы, содержащие несколько полей, следует определять в окне индексов.

1. В окне конструктора откройте таблицу, для которой вы создаете составной индекс. Для этого в окне базы данных установите указатель на данную таблицу и нажмите кнопку **Конструктор**.
2. Нажмите кнопку **Индексы** на панели инструментов. На экране откроется окно диалога «Индексы».
3. Введите имя индекса в поле столбца Индекс в первой пустой строке. В качестве имени индекса можно использовать имя одного из полей, включенных в индекс, или любое допустимое имя.
4. В столбце Имя поля той же строки нажмите кнопку раскрытия списка и выберите первое поле индекса.
5. В столбце Имя поля следующей строки выберите имя следующего поля индекса. (В этой строке поле столбца Индекс следует оставить пустым). Определите таким же образом остальные поля индекса. Индекс может содержать до 10 полей.
6. Закончив выбор полей для индекса, нажмите кнопку закрытия окна, расположенную в строке заголовка окна диалога.

!!! По умолчанию задается порядок сортировки По возрас-

танию. Для сортировки конкретного поля по убыванию выберите в столбце Порядок сортировки строки с выбранным полем значение По убыванию.

Установка первичного ключа

Когда MS Access напоминает об отсутствии первичного ключа и предлагает его создать, нажмите **Да**. Access добавит поле «Код» в первой строке описания структуры таблицы. Поле «Код» имеет тип данных «Счетчик». Это значит, что каждый раз при создании новой записи значение счетчика увеличивается на 1. Этот номер и будет первичным ключом для каждой новой записи.

Если хотите установить первичный ключ самостоятельно, нажмите на поле, которое будет в дальнейшем использоваться в качестве первичного ключа. Затем нажмите правую кнопку мыши и выберите пункт всплывающего меню Первичный ключ. В области выбора поля, которое будет использоваться как ключ, появится пиктограмма с изображением ключа. Для установки первичного ключа также можно использовать кнопку Ключевое поле на панели инструментов.

Создание схемы базы данных в MS Access

Определение связей между таблицами

Связи между двумя таблицами устанавливаются с помощью полей этих таблиц, содержащих одну и ту же информацию. Чаще всего связывают первичный ключ одной таблицы с совпадающими полями другой таблицы.

!!! Поля, с помощью которых устанавливается связь между таблицами, могут иметь различные имена, но удобнее использовать совпадающие имена.

Существующие типа связей: «один – ко – многим», «многие – ко – многим», «многие – к – одному», «один – к – одному». Первый тип наиболее важен.

Таблица, которая составляет часть «один» в отношении «один – ко многим», называется главной таблицей, а таблица со стороны «много» - подчиненной таблицей. Отношение «один – ко многим» реализуется с помощью внешнего ключа.

Внешний ключ – это поле (или поля) подчиненной таблицы, содержащее такой же тип информации, что и первичный ключ главной таблицы.

Отношение «один – ко – многим» создается посредством связывания первичного ключа главной таблицы с внешним ключом подчиненной.

Создание связей между таблицами в MS Access осуществля-

ется в окне диалога «Схема данных».

Перед определением связей между таблицами надо предварительно закрыть все открытые таблицы. Не допускается создание или удаление связей между открытыми таблицами.

Для определения связей между таблицами необходимо выполнить следующие действия:

1. Откройте окно диалога «Схема данных», выполнив команду **Сервис/Схема данных** или нажав кнопку **Схема данных** на панели инструментов. На экране откроется окно диалога «Схема данных».

2. Добавьте в это окно диалога последовательно 2 связываемые таблицы. Для этого выполните команду **Связи/Показать таблицу** или нажмите кнопку **Добавить таблицу** на панели инструментов. На экране откроется окно диалога «Добавление таблицы».

3. В списке таблиц выделите первую добавляемую таблицу и нажмите кнопку **Добавить**. Затем выберите вторую добавляемую таблицу и также нажмите кнопку **Добавить**. Затем нажмите кнопку **Закрыть** для закрытия окна «Добавление таблицы». В окне «Схема данных» появились две связываемые таблицы.

4. Для связывания таблиц выберите поле в первой связываемой таблице и переместите его с помощью мыши на соответствующее поле второй таблицы. Для связывания сразу нескольких полей выберите эти поля при нажатой клавише **Ctrl** и переместите во вторую таблицу группу выделенных полей.

В большинстве случаев связывают ключевое поле (представленное в списке полей полужирным шрифтом) одной таблицы с соответствующим ему полем внешнего ключа во второй таблице. Связанные поля должны иметь одинаковые типы данных и иметь содержимое одного типа. Кроме того, связанные поля типа Числовой должны иметь одинаковые значения свойства Размер поля. Исключениями из этого правила являются поля счетчика с последовательной нумерацией, которые могут связываться с числовыми полями размера Длинное целое, а также поля счетчика с размером Код репликации, связываемые с полями типа Числовой, для которых также задан размер Код репликации.

5. На экране откроется окно диалога «Связи». В данном окне проверьте правильность имен связываемых полей, находящихся в столбцах. При необходимости выберите другие имена полей. Затем нажмите кнопку **Создать**. Вы вернетесь в окно «Схема данных», где между таблицами будет указана линия

связи.

Тип создаваемой связи зависит от полей, которые были указаны при определении связи:

– отношение «один – ко многим» создается в том случае, когда только одно из связываемых полей является ключевым или имеет уникальный индекс;

– отношение «один – к – одному» создается в том случае, когда оба связываемых поля являются ключевыми или имеют уникальные индексы;

– связь «многие – ко – многим» представляется двумя связями «один – ко – многим» через третью таблицу. Ключ третьей таблицы состоит, по крайней мере, из двух полей, являющихся полями внешнего ключа в двух других таблицах.

!!! В окне «Схема данных» при переносе поля, не являющегося ключевым, на другое поле, которое также не является ключевым, создается неопределенное отношение, В запросах, содержащих таблицы с неопределенным отношением, MS Access по умолчанию создает линию объединения между таблицами, но условия целостности данных при этом не поддерживаются и нет гарантии уникальности записей в любой из таблиц.

В окне «Схема данных» имеется возможность не только создавать связи между таблицами, но и выполнять следующие действия:

- изменить структуру таблицы
- изменить существующую связь
- удалить связь
- удалить таблицу из окна «Схема данных»
- вывести на экран все существующие связи или связи только для конкретной таблицы
- определить связи для запросов, не задавая условия целостности данных.

Связывание двух полей одной таблицы

Для связывания поля таблицы с другим полем той же таблицы дважды добавьте эту таблицу в окно диалога «Схема данных» и создайте требуемую связь, соединив поля линией связи.

Создание между таблицами отношения «многие – ко – многим»

В MS Access отношение «многие – ко –многим» представляет две связи с отношением «один – ко – многим» через третью таблицу, ключ которой состоит, по крайней мере, из двух полей, являющихся полями внешнего ключа в двух других таблицах.

Рассмотрим создание такой связи:

1. Создайте таблицы, между которыми требуется определить связь с отношением «многие – ко – многим».
2. Создайте третью (связующую) таблицу с полями, описание которых совпадает с описанием ключевых полей в каждой из двух связываемых таблиц. В этой таблице указанные поля играют роль внешнего ключа. Другие поля в связующую таблицу можно добавлять без ограничений.
3. Определите в связующей таблице ключ, содержащий все ключевые поля двух связываемых таблиц.
4. Определите связи с отношением «один – ко – многим» между каждой из двух таблиц и связующей таблицей.

ИЗМЕНЕНИЕ СТРУКТУРЫ ТАБЛИЦЫ В ОКНЕ ДИАЛОГА «СХЕМА ДАННЫХ»

1. Находясь в окне «Схема данных», установите указатель мыши на модифицируемую таблицу.
2. Нажмите правую кнопку мыши и выберите из контекстного меню команду Конструктор таблиц.
3. Внесите в структуру таблицы необходимые изменения.
4. Закончив внесение изменений, нажмите кнопку закрытия окна в строке заголовка окна диалога. В ответ на запрос о сохранении изменений выберите «Да» для сохранения изменений и возвращения в окно диалога «Схема данных».

ИЗМЕНЕНИЕ СУЩЕСТВУЮЩЕЙ СВЯЗИ

Прежде чем приступить к изменению связей между таблицами, закройте все открытые таблицы. Затем выполните следующую последовательность действий:

1. Находясь в окне базы данных, нажмите кнопку **Схема данных** на панели инструментов.
2. Если таблицы, связи между которыми требуется изменить, не отображаются в окне «Схема данных», нажмите кнопку Добавить таблицу на панели инструментов, установите указатель на имя нужной таблицы и дважды нажмите кнопку мыши. После этого нажмите кнопку Закрыть.
3. Установите указатель на линию связи, которую требуется изменить, и дважды нажмите кнопку мыши.
4. В открывшемся окне диалога «Связи» внесите нужные изменения и нажмите кнопку ОК.

УДАЛЕНИЕ СВЯЗИ

1. Находясь в окне базы данных, нажмите кнопку Схема данных на панели инструментов.
2. Установите указатель на линию связи, которую требуется удалить, и выделите ее, нажав кнопку мыши.
3. Нажмите клавишу **Delete**. Когда Access предложит вам подтвердить удаление связи, нажмите кнопку «Да».

УДАЛЕНИЕ ТАБЛИЦЫ ИЗ МАКЕТА СХЕМЫ ДАННЫХ

1. Откройте окно «Схема данных».
2. Выберите таблицу, которую требуется удалить из данного окна, и нажмите клавишу **Delete**. Таблица будет удалена из макета схемы данных вместе с определенными для нее связями.

Данная операция изменяет макет только в окне диалога «Схема данных». И таблица, и ее связи будут по-прежнему сохраняться в базе данных.

ОПРЕДЕЛЕНИЕ УСЛОВИЙ ЦЕЛОСТНОСТИ ДАННЫХ

Целостность данных является одним из самых важных требований, предъявляемых к базе данных. Для задания условий целостности данных служат установленные между таблицами связи.

Условиями целостности данных называют набор правил, используемых в MS Access для поддержания связей между записями в связанных таблицах. Эти правила делают невозможным случайное удаление или изменение связанных данных.

При определении условия целостности данных действуют следующие ограничения:

- Невозможно ввести в поле внешнего ключа связанной таблицы значение, не содержащееся в ключевом поле главной таблицы. Однако возможен ввод в поле внешнего ключа пустых значений, показывающих, что записи не являются связанными. Например, нельзя сохранить запись о заказе, сделанном несуществующим клиентом, но можно создать запись для заказа, который пока не отнесен ни к одному из клиентов, если ввести пустое значение в поле КодКлиента.
- Не допускается удаление записи из главной таблицы, если существуют связанные с ней записи в подчиненной таблице.
- Невозможно изменить значение ключевого поля в

главной таблице, если имеются записи, связанные с этой записью.

Определение целостности данных предполагает выполнение следующих действий:

1. В окне «Схема данных» нажмите дважды мышью на линию связи между двумя таблицами. Откроется окно «Связь».
2. Установите флажок «Обеспечение целостности данных». Нажмите ОК.

После этого в окне «Схема данных» линия между двумя списками полей стала темнее и около нее появились 2 новых символа. Около главной таблицы теперь стоит символ «1», который указывает на часть «один» связи «один – ко – многим». Рядом с подчиненной таблицей отображается символ « ∞ », который обозначает часть связи «много».

Теперь любая попытка выполнить действия, нарушающие перечисленные выше ограничения, приведет к открытию окна с предупреждением, а само действие не будет выполнено.

Чтобы преодолеть ограничения на удаление или изменение связанных записей, сохраняя при этом целостность данных, следует включить режимы каскадного обновления и каскадного удаления. При установленном флажке Каскадное обновление связанных полей изменение значения в ключевом поле главной таблицы приводит к автоматическому обновлению соответствующих значений во всех связанных записях. При установленном флажке Каскадное удаление связанных записей удаление записи в главной таблице приводит к автоматическому удалению связанных записей в подчиненной таблице. Таким образом, при выборе этих опций MS Access выполняет изменения в связанных таблицах таких образом, чтобы сохранить целостность данных, даже если вы изменяете значения ключевых полей или удаляете запись в главной таблице.

Задание на лабораторную работу

1. Изучите методические указания.
2. Используя методические указания, выполните следующие действия:
3. Создайте базу данных с помощью мастера.
4. Создайте пустую базу данных для предметной области, описанной в построенной вами концептуальной модели в соответствии с вариантом задания предыдущей лабораторной работы.
5. Создайте таблицы базы данных, используя три

способа их построения.

6. Постройте схему базы данных, связав созданные таблицы.

7. Установите правила целостности для вашей базы данных, обеспечив при этом возможность внесения изменений в данные.

8. Оформите отчет о выполненной работе.

Требования к отчету по лабораторной работе

Отчет должен содержать:

- описание таблиц и их полей с указанием имен, типов данных, размера полей, первичных ключей,
- схему данных,
- описание связей таблиц с указанием типа каждой связи, первичных и внешних ключей, участвующих в организации связи, и используемых режимов обеспечения целостности данных.

Контрольные вопросы

1. Какие компоненты используются для хранения и представления информации в Access?
2. Какие способы создания БД поддерживаются в Access?
3. Какие способы создания таблиц БД поддерживаются в Access?
4. В каком окне БД Access устанавливаются связи между таблицами?
5. Каким образом реализуется связь «один – ко многим»?
6. От чего зависит тип создаваемой связи между таблицами?
7. Как связать поле таблицы с другим полем этой же таблицы?
8. Как создать связь «многие – ко – многим»?
9. Какие режимы поддержки целостности данных можно установить в БД Access?

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ: «РАЗРАБОТКА БАЗЫ ДАННЫХ ACCESS С ПОМОЩЬЮ ПРОГРАММНОГО ПАКЕТА POWER DESIGNER»

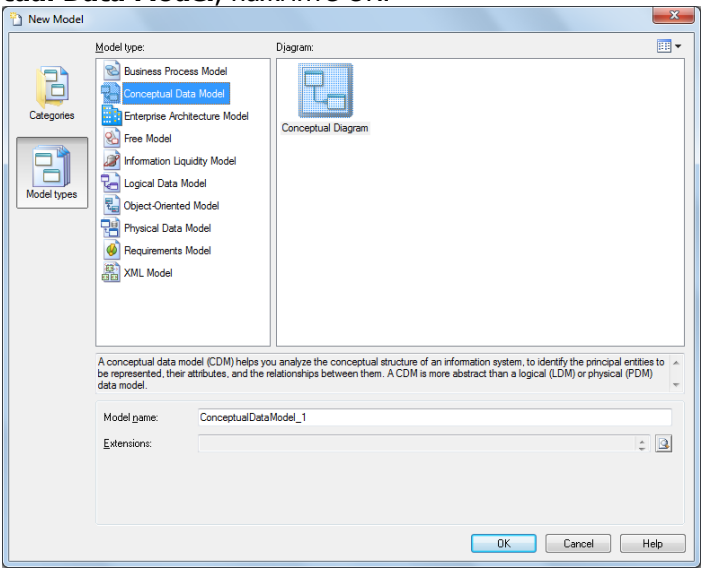
Цель работы: получить навыки автоматизированного построения структуры базы данных с помощью пакета **Power Designer**.

Руководство к выполнению

Пакет **Power Designer** представляет собой инструментарий для автоматизированного построения базы данных.

Создание концептуальной модели в Power Designer (CDM-модели).

Запустите программу **Power Designer**. В открывшемся окне выберите в меню **File** → **New Model** → **Model types** → **Conceptual Data Model**, нажмите ОК:

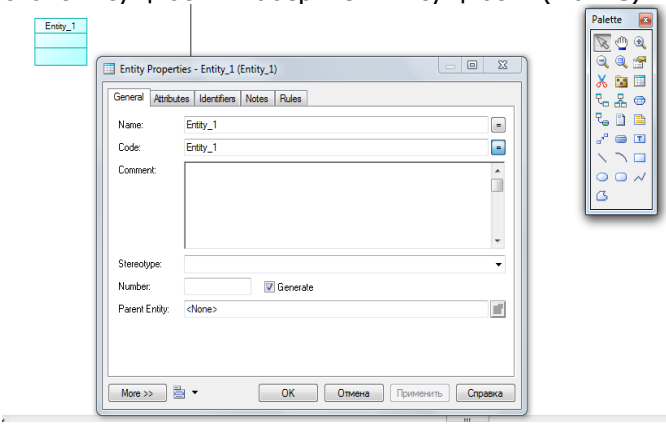


Откроется окно с панелью инструментов, в котором можно построить концептуальную модель базы данных.

Создание сущности

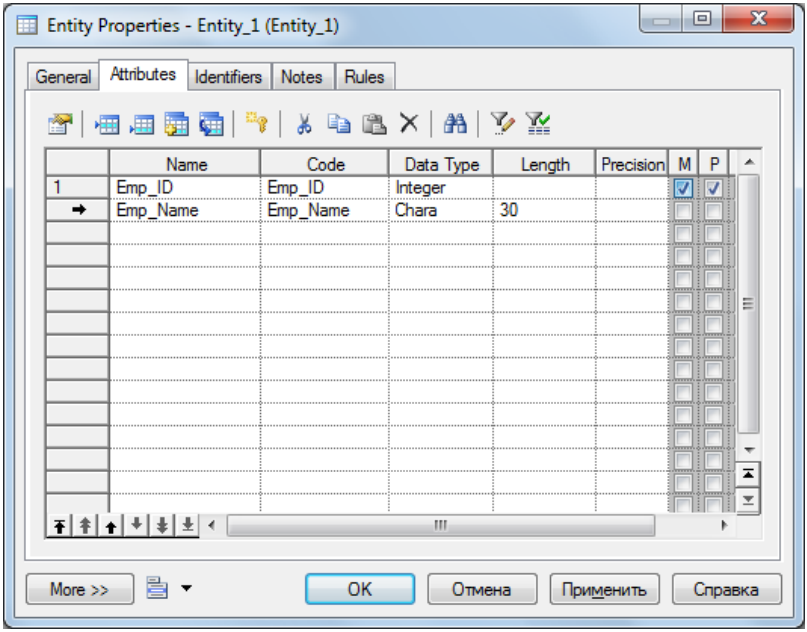
Нажмите кнопку **Entity** на инструментальной панели, затем щелкните где-нибудь в окне модели. В позиции щелчка появится символ сущности.

Дважды щелкните на символ сущности при выделенном указателе "Pointer" на инструментальной панели. Появится список свойств сущности. Наберите имя сущности (**Name**):



Нажмите кнопку **Attributes**. Появится окно атрибутов сущности. Задайте имена атрибутов, выберите для них типы данных, установите первичные ключи (флажок "P").

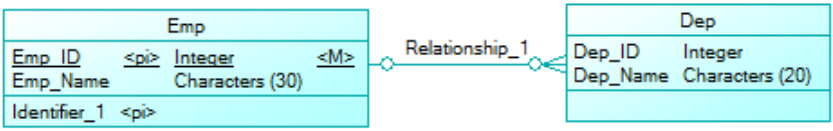
Примечание. Для символьных полей выбирайте тип данных **Characters**.



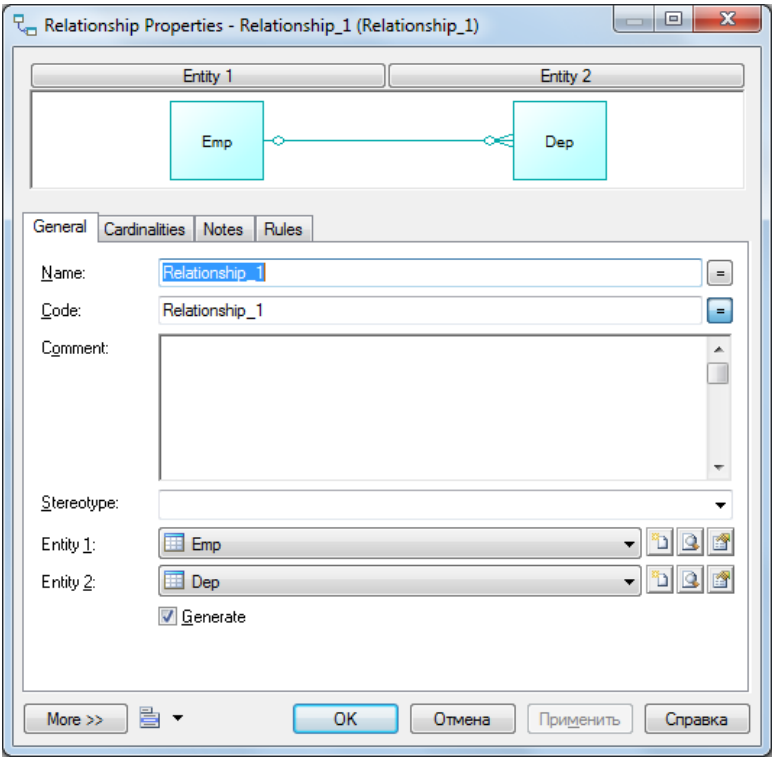
Создание связей

Нажмите кнопку **"Relationship"** на инструментальной панели, щелкните на одной из связываемых сущностей и перетащите указатель на другую сущность. Между данными сущностями появится линия связи.

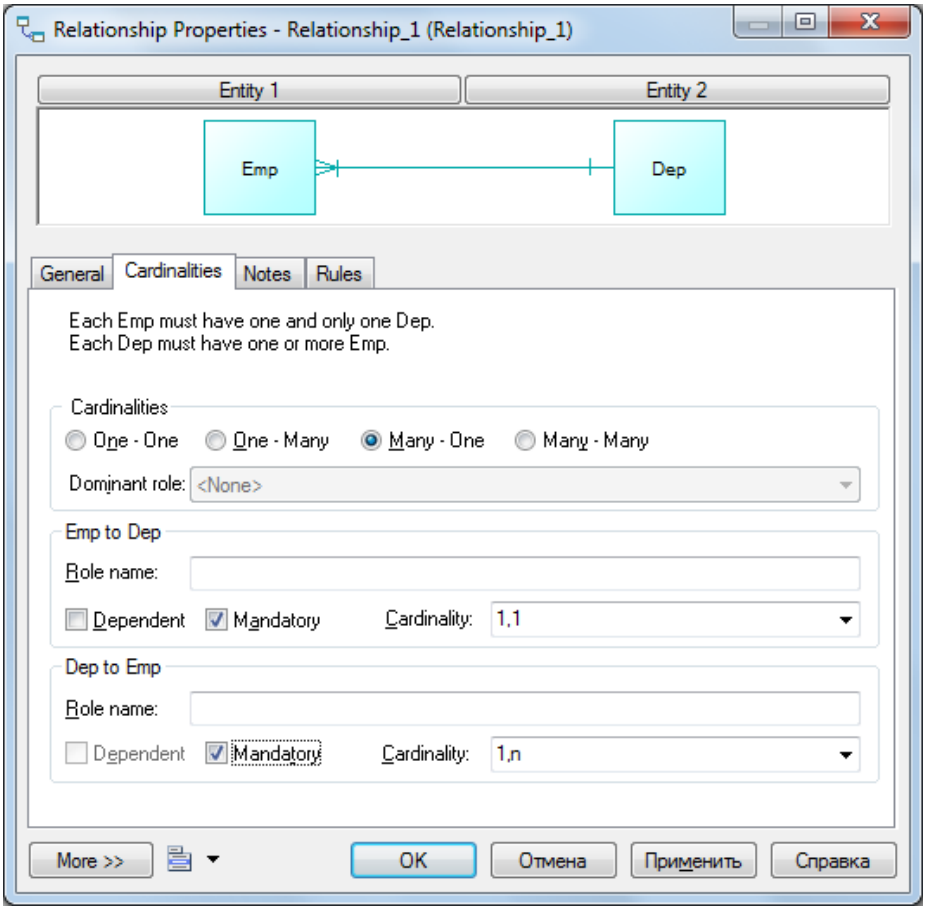
При создании связь называется Relation_n, где n – номер связи в порядке создания связей:



Дважды щелкните символ новой связи при выделенном указателе "Pointer" на инструментальной панели. Появится список свойств связи:

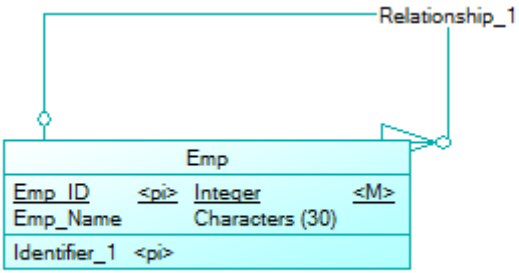


Наберите имя связи (Name), перейдите на вкладку **Cardinalities** и определите тип связи. Щелкните ОК:



Создание рекурсивной связи

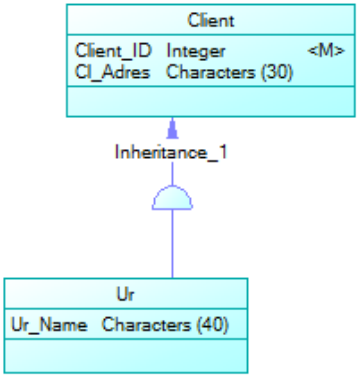
Чтобы создать рекурсивную связь, нажмите кнопку **"Relationship"** на инструментальной панели, перетащите указатель от сущности к ней самой, но несколько ниже, описав овал, и щелкните сущность. Линия связи замкнется на этой сущности:



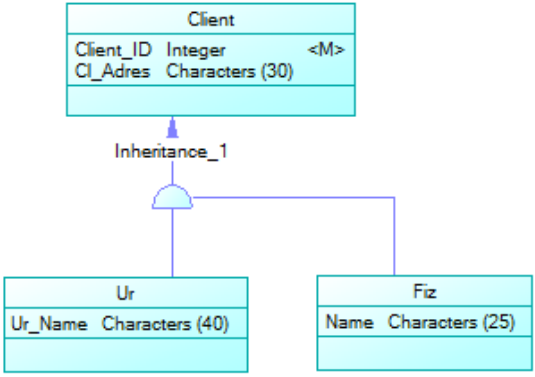
Определение связи «класс/подкласс».

Щелкните кнопку **"Inheritance"** на инструментальной панели:

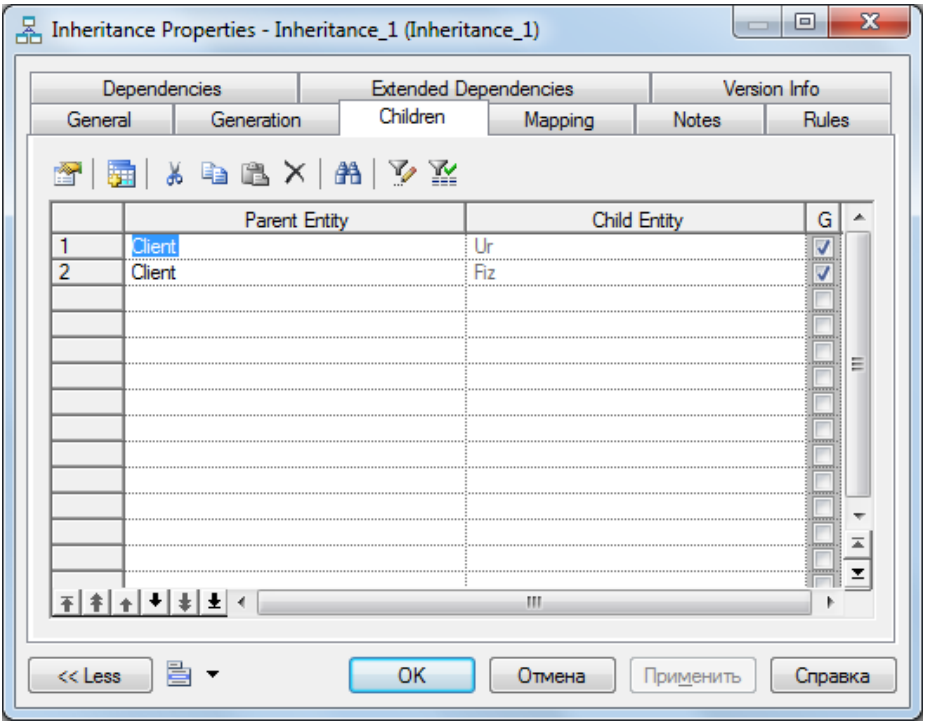
Перетащите указатель от сущности – потомка к сущности – родителю. Появится связь «класс/подкласс» между двумя сущностями с полукругом посередине и стрелкой, указывающей на родителя:



Чтобы создать дополнительных потомков для этой связи, щелкните кнопку **"Inheritance"** на инструментальной панели, перетащите указатель от полукруга связи к дополнительному потомку. Символ полукруга свяжет всех потомков с родителем:



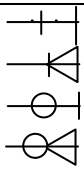
Щелкните правой кнопкой мыши на полукруге при выделенном указателе "Pointer" на инструментальной панели, посмотрите свойства связи "Properties", например, установленные отношения «родитель-потомок»:



Определение обязательных связей

Свойство «Обязательный» определяет, является ли связь между сущностями обязательной или необязательной (то есть участие сущности в связи полное или частичное). Это свойство определяется с точки зрения обеих сущностей в связи.

В CDM-модели вертикальная полоска, пересекающая линию связи, представляет обязательную связь, а круг – необязательную связь.

Конечная точка линии связи	Свойство связи	Кардинальность связи	Описание
	Обязательная Обязательная Необязательная Необязательная	Один Много Один Много	Должен быть один и только один Должен быть один или несколько Может быть один или ни одного Может быть один или несколько, или ни одного

Чтобы определить обязательную связь:

1. Дважды щелкните связь при выделенном указателе "Pointer" на инструментальной панели. Появится окно свойств для выделенной связи.
2. На вкладке **Cardinalities** выберите указатель "Mandatory" в одной или обеих группах, соответствующих направлению связи.
3. Щелкните OK.

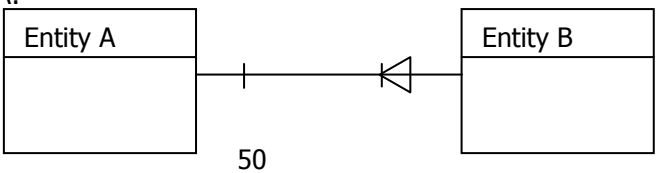
Определение подчиненной связи

В подчиненной связи одна сущность частично определена другой сущностью. Каждая сущность должна иметь идентификатор. В некоторых случаях, однако, атрибуты сущности не достаточны для определения экземпляра сущности. В этих случаях их идентификаторы включают идентификатор другой сущности, с которой они имеют подчиненную связь.

Например, хотя работа может быть выполнена во многих различных проектах, стоимость работы будет разной в каждом проекте. Поэтому уникальный идентификатор сущности «Работа» - это объединение ее атрибута «Название работы» и идентификатора «Номер проекта» из сущности «Проект».

!!! Связь «многие – ко - многим» не может быть подчиненной связью.

В CDM-модели треугольник перед концом линии связи определяет подчиненную связь. Треугольник указывается около подчиненной сущности. Например, на рисунке сущность B зависит от сущности A:



Кроме того, вертикальная полоска, пересекающая линию связи, определяет, что связь не только подчиненная, но и обязательная. Единственная точка в конце линии связи указывает, что кардинальность связи – «один – к – одному». В этом случае каждому экземпляру сущности B должен соответствовать один и только один экземпляр сущности A.

Вертикальная полоска у вершины треугольника служит для указания обязательной связи: каждый экземпляр сущности A требует экземпляр сущности B.

Для определения подчиненной связи:

1. Дважды щелкните связь при выделенном указателе "Pointer" на инструментальной панели. Появится окно свойств для выделенной связи.
2. Выберите или очистите указатель "Dependent" в группе, соответствующей направлению связи.
3. Щелкните OK.

Определение доминирующей связи

В связи «один – к – одному» можно определить одно направление связи как доминирующее (главное).

При определении главного направления связь «один – к – одному» генерирует одну ссылку в PDM-модели. Главная сущность становится родительской таблицей. Если главное направление не определено, связь «один – к – одному» генерирует две ссылки.

Чтобы определить главную связь:

1. Дважды щелкните связь при выделенном указателе "Pointer" на инструментальной панели. Появится окно свойств для выделенной связи.
2. Установите указатель "One to One".
3. Установите "Dominant" в группе, соответствующей направлению связи.
4. Щелкните OK.

Модификация связей

Можно перенести линию связи к другой сущности. Для этого:

1. Щелкните связь.
2. Нажмите CTRL и перенесите один конец линии связи к другой сущности.

Преобразование связи в ассоциированную сущность

Связь, соединяющая две сущности, не имеет атрибутов. Чтобы приписать атрибуты связи, надо преобразовать её в ассоциированную сущность, связанную двумя связями. Ассоциирован-

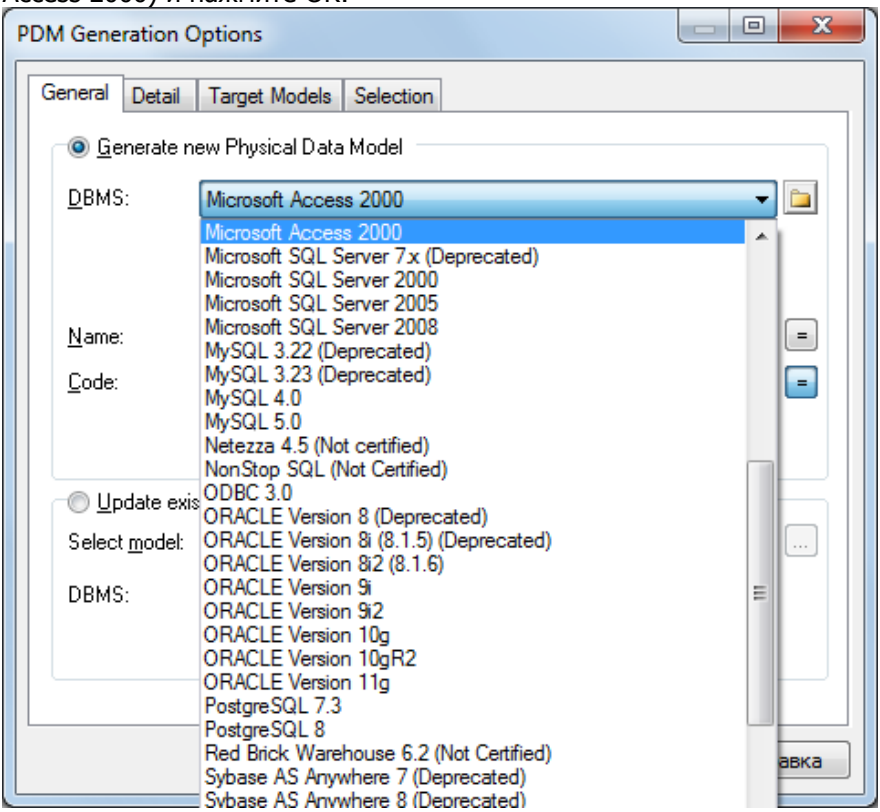
ная сущность получает имя и код соответствующей связи.

Чтобы преобразовать связь в ассоциированную сущность:

1. Щелкните правой кнопкой линию связи. Появится контекстное меню связи.
2. Выберите "**Change to Entity**" из контекстного меню. Ассоциированная сущность с двумя связями заменит данную связь. Ассоциированная сущность получит имя исходной связи.

Построение реляционной модели в Power Designer (PDM-модели)

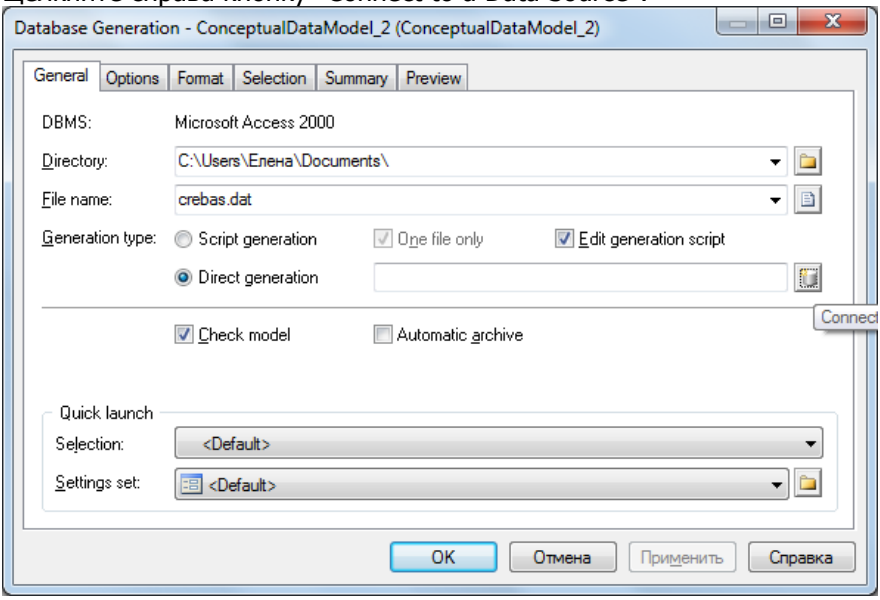
Выбрать пункт меню **"Tools"**, **"Generate Physical Data Model"**. В открывшемся окне выберите нужную СУБД (Microsoft Access 2000) и нажмите ОК:



Если нет ошибок в концептуальной модели, будет создана реляционная модель (PDM-модель) из данной концептуальной модели (CDM-модели).

СОЗДАНИЕ СХЕМЫ БАЗЫ ДАННЫХ MICROSOFT ACCESS С ПОМОЩЬЮ POWER DESIGNER

Чтобы сгенерировать БД MS Access с помощью Power Designer выберите в меню **Database → Generate Database**. В открывшемся окне задайте имя файла (File Name) и директорий (Directory) для файла скрипта. (Файл скрипта содержит описание структуры базы данных). Выберите режим "Direct generation" и щелкните справа кнопку "Connect to a Data Source":



В открывшемся окне нажмите кнопку "Config". После этого добавьте новый источник данных, выбрав нужный драйвер (MS Access Driver). Задайте имя источника данных. Нажмите «Создать». Откроется окно «Создание БД». Выберите каталог, где будет размещен файл базы данных. Задайте имя базы данных. Нажмите ОК. Будет создан файл вашей БД. Нажмите ОК. Вернувшись в окно "Connect to a Data Source", выберите в строке "Connection profile" созданный Вами источник данных и нажмите кнопку "Connect" (связать).

Будет сгенерирован скрипт на языке SQL для создания базы данных. Запустите этот скрипт, нажав кнопку "Run". Затем откройте файл базы данных и проверьте наличие таблиц и связей между таблицами.

Задание на лабораторную работу

1. Изучите вышеизложенные правила построения базы данных с помощью Power Designer.
2. Выполните с помощью Power Designer создание концептуальной и реляционной модели, а также схемы базы данных, описывающей предметную область для вашего задания из лабораторной работы № 1.
3. Оформите отчет по лабораторной работе.

Требования к отчету

Отчет должен содержать:

1. Цель работы.
2. Описание последовательности действий.
3. Схемы концептуальной и реляционной моделей данных, созданные в Power Designer.
4. Полученную с помощью Power Designer схему данных в Access.

Контрольные вопросы

1. Какую модель данных определяет CDM-модель?
2. Как построить связь, которой надо приписать атрибуты?
3. Какую модель данных определяет PDM-модель?
4. Как создать PDM-модель в Power Designer?
5. Можно ли с помощью Power Designer получить готовую схему данных в БД Access? Если да, то как это сделать?

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ НА ТЕМУ: «СОЗДАНИЕ ОДНОТАБЛИЧНЫХ ЗАПРОСОВ С ПОМОЩЬЮ МАСТЕРА И В РЕЖИМЕ КОНСТРУКТОРА»

ЦЕЛИ РАБОТЫ**:**

– освоить процедуру изменения количества отображаемых столбцов таблицы и фиксирования столбцов, сортировки данных;

– научиться использовать фильтры;

– научиться создавать запросы на выборку данных с помощью мастера;

– научиться создавать однотабличные запросы в окне конструктора.

Ход работы

Изменение количества отображаемых столбцов

ЗАДАНИЕ 1**.** ИЗМЕНИТЕ КОЛИЧЕСТВО ОТОБРАЖАЕМЫХ СТОЛБЦОВ В ТАБЛИЦЕ «КЛИЕНТЫ» УЧЕБНОЙ БАЗЫ ДАННЫХ БОРЕЙ, ЗАТЕМ ОТМЕНИТЕ ИЗМЕНЕНИЕ**.**

ПОРЯДОК РАБОТЫ

Скрыть отдельные столбцы таблицы, оставив на экране самые нужные для вас, позволяет команда **Скрыть столбцы**.

Чтобы скрыть столбец, выберите его нажатием на область маркировки этого столбца. Затем выполните команду **Скрыть столбцы** из меню **Формат** или контекстного меню.

Для отображения на экране скрытых столбцов используется команда **Формат/Показать столбцы**. При этом на экране открывается окно диалога «Отображение столбцов» со списком всех полей таблицы. Видимые в данный момент столбцы отмечены галочкой.

Чтобы сделать столбец видимым, установите соответствующий ему флажок. Если хотите скрыть какие-то столбцы, в этом же окне снимите флажок отображения поля.

Фиксирование столбцов

Задание 2. Выполните фиксирование и освобождение столбцов таблицы «Клиенты» учебной базы данных

Борей.

ПОРЯДОК РАБОТЫ

Допустим, при просмотре таблицы «Клиенты» желательно, чтобы на экране всегда отображался столбец с фамилиями клиентов.

Для этого служит команда **Формат/Закрепить столбцы**. Выделите закрепляемый столбец и выполните данную команду.

Если больше не нужно видеть этот столбец постоянно, выберите команду **Формат/Освободить все стобцы**.

Сортировка данных

Задание 3. Упорядочите в таблице «Сотрудники» базы данных Борей фамилии сотрудников по алфавиту.

ПОРЯДОК РАБОТЫ

После выбора поля для сортировки, выделите столбец и выполните команду **Записи/Сортировка**, а затем выберите опцию По возрастанию или По убыванию.

Более простой способ запустить сортировку – нажать соответствующую кнопку на панели инструментов.

Задание 4. Упорядочите таблицу более, чем по одному полю. Отсортируйте таблицу «Сотрудники» базы данных Борей по двум полям. Сначала отсортируйте должности сотрудников. Затем в каждой группе людей с одной и той же должностью выполните сортировку по фамилии.

ПОРЯДОК РАБОТЫ

1. Расположите поле, которое надо отсортировать первым (поле «Должность»), левее поля, которое отсортируем вторым (поле «Фамилия»). Чтобы это сделать, надо снять фиксацию со всех полей. Для этого выполните команду **Формат/Освободить все столбцы**.
2. Переместите поле (столбец) «Должность» левее поля «Фамилия».
3. Поле «Должность» оставьте выделенным. Нужно, чтобы оба сортируемых поля были выделены одновременно. Поэтому, держа нажатой клавишу Shift, нажмите мышью на область выбора поля «Фамилия». В результате выделены оба поля.
4. Нажмите кнопку сортировки по возраста-

нию на панели инструментов.

5. Закройте таблицу, не сохраняя изменений.

Использование фильтров

Фильтр используется для того, чтобы видеть только записи, определяемые заданным критерием. Создать фильтр можно в любом режиме работы с таблицей, используя кнопки панели инструментов и команды меню.

Кнопки и команды фильтрации перечислены в таблице 1.

Таблица 1.

Кнопка	Команда	Назначение
Фильтр по выделенному	Записи/ Фильтр/ Фильтр по выделенному	Выполняет фильтрацию данных по выделенному значению данного столбца
Изменить фильтр	Записи/ Фильтр/ Изменить фильтр Записи/ Фильтр/ Расширенный фильтр	Открывает окно диалога для задания критерия фильтрации
Применить фильтр	Применить фильтр/ Удалить фильтр	Переключает состояния применения и снятия фильтра

Итак, команда **Записи/Фильтр** содержит 3 вида фильтров: **Изменить фильтр**, **Фильтр по выделенному** и **Расширенный фильтр**. Рассмотрим их применение.

Фильтр по выделенному

Это самый простой фильтр.

Задание 5. Откройте таблицу «Поставщики» базы данных Борей. Выберите всех поставщиков товаров из Лондона.

ПОРЯДОК РАБОТЫ

Выберите слово **Лондон** в поле **Город** таблицы «Поставщики» и нажмите кнопку **Фильтр по выделенному** на панели инструментов.

В таблице 2 приведены различные виды выборов и показано, как каждый из них влияет на результат фильтрации.

Таблица 2.

Что выбирается	Где выбирается	Что возвращается

Целое значение	Все поле	Только точное совпадение
Часть значения	Начало поля	Только записи, начинающиеся с выбранных символов
Часть значения	После первого символа	Любая запись, содержащая эти символы

Исключение записей из фильтра

Можно фильтровать записи с помощью исключения данных вместо их включения. Другими словами, возвращены будут только те записи, которые не были выбраны.

Задание 6. Выберите из таблицы «Поставщики» всех поставщиков, кроме поставщиков из Лондона.

ПОРЯДОК РАБОТЫ

1. Откройте таблицу «Поставщики» учебной базы данных Борей.
2. Выберите значения, которые будут критерием.
3. Нажмите на выбранное место правой кнопкой мыши. Появится контекстное меню.
4. Выберите из этого меню команду **Исключить выделенное**.

Команда «Изменить фильтр»

Это более мощный фильтр, чем фильтр по выделенному. В этом фильтре можно устанавливать критерии **И** и **ИЛИ**, а также использовать выражения.

Задание 7. Выберите всех поставщиков из Москвы, в названии которых есть символы «ООО».

ПОРЯДОК РАБОТЫ

В ЭТОМ ЗАПРОСЕ ИСПОЛЬЗУЕТСЯ КРИТЕРИЙ **И**.

1. Откройте таблицу «Поставщики» базы данных Борей.
2. Нажмите кнопку **Изменить фильтр** на панели инструментов. На экране откроется окно диалога, в верхней части которого все поля таблицы «Поставщики», в нижней части – ярлычки «Найти» и «Или».
3. Установите курсор в поле **Город**. Нажмите кнопку раскрытия списка и выберите из него значение **Москва**.
4. В поле ввода **Название** введите значение «**Like**

"*ООО*"».

5. Нажмите кнопку **Применить фильтр** на панели инструментов.

Совместное использование критериев И и ИЛИ.

Задание 8. Выберите всех поставщиков из Москвы, в названии которых есть символы «ООО», и поставщиков из Лондона.

ПОРЯДОК РАБОТЫ

В ЭТОМ ЗАПРОСЕ ИСПОЛЬЗУЮТСЯ КРИТЕРИИ **И** И **ИЛИ**. ПРИ ИСПОЛЬЗОВАНИИ КРИТЕРИЯ **ИЛИ** НАДО, ЧТОБЫ ВЫПОЛНЯЛОСЬ ХОТЯ БЫ ОДНО ИЗ УСЛОВИЙ СРАВНЕНИЯ.

1. Откройте таблицу «Поставщики» базы данных Борей.
2. Нажмите кнопку **Изменить фильтр** на панели инструментов. В открывшемся окне установите курсор в поле **Город** и выберите из списка значение **Москва**.
3. В поле ввода **Название** введите значение «**Like "*ООО*"**».
4. Нажмите ярлычок **ИЛИ**. Откроется новая страница запроса.
5. В открывшемся окне установите курсор в поле **Город** и выберите из списка или введите значение **Лондон**.
6. Нажмите кнопку **Применить фильтр**.

Расширенный фильтр

Для использования расширенного фильтра выполняется команда **Записи/Фильтр/Расширенный фильтр**. При этом откроется окно диалога, в верхней части которого расположен список полей таблицы, а в нижней – бланк формирования выражения для фильтра. По сравнению с предыдущим фильтром данный фильтр позволяет проводить не только фильтрацию, но и сортировку по нескольким полям одновременно. Для этого используется строка **Сортировка**.

Задание 9. Выберите всех поставщиков из Москвы, в названии которых есть символы «ООО», и поставщиков из Канады. Отсортируйте результат по убыванию значения поля Название.

ПОРЯДОК РАБОТЫ

1. Откройте таблицу «Поставщики» и выберите команду **Записи/Фильтр/Расширенный фильтр**.
2. Выберите нужные поля из таблицы «Поставщики» и перенесите их в нижний бланк. В строке «Условие отбора» задайте нужные значения для полей **Название**, **Город**, **Страна**. Причем для каждой группы значений полей, соединенных связкой **ИЛИ**, условие отбора задайте на отдельной строке.
3. Задайте порядок сортировки в строке «Сортировка» нужного столбца.
4. Нажмите кнопку **Применить фильтр**.

Создание запросов с помощью мастера

Для создания запроса в окне базы данных перейдите на вкладку «Запросы» и нажмите кнопку **Создать**. Откроется окно диалога, в котором можно выбрать опцию создания запроса с помощью конструктора запросов или один из мастеров создания запроса. При выборе опции **Простой запрос** вам будет предложено указать список таблиц и их полей, на основании которых будет создан требуемый запрос.

Задание 10. Определить стоимость расходов на доставку по клиентам и типам доставки.

ПОРЯДОК РАБОТЫ

1. В окне базы данных перейдите на вкладку «Запросы» и нажмите кнопку **Создать**. В открывшемся окне выберите опцию **Простой запрос** и нажмите ОК.
2. Из таблицы «Клиенты» выберите поле **Название**, из таблицы «Заказы» - поле **Стоимость доставки**, а из таблицы «Доставка» - наименование типа доставки (поле **Название**). Нажмите кнопку **Далее**.
3. На экране появится следующее окно диалога, в котором надо выбрать признак подробного или итогового запроса. В данном случае нас интересует итоговое значение стоимости доставки, поэтому выберите опцию **Итоговый**.
4. Нажав кнопку **Итоги**, определите, какие итоговые значения требуется получить (Sum). Нажмите ОК для возврата в предыдущее окно мастера, в котором нажмите кнопку **Далее**. Вы перейдете в завершающее окно мастера.
5. Задайте имя запроса и выберите опцию **Открытие результатов выполнения запроса**.

Создание однотабличных запросов в окне конструктора запросов

Для вызова конструктора запросов перейдите в окно базы данных на вкладку «Запросы» и нажмите кнопку **Создать**. В окне диалога «Новый запрос» выберите опцию **Конструктор** и нажмите ОК. Вам предложат выбрать таблицу. Выберите таблицу, нажав кнопку **Добавить** и закройте окно диалога. На экране появится окно конструктора запросов, а в основном меню – команда **Запрос**.

Меню **Запрос** содержит команды, многие из которых дублируются кнопками на панели инструментов. Можно также использовать контекстное меню. В таблице 3 дается описание команд меню **Запрос**.

Таблица 3.

Команда меню	Описание
Запуск	Выполнить запрос
Добавить таблицу	Добавляет новую таблицу в запрос
Выбор типа запроса	Выбирает тип запроса: Выборка, Создание таблицы, Перекрестный, Обновление, Добавление, Удаление
Запрос SQL	Создает запрос SQL на объединение, запрос к серверу или управляющий запрос
Параметры	Задает параметры запроса

В окне конструктора запросов меню **Вид** содержит команды управления запросом, которые представлены в таблице 4.

Таблица 4.

Команда меню	Описание
Режимы работы	Переключает режимы работы (Конструктор запросов, Режим SQL, Режим таблицы)
Групповые операции	Добавляет в бланк запроса строку **Групповая операция**
Имена таблиц	Добавляет в бланк запроса строку **Имя таблицы**
Свойства	Открывает окно свойств запроса

Выполнение простого запроса

Задание 11. Получить названия фирм, имена представителей и номера телефонов из таблицы «Клиенты» базы данных Борей.

ПОРЯДОК РАБОТЫ

1. В окне базы данных перейдите на вкладку «Запросы» и нажмите кнопку **Создать**, выберите опцию **Конструктор** и нажмите ОК.
2. В окне «Добавление таблицы» выберите таблицу «Клиенты» и нажмите кнопку **Добавить**. Закройте окно.
3. На экране откроется окно конструктора запросов с одной таблицей «Клиенты» и пустым бланком запросов.
4. Добавьте поля в бланк запроса. Добавить поля можно одним из двух способов:
   1. выбрать поле таблицы на схеме данных и дважды нажать кнопку мыши.
   2. использовать механизм «перенести-и-оставить». Для этого выделить поля в таблице в схеме данных, нажать кнопку мыши и, не отпуская ее, перенести выбранные поля в бланк запроса. Для выделения группы полей можно использовать мышь совместно с клавишами Shift и Ctrl.

   Любым из этих способов перенесите в бланк запроса поля **Название**, **Обращаться к** и **Телефон**.
5. Выполните запрос, нажав кнопку **Запуск** на панели инструментов.
6. Можете сохранить этот запрос и использовать его в дальнейшем.

Выбор всех полей таблицы

Выбрать все поля таблицы можно двумя способами:

1. Дважды нажать мышью на строке заголовка таблицы и перенести выделенные поля в бланк запроса.
2. Нажать на звездочку (*) в списке полей таблицы и перенести ее в бланк запроса, удерживая кнопку мыши в нажатом состоянии. Имя поля в бланке запроса будет содержать имя таблицы, за которым следует точка и звездочка (например, **Клиенты.***), что означает, что были выбраны все поля таблицы.

Удаление полей из бланка запроса

Для удаления всех полей выберите команду **Правка/Очистить бланк**.

Для удаления данного поля нажмите на область выбора столбца, а затем клавишу **Delete**.

Изменение порядка полей в бланке запроса выполняется также, как в таблице.

Сохранение запроса выполняется командой **Файл/Сохранить как/Экспорт**, которая откроет окно диалога ввода имени запроса.

Использование простого критерия выборки записей

Задание 12. Получить названия фирм, имена представителей и номера телефонов из таблицы «Клиенты» базы данных Борей. Информация нужна только по клиентам из США.

ПОРЯДОК РАБОТЫ

1. Откройте предыдущий запрос.
2. Из таблицы «Клиенты» добавьте в бланк запроса поле **Страна**.
3. В строке «Условие отбора» для поля **Страна** введите «США».
4. Снимите флажок вывода на экран для поля **Страна**, так как это поле используется только для задания условия выборки.
5. Запустите запрос.

Использование в запросах вычисляемых полей

При выполнении запроса можно вычислять значения по одному или нескольким полям таблицы. Например, таблица «Заказано» содержит данные о количестве товаров (поле **Количество**), проданных в каждом заказе, и стоимость единицы изделия

(поле **Цена**). На основе этой информации можно вычислить стоимость проданных товаров по каждому заказу.

Задание 13. Выбрать из таблицы «Заказано» базы данных Борей для каждого заказа список заказанных товаров и стоимость каждой партии товаров.

ПОРЯДОК РАБОТЫ

1. Откройте окно конструктора запросов для таблицы «Заказано» и добавьте в бланк запроса поля **КодЗаказа** и **КодТовара**.
2. Для определения стоимости партии товаров перейдите на строку «Поле» пустого столбца бланка запроса и нажмите кнопку **Построить** на панели инструментов или выберите эту команду из контекстного меню. На экране откроется окно построителя выражений.
3. Для добавления в выражение поля **Цена** таблицы «Заказано» раскройте папку «Таблицы» и выберите из иерархического списка таблицу «Заказано». После этого список правее будет содержать перечень полей указанной таблицы. Выберите из этого списка поле **Цена** и дважды нажмите его мышью или нажмите кнопку **Вставить** окна построителя. Выбранное поле будет добавлено в область ввода выражения в виде [Заказано]![Цена]. Теперь добавьте знак умножения, а затем аналогично добавлению поля **Цена** добавьте в выражение поле **Количество**.
4. После завершения формирования выражения нажмите кнопку ОК, и выражение будет перенесено в строку «Поле» бланка запроса. Access автоматически задаст имя поля (например, «Выражение 1»), которое отделяется от выражения двоеточием. Измените это имя, отредактировав текст в ячейке строки «Поле», указав вместо «Выражение 1» слово «Стоимость».
5. Запустите запрос.

Построение сложных условий для выбора записей

Иногда требуется найти в таблице записи, значения которых не удовлетворяют определенному условию. Для установки таких условий используется оператор **Not**, который печатается перед сравниваемым значением.

Задание 14. Выбрать все записи из таблицы «Клиенты», за исключением записей со значением «США» в поле Страна.

ПОРЯДОК РАБОТЫ

1. Откройте запрос, созданный для выбора клиентов из США.
2. В строке «Условие отбора» поля **Страна** вставьте перед значением «США» оператор **Not** или **<>**.
3. Так как теперь в выборке участвуют клиенты из разных стран, установите флажок вывода на экран для поля **Страна**.
4. Для выполнения запроса нажмите кнопку **Запуск**.

Условие неточного совпадения

Можно выбрать записи по условию неточного совпадения значений. Данное условие позволяет найти требуемые записи, зная лишь приблизительное написание значения. В MS Access для этих целей используется оператор **Like**, который осуществляет сравнение значения поля с заданным образцом. Образец задается сразу же за оператором и может содержать точное значение (например, **Like "Иванов"**) или использовать символы шаблонов для поиска диапазона значений (например, **Like "Ив*"**). Таблица 5 содержит символы шаблонов, которые используются с оператором **Like**, и количество цифр или символов, которым они соответствуют.

Таблица 5.

Символы в образце	Соответствие в выражении
?	Любой один знак
*	Нуль или более знаков
#	Любая одна цифра
[список знаков]	Любой один знак в списке знаков
[!список знаков]	Любой один знак, не входящий в список знаков

Группа из одного или более символов, заключенная в квадратные скобки, используется для установления совпадения с одним символом выражения и может содержать любые символы.

Кроме простого списка символов, заключенного в квадратные скобки, «список знаков» позволяет задать диапазон символов. Можно указать знак (-) для отделения верхней границы диапазона от нижней. Например, использование шаблона [Г-Л] в образце позволяет выбрать все записи, которые в указанной позиции поля содержат одну из заглавных букв в диапазоне от Г до Л. Можно использовать даже несколько диапазонов внутри одной пары квадратных скобок без специальных разделителей. Напри-

мер, [г-лГ-Л] позволяет выбрать как заглавные, так и прописные буквы.

Восклицательный знак в начале «списка знаков» означает, что совпадение наступит, если в выражении будет найден любой символ, отсутствующий в «списке знаков». Восклицательный знак, использованный вне квадратных скобок, соответствует самому себе.

Задание 15. В таблице «Клиенты» найти запись о представителе фирмы, название которой начинается на Ric.

ПОРЯДОК РАБОТЫ

1. Откройте окно конструктора запросов для таблицы «Клиенты».
2. Перенесите поля **Название**, **Обращаться к**, **Телефон**, **Страна** в бланк запроса.
3. Для задания условия выберите поле **Название** и в строке «Условие отбора» поля напечатайте **Like "Ric*"**.
4. Задайте сортировку по возрастанию для поля **Название**.
5. Выполните запрос.

Задание 16. Выберите всех клиентов, наименования которых начинаются на A или C.

ПОРЯДОК РАБОТЫ

Откройте предыдущий запрос. В строке «Условие отбора» поля **Название** введите следующее условие: **Like "[AC]*"**.

Выбор записей по диапазону значений

Иногда необходимо выбрать из таблицы записи, данные которых попадают в диапазон значений.

Для задания диапазона значений в окне конструктора запросов используются операторы <, <=, >, >= и Between, которые можно использовать с текстовыми и цифровыми полями, а также полями дат.

Задание 17. Получить список товаров, чья цена превышает 10000.

ПОРЯДОК РАБОТЫ

1. Откройте окно конструктора запросов для таблицы «Товары».
2. Выделите поля, которые хотите отобразить

в запросе (поля **Название** и **Цена**), и перенесите их в бланк запроса.

3. Для задания условия отбора выберите поле **Цена** и в строке «Условие отбора» поля напечатайте: >10000.

4. Нажмите кнопку **Запуск**.

Задание 18. Получить из таблицы «Заказы» сведения обо всех заказах с 15 по 31 мая 1997г.

ПОРЯДОК РАБОТЫ

1. Откройте окно конструктора запросов для таблицы «Заказы», перенесите поля, которые хотите отобразить в запросе, в бланк запроса.

2. Для задания условия отбора из списка полей таблицы «Заказы» выберите поле **Дата исполнения**. Для ввода условия выборки перейдите на строку «Условие отбора» и нажмите кнопку **Построить** на панели инструментов. В списке операторов сравнения выберите **Between**. В области ввода выражения появится шаблон для ввода параметров данного оператора:

Between "Выражение" And "Выражение"

3. Вставьте вместо первого параметра начальную дату, а в качестве второго – конечную дату. В Access в условиях отбора значение даты выделяется с обеих сторон символом **#** :

Between #15.05.97# And #31.05.97#

Нажмите кнопку **Запуск**.

Задание 19. Получить список клиентов, наименования которых начинаются с C по G.

Это пример задания диапазона строковых данных.

Для выполнения данного запроса в строку «Условие отбора» поля **Название** введите условие отбора: **Like "[C-G]*"**.

Объединение критериев нескольких полей

Часто в критерии выборки задаются условия для нескольких полей таблицы или несколько условий для одного поля. Если запись выбирается только в случае выполнения всех условий, то запрос называется И-запросом. Если же запись выбирается при выполнении хотя бы одного из всех условий, то запрос называется ИЛИ-запросом.

Для задания И-выражения необходимо просто задать условие в строке «Условие отбора» для каждого из полей, образующих критерий.

При задания ИЛИ-выражения каждое из условий выбора, образующих критерий, должно располагаться на отдельной строке бланка запроса.

При вводе условия можно использовать операторы **Or** и **And**, которые позволяют формировать в одной строке сложное условие выборки.

При вводе условия можно формировать любое допустимое в Access логическое условие, которое может содержать функции, операторы сравнения, Or, And, Not и скобки для изменения порядка выполнения выражения.

Задание 20. Из таблицы «Заказы» выбрать информацию о заказах, выполненных с 15 по 31 мая 1997 г., код доставки которых равен 1.

ПОРЯДОК РАБОТЫ

1. Откройте запрос, в котором выбирались заказы в диапазоне дат.
2. Добавьте в бланк запроса поле **Доставка**.
3. Перейдите на строку «Условие отбора» поля и напечатайте 1.
4. Выполните запрос.

Задание 21. Из таблицы «Заказы» выбрать информацию о заказах, выполненных с 15 по 31 мая 1997 г., или с кодом доставки 1.

ПОРЯДОК РАБОТЫ

В отличие от предыдущего запроса напечатайте значение 1 не в строке «Условие отбора», а в строке «Или».

Задание на лабораторную работу

1. Изучите методические указания и выполните описанные в них задания, результаты выполнения заданий предъявите преподавателю на экране дисплея.
2. Заполните данными созданную вами в предыдущей лабораторной работе базу данных.
3. Создайте в конструкторе различные варианты однотабличных запросов к вашей базе данных (запросы с использованием логических критериев И, ИЛИ, НЕ; запросы с вычисляемыми полями; запросы на выбор по диапазону значений).
4. Оформите отчет по лабораторной работе.

Требования к отчету

Отчет должен содержать:

1. Цель работы.
2. Содержимое таблиц базы данных, используемых в запросах.
3. Формулировку запросов.
4. Описание процесса создания запросов.
5. Результаты выполнения запросов.

Контрольные вопросы

1. Как построить запрос с помощью мастера в БД Access?
2. Опишите последовательность действий при создании запроса в режиме конструктора.
3. Какой оператор используется В MS Access для сравнения значения поля с заданным образцом?
4. Какие операторы используются для задания диапазона значений в окне конструктора?
5. Какие логические операторы можно использовать для формирования сложных условий запроса?

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ НА ТЕМУ: «СОЗДАНИЕ МНОГОТАБЛИЧНЫХ ЗАПРОСОВ В КОНСТРУКТОРЕ»

ЦЕЛЬ РАБОТЫ:

– научиться создавать многотабличные запросы в окне конструктора;

– научиться создавать перекрестные запросы в окне конструктора.

Ход работы

МНОГОТАБЛИЧНЫЕ ЗАПРОСЫ

Для формирования многотабличного запроса необходимо добавить в окно конструктора запросов все таблицы, участвующие в выборке, и определить условия их объединения. Для добавления таблицы выберите команду **Запрос/Добавить таблицу** или нажмите кнопку **Добавить таблицу** на панели инструментов. Если в БД установлены отношения между таблицами, участвующими в запросе, то эта связь будет отображаться в виде линии, соединяющей таблицы. В этом случае не придется устанавливать связь между таблицами в конструкторе запросов. Если же между таблицами не существует связи, то можно установить требуемую связь, используя механизм «перенести – и - оставить». Для этого выберите поле в одной из таблиц и перенесите его на связываемое поле в другой таблице.

Объединение двух таблиц

Задание 1. Откройте учебную базу данных Борей. Выполните следующий запрос: определить тип доставки товаров клиентам.

ПОРЯДОК РАБОТЫ

Для выполнения запроса используется информация из двух связанных таблиц Заказы и Заказано. В БД между этими таблицами уже определены постоянные отношения. Для решения поставленной задачи выполните следующее:

1. Добавьте таблицу Заказы в новое окно

конструктора запросов.

2. Добавьте таблицу Заказано.

3. Перенесите в бланк запроса из таблицы Заказано поле Код товара, а из таблицы Заказы – поля Код заказа и Доставка. Установите сортировку по возрастанию для полей Код товара и Доставка.

4. Выполните запрос.

Условие отбора в многотабличных запросах

Задание 2. Выбрать список заказов, в которых товар с кодом 2 доставляется по почте.

ПОРЯДОК РАБОТЫ

Возьмите за основу запрос, созданный в предыдущем примере, и выполните следующее:

1. Для создания условия отбора товара перейдите на строку Условие отбора поля Код Товара и введите код товара. Для этого поля снимите флажок Вывод на экран, так как не имеет смысла выводить столбец, каждая строка которого будет содержать одно и то же значение.

2. Перейдите на Условие отбора поля Доставка и задайте код доставки почтой, равный значению 2, а затем снимите флажок Вывод на экран.

3. Добавьте в бланк запроса поля, содержащие сведения о сотруднике и дату исполнения заказа.

4. Выполните запрос.

Объединение нескольких таблиц

Задание 3. Выбрать товары указанного типа (напр., рыбопродукты), проданные клиентам, находящимся в указанной стране (напр., Франция).

ПОРЯДОК РАБОТЫ

Для решения задачи потребуются таблицы Клиенты, Заказы, Заказано и Товары. В таблице Клиенты содержится информация о местонахождении клиентов, а в таблице Товары – код типа товара. Таблицы Заказы и Заказано позволяют связать информацию о покупателе с проданным ему товаром. Выполните следующие действия:

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Клиенты, Заказы, Заказано и Товары.

2. Добавьте в бланк запроса поля, содержание название клиента и марку товара, а также установите для них флажок Вывод на экран.

3. Для задания условия отбора клиентов по стране местонахождения добавьте в бланк запроса поле Страна таблицы Клиенты. Перейдите на строку Условие отбора этого поля и введите название страны (Франция). Так как выводит этот столбец не имеет смысла, снимите флажок Вывод на экран.

4. Добавьте поле Код типа таблицы Товары и в строке Условие отбора задайте код типа рыбопродуктов, равный 8, а затем снимите флажок Вывод на экран.

5. Выполните запрос.

Итоговые запросы

Часто бывает недостаточно имеющихся в таблицах данных. Например, нужно найти сумму, максимальную величину в поле или количество записей, содержащих определенную величину. В этом случае необходимо выполнить итоговые запросы.

Для создания итогового запроса выберите **Вид/ Групповые операции** или нажмите кнопку **Групповые операции** на панели инструментов. В бланке запроса появится новая строка с наименованием **Групповая операция**. В этой строке надо указать тип выполняемого вычисления.

Таблица 1 содержит перечень всех допустимых видов итоговых операций, которые можно выбрать из раскрывающегося списка в строке **Групповая операция**.

Таблица 1.

Значение	Выполняемая операция
Sum	Сложение
Avg	Среднее значение
Min	Минимальное значение
Max	Максимальное значение
Count	Количество записей, содержащих значение
StDev	Стандартное отклонение
Var	Дисперсия
First	Значение в первой записи
Last	Значение в последней записи

Операция Count действует следующим образом: просматриваются все записи для указанного поля и вычисляется количество записей, которые содержат в этом поле какое-либо значение. Данная операция пропускает пустые значения.

Для удаления строки **Групповая операция** достаточно нажать еще раз на кнопку **Групповая операция**.

Определение суммы всех значений по полю

Задание 4. Определить итоговую стоимость всех перевозок товаров.

ПОРЯДОК РАБОТЫ

Стоимость перевозки указана в поле Стоимость доставки таблицы Заказы. Для создания запроса, вычисляющего их общую сумму, выполните следующие действия:

1. Откройте новое окно конструктора запросов и добавьте в него таблицу Заказы.
2. Добавьте в бланк запроса поле Стоимость доставки.
3. Выберите команду Вид/ Групповые операции для добавления в бланк запроса строки Групповая операция.
4. Нажмите кнопку мыши в первом столбце строки Групповая операция и из раскрывающегося списка выберите значение **Sum**, которое указывает MS Access просуммировать все значения поля Стоимость доставки.
5. Нажмите кнопку Запуск.

Определение максимального и среднего значения по полю

Задание 5. Определить не только итоговую стоимость доставки, но также максимальное и среднее значение стоимости доставки.

ПОРЯДОК РАБОТЫ

Для решения задачи вернитесь в окно конструктора запросов предыдущего запроса и добавьте в бланк запроса поле Стоимость доставки еще 2 раза. Для второго поля Стоимость доставки из раскрывающегося списка в строке Групповая операция выберите **Max**, а для третьего – **Avg**. Выполните запрос.

Определение условия выборки в итоговых запросах

Задание 6. Определить стоимость доставки товаров по почте.

ПОРЯДОК РАБОТЫ

1. В бланк запроса, созданного в предыдущем примере, добавьте поле Доставка. Сразу после добавления этого поля в строке Групповая операция устанавливается значение **Группировка**.
2. Для ввода условия выборки измените это значение на **Условие**. При выборе этого значения автоматически снимается флажок **Вывод на экран**, и поле не выводится на экран при выполнении запроса.
3. После этого введите в строке Условие отбора требуемое условие. Для определения итоговой стоимости доставки по почте напечатайте код доставки, равный 2.
4. Для просмотра результатов выборки нажмите кнопку Запуск.

Группировка полей запроса

Группировка позволяет получить вычисляемую информацию о подгруппах таблицы.

Задание 7. Получить сведения об итоговом количестве заказов каждого вида товара.

ПОРЯДОК РАБОТЫ

1. Добавьте таблицу Заказано в окно конструктора запросов.
2. В первый столбец бланка запроса поместите поле Код Товара.
3. Во второй столбец поместите Код Заказа.
4. Выберите команду **Вид/Групповые операции** для добавления в бланк запроса строки Групповая операция.
5. В строке Групповая операция столбца Код Товара установите значение **Группировка**, а для столбца Код Заказа – значение **Count**.
6. Количество заказов товара представляет интерес, но также интересно знать количество проданного товара. Один товар может продаваться очень часто, но маленькими партиями, а другой товар реже, но крупными партиями. Поэтому добавьте в запрос поле Количество и в строке Групповая операция установите значение **Sum**.

7. Установите сортировку по убыванию для поля Код Заказа.
8. Выполните запрос.

Поле, используемое для группировки, не обязательно должно находиться в той же таблице, что и итоговое поле. Напр., вместо товаров можно использовать для группировки типы товаров, а поле с этими данными находится в таблице Товары.

Задание 8. Создайте указанный запрос.

ПОРЯДОК РАБОТЫ

1. Откройте окно конструктора запросов для предыдущего запроса.
2. Добавьте в него таблицу Товары.
3. Удалите из бланка запроса поле Код Товара.
4. Добавьте в бланк запроса поле Код Типа из таблицы Товары и в строке Групповая операция установите значение **Группировка**.
5. Выполните запрос.

Группировка по нескольким полям

Можно группировать данные не только по одному полю, но и по нескольким полям одновременно, а также создавать группы внутри групп. Рассмотрим эти возможности на примере.

В БД Борей существует 3 типа доставки товаров: РосТранс, Почта, Иное.

Задание 9. Определить количество заказов по каждому покупателю и типу доставки.

ПОРЯДОК РАБОТЫ

Можно сгруппировать записи сначала по покупателям, затем – по типам доставки. Порядок полей группировки, размещенных в бланке запроса, определяет порядок группировки выбираемых данных. Для создания запроса выполните следующие действия:

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Клиенты, Заказы и Доставка.
2. В бланк запроса поместите поля Название из таблиц Клиенты и Доставка, а также поле Код Заказа из таблицы Заказы.
3. Нажмите кнопку Групповые операции на панели инструментов.
4. В строке Групповая операция первых двух столб-

цов установите значение **Группировка**, а для столбца Код Заказа – значение **Count**.

5. Выполните запрос.

Включение в запрос выражений

MS Access позволяет выполнять итоговые операции над вычисляемыми полями выборки. Напр., таблица Заказы содержит данные о клиентах, а таблица Заказано – о количестве товаров, проданных в каждой партии, и цене товара. На основе этой информации можно вычислить итоговую стоимость заказа каждым покупателем по каждому товару.

Чтобы включить в итоговый запрос выражение, надо добавить в запрос вычисляемое поле и указать тип итоговых вычислений, осуществляемых над этим полем.

Выражения для вычисления могут содержать формулы, связанные арифметическими операторами. В качестве элементов формулы могут использоваться поля, константы и функции. Для изменения порядка вычислений в выражениях используются круглые скобки.

Задание 10. Определить итоговую стоимость каждого проданного товара отдельно для каждого покупателя.

ПОРЯДОК РАБОТЫ

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Заказы и Заказано.
2. В бланк запроса поместите поля Код Клиента из таблицы Заказы и Код Товара из таблицы Заказано.
3. В бланк запроса добавьте вычисляемое поле и введите для него выражение для вычисления стоимости партии товаров:

Стоимость: [Заказано] ! [Количество] * [Заказано] ! [Цена]

4. Нажмите кнопку Групповые операции на панели инструментов для добавления в бланк запроса строки Групповая операция.
5. В строке Групповая операция первых двух столбцов установите значение **Группировка**, а для столбца, содержащего выражение, – значение **Sum**.
6. Выполните запрос. Вы должны получить список клиентов и итоговую стоимость заказанных ими товаров.

Использование более сложных условий отбора записей в итоговых запросах

Задание 11. Найти клиентов, у которых для каждого типа доставки было оформлено более трех заказов.

ПОРЯДОК РАБОТЫ

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Клиенты, Заказы и Доставка.
2. В бланк запроса поместите поля Название из таблиц Клиенты и Доставка, а также поле Код Заказа из таблицы Заказы.
3. Нажмите кнопку Групповые операции на панели инструментов для добавления в бланк запроса строки Групповая операция.
4. В строке Групповая операция первых двух столбцов установите значение **Группировка**, а для столбца, Код Заказа - значение **Count**.
7. Перейдите на строку Условие отбора столбца Код Заказа и напечатайте: >3.
8. Выполните запрос. Из списка клиентов и соответствующих им типов доставки должны быть выбраны только те записи, в которых количество заказов превышает 3.

Задание на лабораторную работу

1. Изучите методические указания и выполните описанные в них задания, результаты выполнения заданий предъявите преподавателю на экране дисплея.
2. Используя справку в Access, изучите назначение и способы создания перекрестных запросов. Постройте в конструкторе следующий перекрестный запрос к базе данных Борей: по каждому клиенту (коду клиента) и каждому типу доставки (название типа) посчитать число заказов (кодов заказа), дата исполнения которых находится в диапазоне с 1.01.1997 по 1.02.1997. В качестве заголовков строк выберите коды клиентов, в качестве заголовков столбцов – название типа доставки. Используйте для образца имеющийся в базе данных Борей запрос «Квартальные отчеты» или «Квартальные обороты по товарам».
3. Создайте в конструкторе различные варианты многотабличных запросов к вашей базе данных (созданной по заданию лабораторной работы № 1), подобные запросам, описанным в методических указаниях.
4. Оформите отчет по лабораторной работе.

Требования к отчету

Отчет должен содержать:

1. Цель работы.
2. Содержимое таблиц базы данных, используемых в запросах.
3. Формулировку запросов.
4. Описание процесса создания запросов.
5. Результаты выполнения запросов.

Контрольные вопросы

1. Опишите последовательность действия по созданию многотабличного запроса в конструкторе.
2. Опишите действия по созданию итогового запроса.
3. Перечислите допустимые виды итоговых операций.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ НА ТЕМУ: «СОЗДАНИЕ МНОГОТАБЛИЧНЫХ ЗАПРОСОВ В КОНСТРУКТОРЕ»

ЦЕЛЬ РАБОТЫ:

– научиться создавать многотабличные запросы в окне конструктора;

– научиться создавать перекрестные запросы в окне конструктора.

Ход работы

МНОГОТАБЛИЧНЫЕ ЗАПРОСЫ

Для формирования многотабличного запроса необходимо добавить в окно конструктора запросов все таблицы, участвующие в выборке, и определить условия их объединения. Для добавления таблицы выберите команду **Запрос/Добавить таблицу** или нажмите кнопку **Добавить таблицу** на панели инструментов. Если в БД установлены отношения между таблицами, участвующими в запросе, то эта связь будет отображаться в виде линии, соединяющей таблицы. В этом случае не придется устанавливать связь между таблицами в конструкторе запросов. Если же между таблицами не существует связи, то можно установить требуемую связь, используя механизм «перенести – и - оставить». Для этого выберите поле в одной из таблиц и перенесите его на связываемое поле в другой таблице.

Объединение двух таблиц

Задание 1. Откройте учебную базу данных Борей. Выполните следующий запрос: определить тип доставки товаров клиентам.

ПОРЯДОК РАБОТЫ

Для выполнения запроса используется информация из двух связанных таблиц Заказы и Заказано. В БД между этими таблицами уже определены постоянные отношения. Для решения поставленной задачи выполните следующее:

1. Добавьте таблицу Заказы в новое окно конструк-

тора запросов.

2. Добавьте таблицу Заказано.

3. Перенесите в бланк запроса из таблицы Заказано поле Код товара, а из таблицы Заказы – поля Код заказа и Доставка. Установите сортировку по возрастанию для полей Код товара и Доставка.

4. Выполните запрос.

Условие отбора в многотабличных запросах

Задание 2. Выбрать список заказов, в которых товар с кодом 2 доставляется по почте.

ПОРЯДОК РАБОТЫ

Возьмите за основу запрос, созданный в предыдущем примере, и выполните следующее:

1. Для создания условия отбора товара перейдите на строку Условие отбора поля Код Товара и введите код товара. Для этого поля снимите флажок Вывод на экран, так как не имеет смысла выводить столбец, каждая строка которого будет содержать одно и то же значение.

2. Перейдите на Условие отбора поля Доставка и задайте код доставки почтой, равный значению 2, а затем снимите флажок Вывод на экран.

3. Добавьте в бланк запроса поля, содержащие сведения о сотруднике и дату исполнения заказа.

4. Выполните запрос.

Объединение нескольких таблиц

Задание 3. Выбрать товары указанного типа (напр., рыбопродукты), проданные клиентам, находящимся в указанной стране (напр., Франция).

ПОРЯДОК РАБОТЫ

Для решения задачи потребуются таблицы Клиенты, Заказы, Заказано и Товары. В таблице Клиенты содержится информация о местонахождении клиентов, а в таблице Товары – код типа товара. Таблицы Заказы и Заказано позволяют связать информацию о покупателе с проданным ему товаром. Выполните следующие действия:

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Клиенты, Заказы, Заказано и Товары.

2. Добавьте в бланк запроса поля, содержание название клиента и марку товара, атакже установите для них флажок Вывод на экран.

3. Для задания условия отбора клиентов по стране местонахождения добавьте в бланк запроса поле Страна таблицы Клиенты. Перейдите на строку Условие отбора этого поля и введите название страны (Франция). Так как выводит этот столбец не имеет смысла, снимите флажок Вывод на экран.

4. Добавьте поле Код типа таблицы Товары и в строке Условие отбора задайте код типа рыбопродуктов, равный 8, а затем снимите флажок Вывод на экран.

5. Выполните запрос.

Итоговые запросы

Часто бывает недостаточно имеющихся в таблицах данных. Например, нужно найти сумму, максимальную величину в поле или количество записей, содержащих определенную величину. В этом случае необходимо выполнить итоговые запросы.

Для создания итогового запроса выберите **Вид/ Групповые операции** или нажмите кнопку **Групповые операции** на панели инструментов. В бланке запроса появится новая строка с наименованием **Групповая операция**. В этой строке надо указать тип выполняемого вычисления.

Таблица 1 содержит перечень всех допустимых видов итоговых операций, которые можно выбрать из раскрывающегося списка в строке **Групповая операция**.

Таблица 1.

Значение	Выполняемая операция
Sum	Сложение
Avg	Среднее значение
Min	Минимальное значение
Max	Максимальное значение
Count	Количество записей, содержащих значение
StDev	Стандартное отклонение
Var	Дисперсия
First	Значение в первой записи
Last	Значение в последней записи

Операция Count действует следующим образом: просматриваются все записи для указанного поля и вычисляется количество записей, которые содержат в этом поле какое-либо значение. Данная операция пропускает пустые значения.

Для удаления строки **Групповая операция** достаточно нажать еще раз на кнопку **Групповая операция**.

Определение суммы всех значений по полю

Задание 4. Определить итоговую стоимость всех перевозок товаров.

ПОРЯДОК РАБОТЫ

Стоимость перевозки указана в поле Стоимость доставки таблицы Заказы. Для создания запроса, вычисляющего их общую сумму, выполните следующие действия:

1. Откройте новое окно конструктора запросов и добавьте в него таблицу Заказы.
2. Добавьте в бланк запроса поле Стоимость доставки.
3. Выберите команду Вид/ Групповые операции для добавления в бланк запроса строки Групповая операция.
4. Нажмите кнопку мыши в первом столбце строки Групповая операция и из раскрывающегося списка выберите значение **Sum**, которое указывает MS Access просуммировать все значения поля Стоимость доставки.
5. Нажмите кнопку Запуск.

Определение максимального и среднего значения по полю

Задание 5. Определить не только итоговую стоимость доставки, но также максимальное и среднее значение стоимости доставки.

ПОРЯДОК РАБОТЫ

Для решения задачи вернитесь в окно конструктора запросов предыдущего запроса и добавьте в бланк запроса поле Стоимость доставки еще 2 раза. Для второго поля Стоимость доставки из раскрывающегося списка в строке Групповая операция выберите **Max**, а для третьего – **Avg**. Выполните запрос.

Определение условия выборки в итоговых запросах

Задание 6. Определить стоимость доставки товаров по почте.

ПОРЯДОК РАБОТЫ

1. В бланк запроса, созданного в предыдущем примере, добавьте поле Доставка. Сразу после добавления этого поля в строке Групповая операция устанавливается значение **Группировка**.
2. Для ввода условия выборки измените это значение на **Условие**. При выборе этого значения автоматически снимается флажок **Вывод на экран**, и поле не выводится на экран при выполнении запроса.
3. После этого введите в строке Условие отбора требуемое условие. Для определения итоговой стоимости доставки по почте напечатайте код доставки, равный 2.
4. Для просмотра результатов выборки нажмите кнопку Запуск.

Группировка полей запроса

Группировка позволяет получить вычисляемую информацию о подгруппах таблицы.

Задание 7. Получить сведения об итоговом количестве заказов каждого вида товара.

ПОРЯДОК РАБОТЫ

1. Добавьте таблицу Заказано в окно конструктора запросов.
2. В первый столбец бланка запроса поместите поле Код Товара.
3. Во второй столбец поместите Код Заказа.
4. Выберите команду **Вид/Групповые операции** для добавления в бланк запроса строки Групповая операция.
5. В строке Групповая операция столбца Код Товара установите значение **Группировка**, а для столбца Код Заказа – значение **Count**.
6. Количество заказов товара представляет интерес, но также интересно знать количество проданного товара. Один товар может продаваться очень часто, но маленькими партиями, а другой товар реже, но крупными партиями. Поэтому добавьте в запрос поле Количество и в строке Группо-

вая операция установите значение **Sum**.

7. Установите сортировку по убыванию для поля Код Заказа.
8. Выполните запрос.

Поле, используемое для группировки, не обязательно должно находиться в той же таблице, что и итоговое поле. Напр., вместо товаров можно использовать для группировки типы товаров, а поле с этими данными находится в таблице Товары.

Задание 8. Создайте указанный запрос.

ПОРЯДОК РАБОТЫ

1. Откройте окно конструктора запросов для предыдущего запроса.
2. Добавьте в него таблицу Товары.
3. Удалите из бланка запроса поле Код Товара.
4. Добавьте в бланк запроса поле Код Типа из таблицы Товары и в строке Групповая операция установите значение **Группировка**.
5. Выполните запрос.

Группировка по нескольким полям

Можно группировать данные не только по одному полю, но и по нескольким полям одновременно, а также создавать группы внутри групп. Рассмотрим эти возможности на примере.

В БД Борей существует 3 типа доставки товаров: РосТранс, Почта, Иное.

Задание 9. Определить количество заказов по каждому покупателю и типу доставки.

ПОРЯДОК РАБОТЫ

Можно сгруппировать записи сначала по покупателям, затем – по типам доставки. Порядок полей группировки, размещенных в бланке запроса, определяет порядок группировки выбираемых данных. Для создания запроса выполните следующие действия:

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Клиенты, Заказы и Доставка.
2. В бланк запроса поместите поля Название из таблиц Клиенты и Доставка, а также поле Код Заказа из таблицы Заказы.

3. Нажмите кнопку Групповые операции на панели инструментов.

4. В строке Групповая операция первых двух столбцов установите значение **Группировка**, а для столбца Код Заказа – значение **Count**.

5. Выполните запрос.

Включение в запрос выражений

MS Access позволяет выполнять итоговые операции над вычисляемыми полями выборки. Напр., таблица Заказы содержит данные о клиентах, а таблица Заказано – о количестве товаров, проданных в каждой партии, и цене товара. На основе этой информации можно вычислить итоговую стоимость заказа каждым покупателем по каждому товару.

Чтобы включить в итоговый запрос выражение, надо добавить в запрос вычисляемое поле и указать тип итоговых вычислений, осуществляемых над этим полем.

Выражения для вычисления могут содержать формулы, связанные арифметическими операторами. В качестве элементов формулы могут использоваться поля, константы и функции. Для изменения порядка вычислений в выражениях используются круглые скобки.

Задание 10. Определить итоговую стоимость каждого проданного товара отдельно для каждого покупателя.

ПОРЯДОК РАБОТЫ

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Заказы и Заказано.

2. В бланк запроса поместите поля Код Клиента из таблицы Заказы и Код Товара из таблицы Заказано.

3. В бланк запроса добавьте вычисляемое поле и введите для него выражение для вычисления стоимости партии товаров:

Стоимость: [Заказано] ! [Количество] * [Заказано] ! [Цена]

4. Нажмите кнопку Групповые операции на панели инструментов для добавления в бланк запроса строки Групповая операция.

5. В строке Групповая операция первых двух столбцов установите значение **Группировка**, а для столбца, содержащего выражение, – значение **Sum**.

6. Выполните запрос. Вы должны получить список клиентов и итоговую стоимость заказанных ими товаров.

Использование более сложных условий отбора записей в итоговых запросах

Задание 11. Найти клиентов, у которых для каждого типа доставки было оформлено более трех заказов.

ПОРЯДОК РАБОТЫ

1. Откройте новое окно конструктора запросов и добавьте в него таблицы Клиенты, Заказы и Доставка.
2. В бланк запроса поместите поля Название из таблиц Клиенты и Доставка, а также поле Код Заказа из таблицы Заказы.
3. Нажмите кнопку Групповые операции на панели инструментов для добавления в бланк запроса строки Групповая операция.
4. В строке Групповая операция первых двух столбцов установите значение **Группировка**, а для столбца, Код Заказа - значение **Count**.
5. Перейдите на строку Условие отбора столбца Код Заказа и напечатайте: >3.
6. Выполните запрос. Из списка клиентов и соответствующих им типов доставки должны быть выбраны только те записи, в которых количество заказов превышает 3.

Задание на лабораторную работу

1. Изучите методические указания и выполните описанные в них задания, результаты выполнения заданий предъявите преподавателю на экране дисплея.
2. Используя справку в Access, изучите назначение и способы создания перекрестных запросов. Постройте в конструкторе следующий перекрестный запрос к базе данных Борей: по каждому клиенту (коду клиента) и каждому типу доставки (название типа) посчитать число заказов (кодов заказа), дата исполнения которых находится в диапазоне с 1.01.1997 по 1.02.1997. В качестве заголовков строк выберите коды клиентов, в качестве заголовков столбцов – название типа доставки. Используйте для образца имеющийся в базе данных Борей запрос «Квартальные отчеты» или «Квартальные обороты по товарам».

3. Создайте в конструкторе различные варианты многотабличных запросов к вашей базе данных (созданной по заданию лабораторной работы № 1), подобные запросам, описанным в методических указаниях.
4. Оформите отчет по лабораторной работе.

Требования к отчету

Отчет должен содержать:

1. Цель работы.
2. Содержимое таблиц базы данных, используемых в запросах.
3. Формулировку запросов.
4. Описание процесса создания запросов.
5. Результаты выполнения запросов.

Контрольные вопросы

1. Опишите последовательность действия по созданию многотабличного запроса в конструкторе.
2. Опишите действия по созданию итогового запроса.
3. Перечислите допустимые виды итоговых операций.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ НА ТЕМУ: РАЗРАБОТКА ФОРМ В РЕЖИМЕ КОНСТРУКТОРА»

Цель работы: освоить средства создания простых форм в СУБД **Access**.

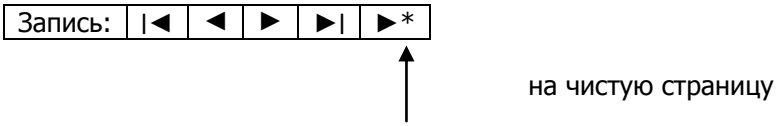
Руководство к выполнению

Формы – средство представления информации, используются для ввода и редактирования данных.

Простейший способ создания формы:

1. Открыть окно БД.
2. Активизировать категорию «Таблицы».
3. Установить указатель на таблицу, для которой создается форма (например, таблица «Клиенты»).
4. Выполнить команду «Вставка|Автоформа» или нажать кнопку «Новый объект:автоформа». Из раскрывающегося списка выбрать опцию «Автоформа».

Кнопки перемещения по записям: PageUp, PageDown, а также



на чистую страницу

Переход из формы в режим таблицы: «Вид|Режим таблицы».

Альтернативные способы создания формы:

- Конструктор;
- Мастер;
- Автоформа: в столбец;
- Автоформа: ленточная;
- Автоформа: табличная.

Для создания формы

1. В окне БД активизировать категорию «Формы».
2. Нажать кнопку «Создать».
3. В открывшемся окне «Новая форма» выбрать вариант.
4. В поле ввода внизу выбрать из списка требуемую таблицу (Клиенты).

5. Нажать ОК.

В ленточной автоформе на каждой странице размещается более одной записи, что позволяет сравнивать данные нескольких записей.

Табличная автоформа создается аналогично ленточной, но при запуске на формирование форма открывается в режиме таблицы. Для перехода в режим формы выполнить команду «Вид|Режим формы». На экране появится табличная автоформа. Она похожа на ленточную, но каждая запись находится на отдельной странице.

Создание форм с помощью мастера

В окне диалога «Новая форма» выбрать из списка «Мастер форм» и нажать ОК.

Раскрыть список «Таблицы и запросы» и выбрать таблицу «Клиенты».

Из списка «Доступные поля» перенести в список «Выбранные поля» все поля таблицы «Клиенты». Нажать «Далее».

В следующем окне задать внешний вид формы. Нажать «Далее».

Выбрать стиль формы. Нажать «Далее».

Ввести имя формы. Установить опцию «Открыть форму для просмотра». Нажать «Готово».

Создание форм в конструкторе

В окне конструктора форм в правой части экрана находится перемещаемая панель элементов.

Окно содержит сетку для выравнивания объектов формы.

Любая форма состоит из объектов, которые имеют характерные для них свойства. Для каждого объекта можно определить действия, выполняемые при наступлении определенных событий. Процесс создания формы состоит в размещении объектов в форме и определении для них свойств, связанных с ними событий и выполняемых действий.

При открытии окно конструктора формы содержит одну область – область данных. Кроме этого форма может содержать область заголовка, примечания и нижний и верхний колонтитулы. Для добавления этих областей используется команда «Вид|Заголовок/Примечание формы» и «Вид|Колонтитулы».

Размеры областей можно изменять: установить указатель мыши на верхнюю часть границы между областями. Когда курсор мыши примет вид двусторонней стрелки с прямоугольником посередине, нажать кнопку мыши и переместить границу при нажатой

кнопке.

Панель элементов используется для размещения объектов в форме.

Кнопка «Выбор объектов» - для выделения элемента управления, раздела или формы;

кнопка «Мастера» - для создания элементов управления;

кнопка «Надпись» - для размещения в форме текста;

кнопка «Поле» - для добавления полей данных в форму.

Свойства

Чтобы получить доступ к свойствам объекта, надо выделить его и выбрать команду «Вид|Свойства» или нажать кнопку «Свойства» на панели инструментов.

Объекты

В процессе создания формы можно перемещать, удалять или изменять размеры объектов и их свойства.

Объект можно перемещать мышью, не выделяя его. Или выделить объект, установить указатель мыши на границу выбранного объекта. Когда появится значок открытой руки, нажать кнопку мыши и переместить объект.

Можно изменять размеры объектов (меню «Формат|Размер»), выравнивать их. Для выравнивания используется меню «Формат|Выровнять».

Для точного размещения объектов используется сетка. Выполнить «Формат|Привязать к сетке».

Для перемещения объекта на небольшое расстояние нажать CTRL и клавиши – стрелки.

Для выделения нескольких объектов нажать Shift и нажать мышью все объекты.

Практическое задание: создание формы в режиме конструктора

Выберите в окне БД:

Форма → Создать →Конструктор → Формат |Автоформат или кнопка «Автоформат» на панели инструментов (для изменения стиля формы).

Измените размер формы: установите указатель мыши в правый нижний угол формы. Когда курсор примет вид двойной стрелки, установите требуемый размер формы.

Откройте окно свойств: «Вид|Свойства».

На вкладке «Макет» в поле «Подпись» введите текст заго-

ловка формы – Покупатели.

На вкладке «Данные» установите значение свойства «Источник записей» - выберите из списка таблицу Клиенты.

Создание заголовка

Выберите «Надпись» на панели элементов.

Установите указатель мыши на место расположения текста и введите текст заголовка (Покупатели), после чего нажмите Enter.

Выделите созданный объект.

Используя панель инструментов «Формат», задайте для него шрифт, размер, цвет шрифта, цвет и толщину границы, цвет фона.

Размещение полей ввода

Выберите на панели элементов «Поле».

Нажмите мышью место размещения поля. Выделите поле ввода и задайте для него свойства.

Другой способ:

Выберите на панели инструментов «Список полей». Перенесите из данного списка в форму все поля таблицы Клиенты.

Используя панель инструментов форматирования, задайте для них шрифт, размер, цвет шрифта и фона.

Создание кнопок

Выберите на панели элементов «Кнопка».

Установите указатель мыши на место в форме, где будет размещена кнопка, и нажмите кнопку мыши.

Запустится мастер создания кнопки.

Создайте кнопку перехода на следующую запись, на предыдущую запись, кнопку открытия формы по таблице Заказы (по нажатию кнопки должна открываться подчиненная форма Заказы).

Улучшение внешнего вида формы

Разместите поля, сгруппировав их по содержимому.

Одна группа – КодКлиента, Название, …

Вторая группа – вся адресная информация.

Третья группа – телефон, факс.

Выровняйте надписи полей по правому краю, поля – по левому краю.

Разделите группы линией.

Сделайте линию утопленной.

Увеличьте ширину и высоту области данных (выберите «Правка|Выделить все» и перетащите все элементы управления так, чтобы оставить 0,6 см свободного пространства от каждого каря области данных).

Заключите все элементы управления в прямоугольник: щелкните на кнопке «Прямоугольник», щелкните там, где будет левый верхний угол фигуры, и протащите указатель, создавая прямоугольник необходимого размера.

Установите светло-серый фон для него. Кнопкой «Оформление» измените эффект с «вдавленного» на «приподнятое». Командой «Формат|На задний план» уберите непрозрачный прямоугольник на задний план.

Еще раз щелкните на кнопке «Прямоугольник» и установите темно-серый фон и «утопленное» оформление. Между первым прямоугольником и краями области данных нарисуйте второй прямоугольник. Выполните команду «Формат|На задний план». Теперь кажется, что элементы управления плавают на светло-сером прямоугольнике формы, обведенном темно-серым контуром.

Добавьте надпись «Клиенты» в заголовок формы. Установите для нее шрифт Arial размером 14 пунктов с наклонным полужирным начертанием. Чтобы получить эффект с тенью, создайте надпись с черным цветом символов. Затем скопируйте ее в буфер обмена и вставьте в заголовок формы. Измените цвет символов на белый и выполните «Формат| На задний план». Выключите привязку к сетке и переместите белую надпись так, чтобы она оказалась чуть ниже и правее первой надписи.

Область выделения, полосы прокрутки

Форма выводит только одну запись, поэтому область выделения с левой стороны формы бесполезна. И так как форма выводит все данные в одном окне, нам не нужны и полосы прокрутки. Откройте окно свойств формы и установите для свойства «Область выделения» значение «Нет», для свойства «Полосы прокрутки» - значение «Отсутствуют».

Последовательность перехода

Выполните «Вид| Последовательность перехода». Щелкните кнопку «Авто», чтобы изменить последовательность перехода по клавише Tab по полям формы таким образом, чтобы она соответствовала расположению элементов управления в форме слева

направо и сверху вниз. Можете внести дополнительные изменения в список, щелкая на области выделения элементов управления и перетаскивая их в нужное место в списке. Чтобы сохранить изменения, щелкните ОК.

Можно запретить пользователю сворачивать и разворачивать окно формы, открывать оконное меню (команды Восстановить, Переместить, Размер, Свернуть, Развернуть, Закрыть, Следующее).

При установке свойства формы «Кнопка оконного меню» в «Нет» из строки заголовка окна формы удаляются все кнопки.

Поле со списком

Выделите поле «Код Клиента» и поле «Страна» и преобразуйте их в поле со списком командой «Формат|Преобразовать элемент в». Но при этом придется самим установить свойства, определяющие работу списка. В окне свойств поля в ячейке «Тип источника строк» установите значение «Таблица» или «Запрос». В ячейке «Источник строк» задайте имя соответствующего запроса, который необходимо предварительно создать в окне Запросы.

Создать поле со списком можно с помощью мастера. Для этого удалить соответствующее поле из формы. Нажать кнопку «Мастера» на панели элементов, а затем кнопку «Поле со списком» и перетащить в форму поле «Код Клиента» из списка полей таблицы. Запустится мастер по созданию полей со списком. В этом поле будут использоваться значения из запроса, так что установите первый переключатель и щелкните «Далее».

Установка значений с помощью макросов

Одним из способов обеспечения целостности БД является автоматическая установка значений определенных полей при вводе значений других полей таблицы.

Например, с помощью макроса можно установить значение полей Страна и Индекс в форме Клиенты в зависимости от значения поля Город.

Создайте макрос, содержащий следующие макрокоманды:

Макрокоманда	Аргумент	Значение
Задать Значение	Элемент Выражение	[Страна] DlookUp("[Страна]";"[Клиенты]";"[Клиенты].[Город]=[Forms]![ФормаКлиенты]![Город]")

Задать Значение Условие макрокоманды: Not([Страна]=" "Or [Страна] Is Null)	Элемент Выражение	[Индекс] DlookUp("[Индекс]";"[Клиенты]";"[Клиенты].[Город]= [Forms]![ФормаКлиенты]![Город]")
К Элементу Управления	Имя Элемента	Индекс
Задать Значение	Элемент Выражение	[Forms]![ФормаКлиенты]![Индекс].SelStart 255

Когда пользователь вводит название города, первая макрокоманда «Задать Значение», используя встроенную функцию DlookUp, запрашивает соответствующее значение поля Страна из таблицы Клиенты. Если это значение не является пустой строкой или Null, то вторая макрокоманда «Задать Значение» запрашивает индекс из таблицы Клиенты, затем макрокоманда «К Элементу Управления» передает фокус элементу управления Индекс, а следующая макрокоманда «Задать Значение» устанавливает свойство SelStart элемента управления Индекс в наибольшее значение (255), чтобы поместить курсор в самом конце выводимых в нем данных. Теперь пользователь может изменить две последние цифры индекса.

Столбец «Условие» для второй макрокоманды содержит следующее выражение: Not ([Страна] = " " Or [Страна] Is Null)

Для создания макроса в окне БД Борей щелкните Макрос → Создать, чтобы открыть новое окно макроса. В этом окне верхняя часть используется для определения макроса, а нижняя предназначена для ввода значений аргументов макрокоманд, включенных в него. В верхней части окна первоначально присутствуют два столбца с заголовками «Макрокоманда» и «Примечание». Можно добавить столбцы «Имя макроса» и «Условие», щелкнув на соответствующих кнопках на панели инструментов.

После создания макроса откройте форму Клиенты в режиме конструктора. Выделите поле Город и установите в качестве значения свойства «После обновления» имя вашего макроса. Проверьте работу макроса.

Задание на лабораторную работу

1. Изучите методические указания.
2. Выполните описанные в них действия и создайте форму в режиме конструктора.
3. Результаты работы предъявите преподавателю на экране дисплея.

Контрольные вопросы

1. Для каких целей используется форма в БД Access?
2. Перечислите способы создания форм в Access?
3. Опишите последовательность действия для создания формы.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ ЛАБОРАТОРНОЙ РАБОТЫ НА ТЕМУ: РАЗРАБОТКА СЛОЖНОЙ ФОРМЫ»

Цель работы: получить навыки создания сложных форм с помощью инструментальных средств СУБД **MS Access**.

Руководство к выполнению

Сложная форма предназначена для просмотра и редактирования таблиц, связанных отношением «один – ко - многим».

Для отображения записей основной и подчиненной таблиц можно использовать многотабличную иерархическую форму. Иерархической является форма, содержащая главную и одну или несколько подчиненных форм. Подчиненные формы отображают данные из таблиц со стороны «много».

Создание сложной формы с помощью мастера

Простейший способ создания многотабличных форм – использование мастера.

Создайте с помощью мастера иерархическую форму для связанных таблиц Клиенты и Заказы, имеющие отношение 1 : М. Для этого выполните следующие действия:

1. В окне БД выберите Формы → Создать → Мастер форм.
2. В окне «Создание форм» выберите из списка «Таблицы и запросы» главную таблицу Клиенты и поместите в список «Выбранные поля» все поля таблицы.
3. Выберите из списка «Таблицы и запросы» подчиненную таблицу Заказы. Поместите в список «Выбранные поля» «Код Заказа», «Код Сотрудника», «Дата исполнения». Нажмите «Далее».
4. Выберите вид представления - подчиненные формы. Далее.
5. Выберите внешний вид формы – табличный. Далее.
6. Выберите требуемый стиль. Далее. Задайте имена форм. Нажмите «Готово».

С помощью конструктора форм можно внести в нее изменения.

Для сравнения снова вызовите мастер форм и создайте аналогичную форму, но выбрав опцию «Связанные формы».

Построение сложной формы в режиме конструктора

1. Сначала создайте подчиненную форму:

Создайте базовый запрос с помощью конструктора (этот запрос будет источником данных для формы):

Добавьте таблицы Клиенты, Заказы. В бланк запроса добавьте все поля таблицы Заказы и поле «Код Клиента» из таблицы Клиенты.

Постройте новую форму на основе только что созданного запроса. Выберите табличный режим в окне свойств. Перенесите все поля (кроме «Код Клиента») базового запроса в форму. Установите «Тип источника записей» - имя запроса.

2. Постройте главную форму. Чтобы внедрить в нее подчиненную форму, щелкните на кнопке «Подчиненная форма/отчет» (кнопка «Мастера» должна быть не нажата), щелкните в левом верхнем углу свободного места и протащите указатель, чтобы создать элемент управления «Подчиненная форма». Введите в качестве значения его свойства «Объект - источник» имя ранее созданной подчиненной формы.

Два свойства подчиненной формы определяют связь между основной и подчиненной формой: «Основные поля» и «Подчиненные поля».

Основные поля – поле «Код Клиента» из таблицы Клиенты.

Подчиненные поля – поле «Код Клиента» из таблицы Заказы.

Закончив, убедитесь, что форма правильно отражает информацию (перейдите в режим формы).

Использование в форме набора вкладок

Постройте сложную форму «Клиенты» с набором из трех вкладок: Заказы, Товары, Поставщики.

На вкладке «Заказы» должны быть выведены заказы текущего клиента.

На вкладке «Товары» выводятся товары из заказов данного клиента.

На вкладке «Поставщики» указываются поставщики этих товаров.

Предварительно создайте три запроса по заказам, товарам, поставщикам.

По заказам необходима следующая информация: код заказа, код сотрудника, дата исполнения, стоимость доставки. Задайте сортировку по коду заказа.

По товарам необходима следующая информация: код заказа, код товара, марка товара, цена, количество. Сортировка по

коду товара.

По поставщикам необходима следующая информация: код товара, название поставщика, город, страна. Сортировка по коду товара.

В каждом запросе обязательно добавляйте поле «Код Клиента» из таблицы Клиенты.

Создайте на основе каждого запроса формы, которые должны быть в виде таблиц.

Создайте главную форму по клиентам.

Чтобы добавить в форму набор вкладок:

1. На панели элементов щелкните кнопку «Набор вкладок» и создайте элемент управления шириной 12 см, высотой 4 см. Access построит элемент с двумя вкладками. Откройте окно свойств и установите для свойства «Ширина ярлычка» значение 1,5 дюйма (3,81 см).
2. Пока фокус в наборе вкладок выберите команду Вставка → Вкладка. Access добавит третью вкладку.
3. Выделите первую вкладку и введите «Заказы» в ячейке свойства «Подпись».
4. Выделите вторую вкладку и введите «Товары» в ячейке свойства «Подпись».
5. Выделите третью вкладку и введите «Поставщики» в ячейке свойства «Подпись».
6. Выделите вкладку «Заказы». На панели элементов щелкните кнопку «Подчиненная форма/отчет» и установите значение «Нет» для свойства «Добавление подписи». Добавьте элемент управления «Подчиненная форма» на вкладку «Заказы». В ячейке свойства «Объект - источник» введите имя соответствующей формы и укажите значения свойств «Основные поля» и «Подчиненные поля».
7. Выделите вкладку «Товары» и внедрите в нее соответствующую подчиненную форму.
8. Выделите вкладку «Поставщики» и внедрите в нее соответствующую форму.

Связывание форм с помощью кнопок

Теперь в форме по клиентам вместо набора вкладок создайте три кнопки: Заказы, Товары, Поставщики. По нажатию этих кнопок должны открываться соответствующие формы.

Кнопки создайте с помощью мастера.

В ячейке свойства «Всплывающая подсказка» для каждой кнопки введите подходящий текст подсказки.

Задание на лабораторную работу

1. Изучите методические указания.
2. Выполните описанные в них действия и создайте форму в режиме конструктора.
3. Результаты работы предъявите преподавателю на экране дисплея.

Контрольные вопросы

1. Для чего предназначена сложная форма?
2. Опишите последовательность действий при создании сложной формы с помощью мастера.
3. Опишите последовательность действий при создании сложной формы с помощью конструктора.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ НА ТЕМУ: «СОЗДАНИЕ ОТЧЕТОВ В ACCESS»

Цель работы: научиться создавать отчеты в **Access**.

Руководство к выполнению

Основная сфера применения форм – обеспечение возможности просмотра отдельных или небольших групп связанных записей. Отчеты же представляют собой наилучшее средство отображения информации из базы данных в виде печатного документа. Разработка отчета очень похожа на разработку формы. Используется та же панель элементов, тот же список полей и окно свойств. Построим относительно несложный отчет, пройдя шаг за шагом всю цепочку его создания.

Создайте в **Access** базу данных поставщиков, деталей и проектов, содержащую информацию о поставках деталей поставщиками для различных проектов. База данных включает четыре таблицы. Таблица S содержит информацию о поставщиках, размещенную в следующих полях:

SN - номер поставщика, SNAME - имя поставщика, STATUS – статус поставщика, CITY – город поставщика.

Таблица P содержит информацию о деталях, размещенную в следующих полях:

PN - номер детали, PNAME - название детали, COLOR – цвет детали, WEIGHT – вес детали, CITY – город, где хранится деталь.

Таблица J содержит информацию о проектах, размещенную в следующих полях:

JN - номер проекта, JNAME - название проекта, CITY – город, где реализуется проект.

Таблица SPJ содержит информацию о поставках деталей поставщиками для различных проектов, размещенную в следующих полях:

SN - номер поставщика, PN - номер поставляемой детали, JN - номер проекта, для которого поставляется деталь, QTY – объем поставки (количество поставляемых деталей).

Таблицы данных приведены ниже:

Таблица S

SN	SNAME	STATUS	CITY
S1	Смит	20	Лондон
S2	Джонс	10	Париж
S3	Блэк	30	Париж
S4	Кларк	20	Лондон
S5	Адамс	30	Афины

Таблица P

PN	PNAME	COLOR	WEIGH	CITY
P1	Муфта	Крас-	12	Лондон
P2	Болт	Зеле-	17	Париж
P3	Спица	Синий	17	Рим
P4	Колпак	Крас-	14	Лондон
P5	Диск	Синий	12	Париж
P6	Вал	Крас-	19	Лондон

Таблица J

JN	JNAME	CITY
J1	Sorter	Париж
J2	Display	Рим
J3	OCR	Афины
J4	Console	Афины
J5	RAID	Лондон
J6	EDS	Осло
J7	Tape	Лондон

Таблица SPJ

SN	PN	JN	QTY
S1	P1	J1	200
S1	P1	J4	700
S2	P3	J2	200
S2	P3	J7	800
S2	P3	J1	400
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400

S2	P5	J2	100
S3	P3	J1	200
S3	P4	J2	500
S4	P6	J7	300
S4	P6	J3	300
S5	P5	J7	100
S5	P1	J4	100
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P6	J2	200
S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

Требуется создать отчет - справку о поставках деталей для данного проекта. Отчет будет запускаться из формы ***Проекты***.

СОЗДАНИЕ ФОРМЫ С ОТЧЕТОМ

В окне базы данных выберите вкладку «Формы» и создайте новую форму ***Проекты***, добавив в нее все поля таблицы ***J***.

ПОСТРОЕНИЕ БАЗОВОГО ЗАПРОСА

При создании запроса нам понадобится информация из таблиц ***J, SPJ*** и ***P***. Запрос содержит данные о деталях, поставляемых для проекта, который отображается в форме ***Проекты***.

1. В окне базы данных выберите вкладку «**Запросы**». Он находится в списке «Объекты». Нажмите кнопку «Создать» на панели инструментов окна базы данных. Появится окно «Новый запрос». Выберите пункт «Конструктор» и нажмите кнопку «ОК».
2. В окне «**Добавление таблицы**» добавьте таблицы ***J***, ***SPJ*** и ***P***.
3. Поместите в бланк запроса поле ***JN*** из таблицы ***SPJ***, ***JNAME*** из таблицы ***J***, все поля из таблицы ***P***.
4. В запрос должны попасть данные только по одному конкретному проекту. Его номер отображен в форме ***Проекты***. Для этого добавим условие отбора по полю ***JN***: **[Forms]![Проекты]![JN].**
5. Запрос должен выдавать суммарный объем поставок

данной детали для данного проекта. Для этого добавьте в запрос соответствующую групповую операцию.

6. Установите порядок сортировки записей, попавших в запрос. Они должны отображаться в документе в порядке возрастания порядковых номеров проектов.

7. Сохраните созданный запрос. Не забудьте, что корректно он будет запускаться только из формы ***Проекты***.

РАЗРАБОТКА ОТЧЕТА В РЕЖИМЕ КОНСТРУКТОРА

Сделайте активным окно базы данных. Выберите вкладку «**Отчеты**».

Сделайте щелчок левой кнопкой мыши по пиктограмме «Создать».

Появится диалоговое окно «Новый отчет» (рис. 1). Выберите пункт «Конструктор» в верхнем окне и созданный Вами запрос в поле с раскрывающимся списком. При завершении этого этапа нажмите кнопку «ОК».

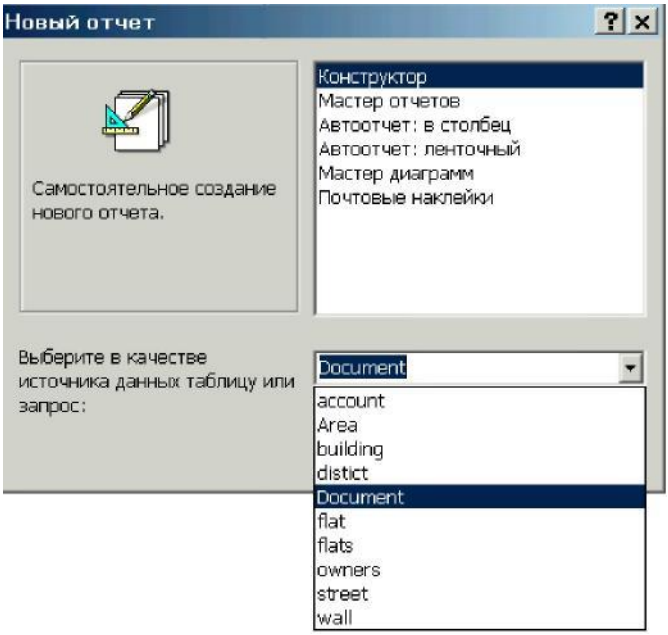


Рис. 1 Создание нового отчета

Заготовка отчета содержит три раздела: верхний и нижний колонтитулы, между которыми находится область данных (см. рис. 2). Вы можете изменить размер любого раздела. Ширина всех разделов должна быть одинаковой. Линейки с сантиметровыми делениями по верхнему и левому краям отчета помогают расположить данные на странице. Если линейки отсутствуют, то для их появления на экране выберите в главном меню Microsoft Access пункт «**Вид**», а в открывшемся подменю строку – «**Линейка**». Верхний и нижний колонтитулы будут напечатаны соответственно вверху и внизу каждой страницы. Их можно убрать совсем с помощью пунктов меню «**Вид**» и «**Колонтитулы**». Вы можете создать заголовок, который будет напечатан только в начале отчета на первой странице. Для этого используйте пункт меню «**Вид**» и строку «**Заголовок/примечание отчета**».

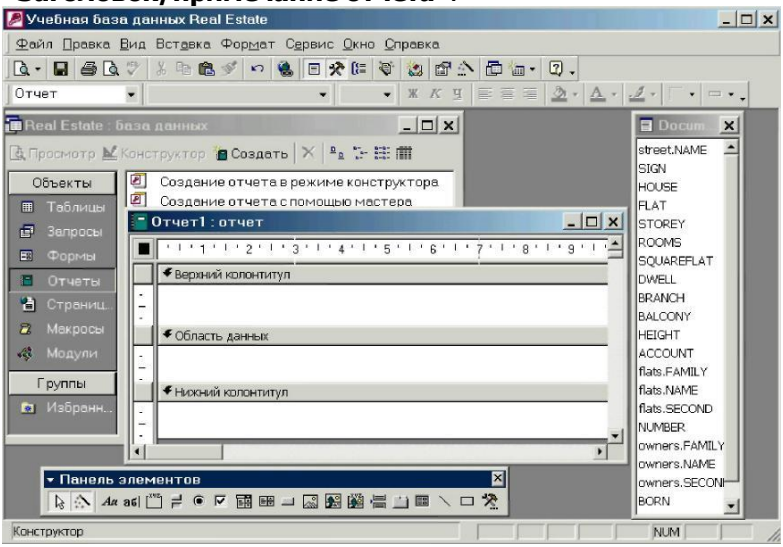


Рис. 2. Конструктор отчетов

Добавление элементов в отчет *Microsoft Access*

Для создания отчета, подобного показанному на рис. 3, выполните следующие действия.

1. Разместите надпись в самом верху заголовка отчета и введите в нее текст «Справка». Выделите надпись и установите шрифт ***Arial*** размером 10 пунктов. Подчеркните текст и сделайте его выделенным. Выберите пункт «Формат» в главном окне Access, а затем «Размер» и строчку «По размеру данных». Размер элемента управления будет настроен в соответствии с назначен-

ным шрифтом. Выполните аналогичные действия по размещению второй строки заголовка отчета «Данные о поставке деталей для проекта».

2. «Перетащите» поля ***JN***, ***JNAME*** из списка полей в заголовок отчета. Выполните форматирование. Для этого выберите нужные элементы управления, щелкая по ним левой кнопкой мыши при нажатой клавише <Shift>. Откройте пункт «Формат» главного окна и пункт «Выровнять» открывшегося подменю. В появившемся окне вам будет предложено несколько способов форматирования.

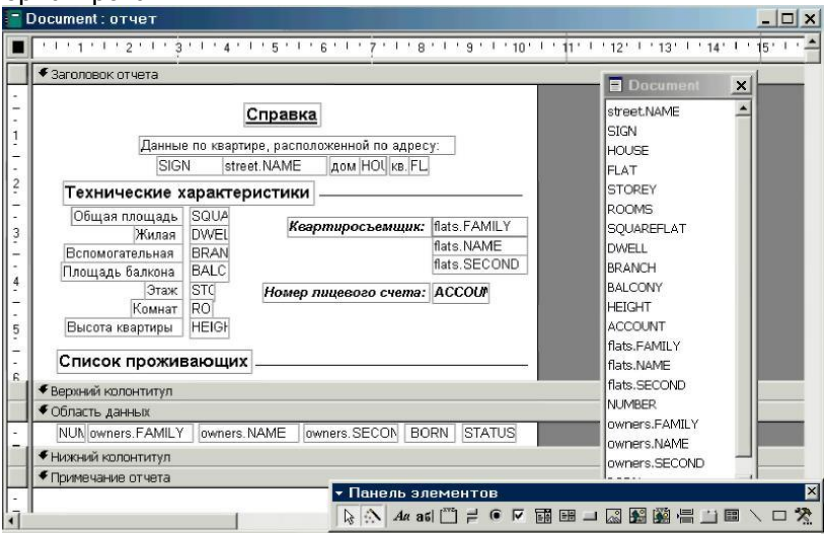


Рис. 3. Окончательный вид отчета ***Document*** в конструкторе отчетов

3. «Перетащите» поля ***PN***, ***PNAME***, ***WEIGHT***, ***Sum-QTY*** в область данных. Удалите подписи, которые к ним сгенерирует Access, и выполните форматирование.

4. Чтобы названия полей ***PN***, ***PNAME***, ***WEIGHT***, ***Sum-QTY*** появлялись только один раз на странице отчета, разместите их в области «Заголовок отчета». При этом создайте следующие надписи: Номер детали, Название детали, Вес детали, Объем поставки. Разместите эти надписи в одну строку.

5. Оформите верхний и нижний колонтитулы. В нижнем колонтитуле расположите поле, присоединенную надпись поля удалите, для свойства поля *Данные* введите значение:

="-страница " & [Page] & " из " & [Pages] & " -"

В области верхнего колонтитула и в верхней части нижнего колонтитула расположите го- ризонтальные линии.

6. Осталось поместить в отчете итоговое поле, в котором будет выводиться суммарный объем поставок всех деталей для данного проекта. Для этого служит раздел «Примечания». Поместите в его верхнюю часть горизонтальную линию, а под ней – свободное поле. Для присоединенной надписи поля задайте текст «Общий объем поставок для проекта», а для свойства *Данные* поля укажите с помощью построителя выражений значение:

=Sum([Sum-QTY])

Отчет готов. Его вид в режиме конструктора приведен на рис. 4, а в режиме предварительного просмотра – на рис. 5.

Закончив работу, не щелкайте по кнопке «Предварительный просмотр». В автономном виде этот отчет не запустится по изложенным ранее причинам. Добавьте кнопку «Просмотр отчета» в форму ***Проекты*** и запускайте отчет из этой формы.

Отчет запускается в режиме предварительного просмотра.

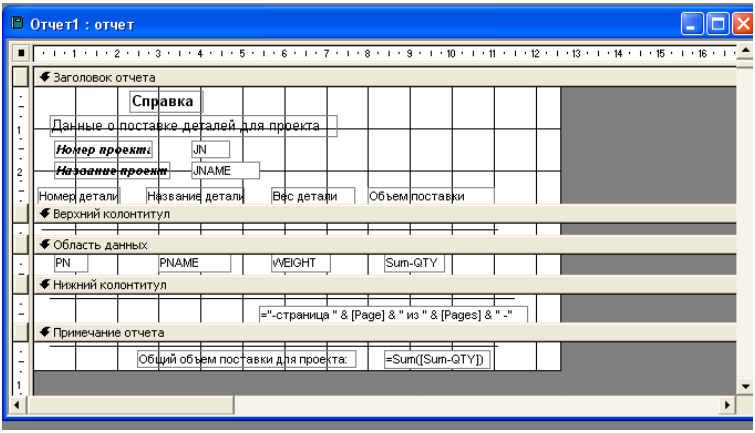


Рис. 4. Вид отчета в режиме конструктора.

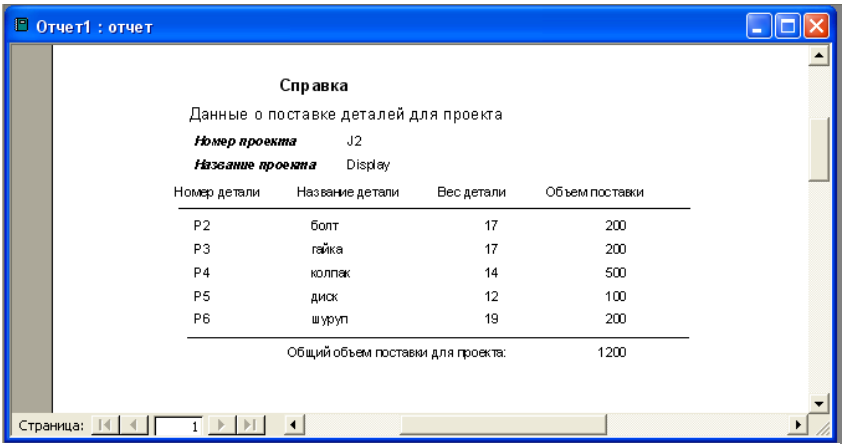


Рис.5. Вид отчета в режиме просмотра.

Задание

1. Изучите методические указания.
2. Создайте описанный в методических указаниях отчет.

Контрольные вопросы

1. Каково назначение отчетов?
2. Опишите последовательность действий по созданию отчета в режиме конструктора.

Лабораторная работа на тему «Программирование на Прологе базы знаний с простыми объектами»

Цель работы: получить навыки программирования баз знаний с простыми объектами на языке логического программирования Пролог.

Справочные сведения

Язык логического программирования Prolog (PROgramming language based in LOGic - язык программирования на основе логики) появился в 70-е годы прошлого столетия одновременно с такими распространенными сейчас языками, как Pascal и C++.

Его ориентация - нетрадиционное применение вычислительной техники: понимание естественного языка, базы знаний, экспертные системы и другие задачи, которые принято относить к проблемам искусственного интеллекта.

Сила этого языка - в принципиально отличном от традиционных процедурных языков программирования подходе к описанию решения задачи. Пролог-программы описывают не процедуры (алгоритмы) решения задач, а логическую модель предметной области с помощью фактов о количественных и качественных характеристиках объектов предметной области (декларативные знания) и правил. Правила описывают определения понятий и отношения (связи) между ними (концептуальные знания).

Таким образом, Пролог в отличие от других процедурных языков является *описательным* .языком.

Несмотря на яркие и очевидные достоинства, Пролог, в отличие от своих сверстников (Pascal и C++), развивался и применялся в узком кругу специалистов в области искусственного интеллекта.

1. Структура Пролог – программы

Обычно программа на Прологе состоит из следующих разделов: DOMAINS - раздел объявления типов данных, которые будут использоваться в программе.

PREDICATES - раздел объявления предикатов, используемых в программе (кроме стандартных предикатов).

CLAUSES - раздел объявления фактов и правил, используемых Пролог- программой для поиска решений поставленных задач.

Перед разделом CLAUSES или после него может распола-

гаться раздел GOAL, в котором определяются вопросы (цели). Вопросы - это средство извлечения информации из Пролог- программы.

В разделе DOMAINS объявляются типы данных для объектов, используемых в качестве аргументов для предикатов. Типы данных называют доменами. Домены подразделяются на простые и структурированные, базисные (стандартные) и нестандартные. Домен описывает множество значений, которые может принимать переменный аргумент предиката в процессе решения задачи.

В языке Пролог имеются следующие базисные типы данных (домены):

symbol - последовательность букв, цифр и знаков подчеркивания, начинающихся со строчной буквы, или любая последовательность символов, заключенная в двойные кавычки (длина не более 250 символов),

string - любая последовательность символов, заключенная в двойные кавычки.

Типы данных symbol и string одинаковы по синтаксису, но обрабатываются программой по-разному. Данные типа *symbol* в отличие от типа *string* запоминаются в таблице символов. Таблица символов размещается в оперативной памяти, поэтому ее использование обеспечивает наиболее быстрый поиск, но требует использование оперативной памяти.

char - отдельный символ, заключенный в апострофы.

Существует набор специальных символьных констант, перед которыми указан слеш. Они представляют специальные функции:

'\n' - перевод строки (новая строка),

'\r' - возврат каретки,

'\t' - табуляция (горизонтальная).

Символьные константы могут быть описаны своим ASCII - кодом после символа слеш:

'\251', '\3'.

integer - целое число в диапазоне от -32768 до 32767,

real - вещественное число, которое может быть представлено в обычной и экспоненциальной формах,

file - имя файла.

В случае использования только базисных доменов раздел DOMAINS может отсутствовать. Но для улучшения читабельности и ясности программы заголовок DOMAINS рекомендуется всегда включать в текст.

Нестандартные домены объявляются с использованием базисных:

< имя нестандартного домена > = < имя базисного домена >

Введение нестандартных доменов улучшает читабельность программы. Кроме того, обеспечивается контроль типов значений переменных смешивать переменные разных типов нельзя.

Примеры объявления нестандартных доменов:

```
DOMAINS
  name = symbol
  num, page = integer
  res = real
```

Объекты перечисленных выше типов называются простыми. Пролог позволяет объявлять и использовать также и составные объекты: структуры, списки и др.

Структуры позволяют рассматривать множество разнородных объектов как единое целое.

Объявление структуры в разделе DOMAINS имеет следующий вид:

< структура > = < функтор > (< домен 1 >, < домен 2 >,… ,<домен N >)

Аргументы функтора - домены - это либо простые базисные домены, либо имена ранее объявленных нестандартных доменов, либо структуры.

Ссылки на доменную структуру осуществляются по имени функтора.

Примеры объявления структур:

```
DOMAINS
        st = student ( fam, pr, oc)
        fam,pr = symbol
        oc, num = integer
        g = gruppa (num, st)
DOMAINS
name, phone = symbol
birthday = date (integer, symbol, integer)
```

Список - это составной объект, который содержит упоря-

доченный набор произвольного числа других объектов. Эти объекты - члены списка и должны принадлежать одному и тому же доменному типу. Объектами списка могут быть: целые числа, действительные числа, символы, символьные строки, структуры, списки.

Совокупность элементов списка заключается в квадратные скобки, элементы отделяются друг от друга запятыми. Список, не содержащий элементов, называется пустым списком и обозначается [ ].

Объявление списка в разделе DOMAIN имеет следующий формат:

< имя списка >=< тип элемента >*

Например:

DOMAINS
elemlist = symbol*
или
elemlist = element*
element = symbol

Работающий со списками предикат в разделе PREDICATES описывается, например, так:

PREDICATES
writelist (elemlist)

Непосредственно сам список записывается в разделах GOAL или CLAUSES.

Например:

CLAUSES
writelist (["Иванов С.П.", "Сидоров К.Н.", "Петров Р.А."]).

В разделе **PREDICATES** описываются используемые в программе предикаты. Описание предиката содержит имя предиката и список доменов его аргументов:

<имя предиката>(<**домен** 1 >**,**< **домен 2>,…, <домен** N >)

Имя предиката должно начинаться со строчной латинской буквы, за которой могут следовать цифры, знаки подчеркивания, латинские буквы.

Предикат с одним и тем же именем может иметь различное число аргументов. Это будут разные предикаты и они объявляются отдельно. Могут использоваться предикаты без аргументов (нульместные предикаты).

Пример объявления предикатов:

```
DOMAINS
        name, phone = symbol
        birthday = date (integer, symbol, integer)
PREDICATES
person (name, phone, birthday)
```

В разделе **CLAUSES** описываются *утверждения,* каждое из которых является фактом или *правилом.* В конце каждого утверждения ставится точка.

Правило в языке Пролог - формула логики предикатов:

$$P1(\overline{x}) \& P2(\overline{x}) \& ... \& Pm(\overline{x}) \rightarrow Q(\overline{x}),$$

где $P1(\overline{x})$, $P2(\overline{x})$,..., $Pm(\overline{x})$, $Q(\overline{x})$ - атомарные формулы - n-местные предикаты,

$\overline{x}$ = | xl, x2,..., xn | — вектор-строка аргументов предиката.

Эта формула на языке Пролог записывается в виде:

$$Q(\overline{x}) :- P1(\overline{x}), P2(\overline{x}), ..., Pm(\overline{x}).$$

Предикат $Q(\overline{x})$ называется *заголовком* правила, а предикаты $P1(\overline{x})$, $P2(\overline{x})$, ..., $Pm(\overline{x})$ - *телом* правила.

Правило читается так: $Q(\overline{x})$ следует из $P1(\overline{x})$ и $P2(\overline{x})$ и ... и $Pm(\overline{x})$.

Предикаты правил в качестве аргументов могут содержать переменные. Имена переменных (аргументы предикатов и функций) должны начинаться с *прописных* букв.

В качестве имени переменной в предикатах может исполь-

зоваться знак подчеркивания. Такая переменная называется ***анонимной***. Она используется в том случае, если ее значение безразлично для Пролог-программы. Значение анонимной переменной вычисляется интерпретатором Пролога, но не выводится.

Вместо двоеточия с дефисом (:-) в правилах можно использовать ключевое слово *if,* вместо запятой - ключевое слово *and*, вместо точки с запятой ключевое слово *or.*

Факт на языке Пролог соответствует предикату с аргументами- константами и записывается в виде P(a1, a2, ..., an), где a1, a2,..., an— константы.

Факт можно рассматривать как частный случай правила, представляющего собой истинное высказывание.

Пример записи фактов и правил приведен в программе *№* 1.

ПРОГРАММА №1

```
DOMAINS
        person = symbol
PREDICATES
        is_a_citizen ( person)              % есть гражданин
  (личность)
        right_income ( person)              % нормальный до-
  ход (личность)
        has_2_kids (person)                 % имеет двух детей
  (личность)
        has_kids ( person,integer )         % имеет детей (
  личность, целые )
        married (person, person)                    % женат
  (личность, личность)
        makes_bucks (person, integer )      % зарабатывает
  (личность, целые )
        average_taxpayer (person)           % средний налого-
  плательщик (личность)

CLAUSES
        is_a_citizen (albert).
        is_a_citizen ( mark ).
        is_a_citizen (anna).
        has_kids ( mark, 2 ).
        has_2_kids (Person) :- has_kids (Person, X), X = 2.
        married ( mark, anna).
        makes_bucks ( albert, 150).
```

```
        makes_bucks ( mark, 1500 ).
        right_income (Person) :- makes_bucks ( Person, Z),
                                 Z >= 500, Z <= 2000.
average_ taxpayer (Person) :- right_income (Person),
                              married ( Person, _),
                              has_2_kids ( Person),
                              is_a_citizen ( Person ),
                             write ( " average tax-
                          payer ", Person ).
```

В теле правила, кроме объявленных в программе предикатов, могут испольэоваться ***стандартные*** предикаты и ***операции сравнения***.

В приведенном примере стандартным предикатом является предикат вывода **write** (" **average taxpayer** ", **Person**).

Стандартные предикаты выполняют разнообразные функции по вводу-выводу, работе с файлами, обработке строк и т.д.

В правилах можно использовать ***операции сравнения***:

< (меньше), <= (меньше или равно), > (больше), >= (больше или равно), = (равно), <> (не равно). Сравнивать между собой можно выражения и переменные.

Кроме основных (обязательных) разделов в Пролог - программе могут быть разделы **CONSTANTS** и **DATABASE**.

В разделе CONSTANTS можно объявлять и затем использовать символические константы, например:

```
CONSTANTS
zero = 0
one = 1
two =2
pi = 3.141592
four = 4
```

В разделе DATABASE описываются предикаты динамической базы данных. Указанные в этом разделе предикаты после подстановки в них вместо переменных констант (т.е. превращения их в факты) могут быть помещены и, если потребуется, удалены во время выполнения программы в динамическую базу данных с помощью встроенных стандартных предикатов asserta, assertz, retract, retractall.

В программе может быть несколько разделов DOMAINS, PREDICATES, CLAUSES. В этом случае необходимо выполнить сле-

дующие ограничения:

- константы, объекты и предикаты должны объявляться до их исполнения;
- программа может содержать только один раздел GOAL (цель);
- все предложения, определяющие один и тоn же предикат, должны следовать подряд.

В разделе GOAL указываются внутренние запросы к программе, которые записываются в *форме факта* или в *форме вопроса.* Например, запрос в форме факта для программы № 1 может выглядеть так:

```
GOAL
married ( mark, anna ).
```

Если такой факт есть в разделе CLAUSES, то ответом системы будет YES.

Если факт не описан в разделе CLAUSES, то на запрос (программа №1)

```
GOAL
makes_bucks (robert, 1000 )
```

система ответит NO.

Ответом системы на запрос, который может быть выведен из фактов и правил раздела CLAUSES, также будет YES, например:

```
GOAL
average_taxpayer (mark).
```

Запросы в форме вопроса записываются так:

```
GOAL
average_taxpayer (Person).
```

Пролог-система на этот внутренний запрос осуществляет поиск только одного первого подходящего решения. При этом система не сообщает о результатах найденного решения. Отображение на экран найденного значения переменной Person необходимо запрограммировать с помощью встроенных стандартных предикатов вывода.

В частности, в программе № 1 в последнем правиле, помимо других предикатов, для этой цели записан стандартный предикат

write (Person, " is an average taxpayer ").

В результате на экран дисплея будет выведено сообщение:

mark is an average taxpayer.

Раздел GOAL может отсутствовать в Пролог-программе. В этом случае, после запуска программы на выполнение, в диалоговом окне появляется требование на внешний запрос: **goal.** При использовании внешнего запроса Пролог отыскивает все варианты решения. В этом же диалоговом окне выводятся значения переменных запроса.

Так, например, на внешний запрос

goal: average_taxpayer (Person)

Пролог-программа ответит:

Person = mark

Управление вычислениями в Пролог-программе

Процедуру поиска ответа на запросы, сформулированные в самой **Пролог-программе** в разделе GOAL (внутренний запрос) или в диалоговом окне (внешний запрос), рассмотрим на примере программы № 2 "Средний налогоплательщик ".

ПРОГРАММА №2

```
DOMAINS
person = symbol
PREDICATES
average_taxpayer (person)          % средний налогоплательщик ( личность )
is_a_citizen ( person )            % есть гражданин ( личность )
married (person, person )                  % женат( личность, личность )
has_2_kids (person, integer)       % имеет двух детей ( личность, целые )
has_kids ( person, integer )       % имеет детей ( личность, целые)
makes_bucks ( person, integer) % зарабатывает (личность,
```

целые) right_income (person, integer) % иметь нормальный доход (личность,целые)

```
CLAUSES
is_a_citizen (tom ).
is_a_citizen ( albert).
is_a_citizen ( mark ).
is_a_citizen ( anna ).
is_a_citizen (chris).
married (tom, chris ).
married ( mark, anna).
has_kids (tom, 1 ).
has_kids ( albert, 2 ).
has_kids ( anna, 2 ).
has_ 2_ kids ( Person, X ) :- married ( Person, _), has_kids ( X ), X = 2.
makes_bucks (tom, 500 ).
makes_bucks ( albert, 2000 ).
makes_bucks ( mark, 1500 ).
makes_bucks ( anna, 0 ).
right_income ( Person, N) :- makes_bucks ( Person, N ),
                             N >= 1000,
                                N <= 2000.
average_taxpayer ( Person ) :- is_a_citizen (Person),
                                married ( Person, _),
                                right_income ( Person, _ ),
                                has_2_kids ( Person, _ ).
GOAL
average_taxpayer ( Person),
write ( Person," is an average taxpayer ").
```

При поиске решения задачи, поставленной в разделе GOAL логической программы, Пролог использует метод проб и возвращений назад, называемый "поиском с возвращением".

Поиск начинается с сопоставления цели с утверждениями программы (фактами и заголовками правил). При этом интерпретатор Пролога просматривает утверждения программы сверху вниз в порядке следования и выбирает первое утверждение, для которого сопоставление успешно. Цель average_taxpayer (Person) успешно сопоставлена с заголовком правила average_taxpayer (Person), поэтому заголовок правила (исходная цель) заменяется подцелями в теле правила:

is_a_citizen (Person), married (Person, _), right_income (Per-

son, _), has_2_kids (Person, _).

Теперь Пролог последовательно пытается достичь этих целей.

Первая подцель is_a_citizen (Person) сопоставляется с первым фактом, записанным в разделе CLAUSES, is_a_citizen (tom).

В результате переменная Person получает значение tom.

Вторая подцель married (tom, _) сопоставляется с фактом married (tom, chris). Это сопоставление успешно.

Третья подцель right_income (tom, _) требует привлечения правила right_income (Person, N) :- makes_bucks (Person, N), N>=1000, N<=2000.

Сопоставление цели с заголовком этого правила будет успешно, если подцели тела правила

makes_bucks (Person, N), N >= 1000, N <= 2000

сопоставимы с фактами из раздела CLAUSES.

Очевидно, эта подцель не сопоставима с фактом makes_bucks (tom, 500). Поэтому Пролог возвращается к точке ветвления программы (точке отката) и попробует проверить альтернативный вариант.

Первая подцель: is_a_citizen (albert).

В этом случае переменная Person = albert.

Вторая подцель: married (albert, _)

не находит подтверждения, так как этого факта нет в программе. Поэтому вновь происходит откат к точке ветвления. Пролог-программа рассматривает третий альтернативный вариант.

Первая подцель: is_a_citizen (mark).

Переменная получает значение Person = mark.

Вторая подцель: married (mark, _) сопоставляется с фактом married (mark, anna).

Сопоставление *третьей подцели* right_income (mark, _) с фактами программы также успешно, так как в программе есть факт makes bucks (mark, 1500).

Таким образом, цель average_taxpayer (Person) успешна для Person = mark.

Вторая цель, указанная в разделе GOAL, представляет собой встроенный стандартный предикат вывода на дисплей

write (Person,″ is an average taxpayer ″).

В диалоговом окне будет выведено сообщение:

mark is an average taxpayer

Задания на лабораторную работу

Вариант А

Исходные данные. Известна родословная библейской семьи (рис.1):

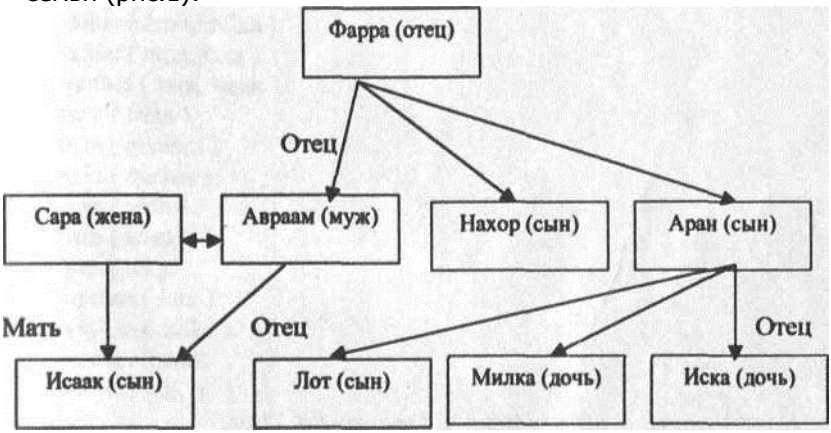


Рис.1

Пример базы знаний "Родственные отношения в библейской семье":

ПРОГРАММА №3

```
DOMAINS
name =
symbol
PREDICA
TES
        father
    (name, name)
        man (name)
        mother ( name,  name )
    woman (name)
        married (name, name)
        son (name, name )
        daughter ( name, name)
        brother ( name, name)
        sister (name, name )
        grandfather ( name, name )
        grandson (name, name )
        granddaughter ( name, name)
        kids (name, name )
```

```
 CLAUSES
        father ( farra, av-
raam).
        father ( farra, nachor).
        father ( farra, aron ).
        father ( avraam,
isaak).
        father (aron, lot).
        father (aron, milka ).
        father (aron, iska ).
         mother ( sara, isaak).
        man (farra).
        man (avraam).
        man (nachor).
        man (aron).
        man (isaak).
        man (lot).
        woman (sara).
        woman (milka).
        woman (iska).
        married ( avraam, sara ).
        son  (X, Y ) :-
        father (Y, X ); mother ( Y, X ), man ( X ).
 daughter (X, Y ) :-
        woman ( X ),  father (X, Y ).
 brother ( X, Y ) :-
        father (Z, X ), father (Z, Y ), man ( X ).
 sister ( X, Y ) :-
        father (Z, X), father ( Z, Y ), woman ( X ).
 grandfather ( X, Y ):-
        father ( X , Z ),  father(Z,Y).
 grandson (X, Y ) :-
        man ( X), father ( Y, Z), father ( Z, X ).
 granddaughter (X, Y ) : -
        woman ( X ),  father (Y, Z), father (Z, X).
        kids (X, Y) :-
        father ( Y , X );  mather (Y, X).
```

Задание:

а) сформулировать следующие запросы к системе:

1) кто сыновья Фарры?
2) кто дети Арона?
3) есть ли невестка у Фарры, если есть, то кто она?

4) назовите всех внуков Фарры;
5) кто дед Милки?
6) кто братья Авраама?
7) кто сестры Лота?
8) есть ли сестра у Иски?
9) назовите внуков Фарры;

б) описать процессы формирования ответов на указанные выше вопросы при внешних и внутренних запросах.

Вариант Б

Исходные данные. В библиотеке имеются следующие книги:

1)Шолохов М. А. 'Тихий Дон", 736 с.
2)Шолохов М. А. "Поднятая целина", 475 с.
3)Шолохов М. А. "Они сражались за Родину", 387 с.
4) Толстой Л1. Н. "Анна Каренина", 702 с.
5) Куприн А. И. "Поединок", 180 с.

Пример базы данных "Каталог книг":

ПРОГРАММА № 4

```
DOMAINS
title, autor = symbol
pages = integer
PREDICATES
        book (title, pages)
        written_by (autor, title)
        long_novel (title)
CLAUSES
      written_by ( "Шолохов", "Тихий Дон" ).
      written_by ("Шолохов М.А.", "Поднятая целина" ).
      written_ by ("Шолохов М.А.", "Они сражались за
Родину" ). written_by ("Толстой Л.Н.", "Анна Каренина" ).
      written_by ("Куприн А.И.", "Поединок" ).
      book ( "Тихий Дон", 736 ).
      book ("Поднятая целина", 475 ).
      book ("Они сражались за Родину", 388 ).
       book ("Поединок", 180).
      long_novel (Title ) :- written_by (_, Title), book (Ti-
tle, Length ),
                                  Length > 400.
```

***Задание*:**

а) составить базу данных об этих книгах на языке Пролог, указав автора, название, число страниц;

б) сформулировать следующие внутренние и внешние запросы к системе:

1) кто автор "Поединка"?
2) какие книги М. А. Шолохова есть в библиотеке?
3) какие книги относятся к "длинным" романам?
4) сколько страниц имеет книга "Анна Каренина"?

в) описать процессы формирования ответов на указанные вопросы при внутренних и внешних запросах.

Содержание отчета

Отчет по лабораторной работе должен содержать:

а) название и цель выполнения работы,
б) исходные данные,
в) программу на языке программирования Пролог,
г) описание управления процессами вычислений при внутренних и внешних запросах.

Контрольные вопросы

1. Какие разделы содержит программа на Прологе? Каково назначение каждого раздела?
2. Что представляет собой предикат?
3. Чем факты отличаются от правил?
4. Как интерпретатор Пролога ищет ответ на запрос?
5. Чем внешние запросы отличаются от внутренних?

ЛАБОРАТОРНАЯ РАБОТА НА ТЕМУ: « ПРОГРАММИРОВАНИЕ НА ПРОЛОГЕ БАЗЫ ДАННЫХ СО СТРУКТУРИРОВАННЫМИ ОБЪЕКТАМИ»

Цель работы: получить навыки программирования баз данных со структурированными объектами на языке логического программирования Пролог.

Задания на лабораторную работу

Вариант А

Исходные данные. Известна информация о результатах сдачи семестровых экзаменов студентами 4-го курса факультета «Автоматика и робототехника».

Учебная группа Р4-1

№	Фамилия, имя, отчество	МОУ	ТАУ	Информатика
1	Артамонов В.А.	4	5	4
2	Волошенко А.А.	4	3	3
3	Ивахненко М.А.	4	3	4
4	Камышанцев А.В.	3	3	4
5	Кудрявцева Л.Е.	5	5	5

Группа Э4-1

№	Фамилия, имя, отче-ство	ТАУ	Электротехника	Электропривод
1	Кузнецов И.О.	3	3	4
2	Литвинов Д.А.	5	5	5
3	Лобанова О.С.	4	5	3

Пример базы данных «Семестровая успеваемость студентов 4-го курса факультета «Автоматика и телемеханика».

ПРОГРАММА №5

```
DOMAINS
st = student (fam, pr, oc )          % структура с функтором
```

```
student
        fam, pr = symbol                % и доменами fam,pr,oc oc
= integer
        g = gr(num,st)                  % структура с функтором gr
        num = symbol                    % и доменами num, st
        PREDICATES
        kurs_4_AiR ( g)                 % одноместный предикат,
аргументами
                                        % которого является структу-
                                ра доменов
        CLAUSES                 % утверждения-факты
        kurs_4_AiR (gr(r4_1, student( "Артамонов В. А.", "МОУ", 4 ))).
        kurs_4_AiR (gr (r4_l, student ("Волошенко А.А.", "МОУ", 4 ))).
        kurs_4_AiR (gr (r4_l, student( "Ивахненко М.А.", "МОУ", 4 ))).
        kurs_4_AiR (gr(r4_l, student ("Камышанцев А. В.", "МОУ", 3
))).
        kurs_4_AiR (gr (r4_l, student ( "Кудрявцева Л.Е.", "МОУ", 5
))).
        kurs_4_AiR (gr (r4_l, student ("Артамонов В.А.", "ТАУ", 5 ))).
        kurs_4_AiR (gr ( r4_1, student ( "Волошенко А. А.", "ТАУ", 3
))).
        kurs_4_AiR (gr (r4_l, student ( "Ивахненко М..А.", "ТАУ", 3 ))).
        kurs_4_AiR (gr (r4_1, student ( "Камышанцев А. В.", "ТАУ", 3
))).
        kurs_4_AiR ( gr ( r4_l, student ("Кудрявцева Л.Е.", "ТАУ", 5
))).
        kurs_4_AiR (gr (r4_l, student ("Артамонов В.Д.", "Информати-
ка", 4 ))). kurs_4_AiR (gr (r4_l, student ("Волошенко А. А.", "Ин-
форматика", 3 ))). kurs_4_AiR (gr (r4_l, student ("Ивахненко М.А.",
"Информатика", 4 ))). kurs_4_AiR (gr(r4_l, student("Камышанцев
А.В.","Информатика", 4 ))). kurs_4_AiR (gr ( r4_l, student ( "Куд-
рявцева Л.Е.", "Информатика", 5 ))). kurs_4_AiR (gr ( e4_l, student
("Кузнецов И.О.", "ТАУ", 3 ))).
        kurs_4_AiR ( gr (e4_1, student ( "Литвинов Д.Д.", "ТАУ", 5 ))).
        kurs_4_AiR ( gr ( e4_1, student ( "Лобанова О.С.", "ТАУ", 4
))).
        kurs_4_AiR (gr ( e4_1, student ("Кузнецов И.О.", "Электро-
техника", 3 ))). kurs_4_AiR ( gr (e4_1, student ("Литвинов Д.А.",
"Электротехника", 5 ))). kurs_4_AiR ( gr (e4_1, student ("Лобанова
О.С.", "Электротехника", 5 ))). kurs_4_AiR (gr (e4_1, student ("Куз-
нецов И.О.", "Электропривод", 4 ))). kurs_4_AiR (gr (e4_1, student (
"Литвинов Д. А.", "Электропривод", 5 ))).
```

kurs_4_AiR (gr (e4_1, student ("Лобанова О.С.", "Электропривод", 3))).

Управление вычислениями в программе № 5

Программа в разделе CLAUSES содержит утверждения-факты. Решение задачи зависит от сформулированных запросов.

Рассмотрим вначале решение задачи при внешних запросах, задаваемых в диалоговом окне интерфейса программы. После компиляции программы система затребует внешний запрос goal.

Если ввести внешний запрос

kurs_4_AiR (X),

то будет выведена вся информация о результатах сдачи семестровых экзаменов студентами 4-го курса факультета АиР, хранящаяся в базе данных:

X = gr (r4_1 (student ("Артамонов В.А.", "МОУ", 4))
X = gr (r4_1 (student ("Волошенко А.А. "МОУ", 4))
.....
X = gr (r4_l (student("Лобанова О.СЛ "Электропривод", 3))
X = gr (r4_l (student ("Литвинов Д.АЛ "Электропривод", 5))
24 Solutions % 24 решения

В данном случае на структуру g = gr (num, st) не наложено никаких ограничений.

Если необходимо узнать фамилии студентов, получивших на экзамене отличные оценки, то внешний запрос формулируется так:

kurs_4_AiR (gr (_, student (X, _, 5))).

Символы подчеркивания, указанные в запросе, соответствуют ***анонимным переменным.*** Анонимные переменные используются в процессе сопоставления так же, как и любые другие переменные, но значения, присвоенные им в процессе сопоставления, не сохраняются для дальнейшей обработки и на экран дисплея не выводятся.

На этот запрос в диалоговом окне будет выведена инфор-

мация:

X = Кудрявцева Л. Е.
X = Артамонов В. А.
X = Кудрявцева Л. Е.
X = Кудрявцева Л. Е.
X = Литвинов Д. А.
X = Литвинов Д. А.
X = Литвинов Д. А.
X = Лобанова О. С.
7 Solutions

Если необходимо узнать результаты сдачи экзаменов студентом Волошенко А. А., то внешний запрос формируется так:

kurs_4_AiR (gr (r4_l, student ("Волошенко А. А.", X, Y))).

Ответом на этот запрос будет

X = МОУ, Y = 4
X = ТАУ, Y = 3
X = Информатика, Y = 3
3 Solutions

Если формулируется запрос в самой программе в разделе GOAL, то необходимо указать предикаты вывода значений решения.

Например:
GOAL
kurs_4_AiR(X),write(X).

Здесь для вывода значений переменной X используется стандартный предикат **write (...)**. Результатом такого запроса будет единственное подходящее решение:

gr (r4_l, student ("Артамонов В.А.", "МОУ", 4)).

Пролог чувствителен к порядку утверждений-фактов в программе: они просматриваются сверху вниз. Чтобы осуществить поиск всех альтернативных вариантов решения задачи при использовании внутреннего запроса, необходимо организовать циклический перебор всех вариантов.

Одним из способов организации циклических вычислений является использование специального стандартного предиката fail.

Этот предикат всегда завершается неуспехом и вызывает "откат" программы до ближайшей альтернативы. Для перебора всех альтернативных фактов базы данных и вывода их на экран дисплея введем в состав программы нульместный предикат **zikl** и правило вида

```
zikl :-
kurs_4_AiR ( X ), write ( X ), fail.
```

Для этого в разделе PREDICATES добавляем строку **zikl,** а в разделе CLAUSES - указанное выше правило.

Внутренний запрос в разделе GOAL формируем так:

```
GOAL
zikl.
```

В этом случае на экран будут выведены все 24 альтернативных варианта решения.

После нахождения первого решения

X = *gr* (r4_l, student ("Артамонов В.А.", "МОУ", 4))

полученное значение переменной X будет присвоено аргументу предиката write (X), который выведет его на экран. После этого выполняется заведомо ложный предикат **fail.** В результате программа осуществит "откат" до ближайшего предиката, имеющего альтернативные варианты:

kurs_4_AiR (X),

и выбирает второй факт:

X = gr (r4_l, student ("Волошенко А.А.", "МОУ", 4)).

Процесс перебора вариантов продолжается до тех пор, пока не будут выбраны все альтернативные варианты.

Задание:

а) сформулировать следующие внешние и внутренние запросы к системе:

1) результаты сдачи экзаменов студентами группы Р4-1;

2) результаты сдачи экзаменов студентами группы Э4-1;

3) фамилии студентов, получивших во время сессии

удовлетворительные оценки;

4) результаты сдачи студентами обеих групп экзамена по ТАУ;

5) результаты сдачи экзаменов студенткой Лобановой О.С.;

б) описать процессы формирования ответов на указанные запросы.

Вариант Б

Исходные данные. В отделе социальной защиты для выплаты детских пособий собрана информация о составе и доходах семей.

1. Семья Конюховых: муж - Конюхов Иван, дата рождения — 5 мая 1950 года, инженер, зарплата 13500 рублей;

жена - Конюхова Наталья, дата рождения - 25 ноября 1950 года, врач, зарплата - 8000 рублей;

сын - Конюхов Юрий, дата рождения - 5 июня 1974 года, не работает, доходов нет;

дочь - Конюхова Татьяна, дата рождения - 15 мая 1978 года, студентка, стипендия - 2500 рублей.

2. Семья Авенсов:

муж - Авенс Ефим, дата рождения - 14 сентября 1947 года, председатель совета директоров коммерческого банка, декларируемый годовой доход 13000 долларов;

жена - Авенс Илона, дата рождения - 12 августа 1977 года, не работает, доходов нет;

дочь - Авенс Лора, дата рождения - 2 января 1999 года, младенец, пособие на ребенка - 500 рублей.

3.|Семья Петровых:

муж - Петров Сергей, дата рождения - 4 июня 1967 год, слесарь, зарплата 10000 рублей;

жена Петрова Людмила, дата рождения - 17 октября 1968 года, не работает,

доходов нет;

сын - Петров Илья, дата рождения - 14 июля 1989 года, учащийся, пособие на ребенка - 500 рублей;

дочь - Петрова Юлия, дата рождения - 27 апреля 1991 года, учащийся, пособие на ребенка - 500 рублей;

дочь Петрова Марина, дата рождения - 20 мая 1995 года, младенец, пособие на ребенка - 500 рублей.

Особенности программирования с использованием структур

В базе данных семья может описываться одним предложением. Каждая семья состоит из трех компонент: мужа, жены и детей.

Так как количество детей в семьях может быть разным, то их целесообразно представить в виде списка, состоящего из произвольного числа **эле**ментов. Каждого члена семьи в свою очередь можно представить

структурой, состоящей, например, из четырех компонент: имени, фамилии, даты рождения и работы. Информация о работе - должность и оклад.

Для неработающих совершеннолетних членов семьи указывается "не работает" и, если нет дополнительного дохода, то проставляется 0. Для детей (учащихся и студентов) указывается "учащийся" или " студент " и в качестве дохода указывается размер детского пособия или стипендии.

Для детей моложе 6 лег указывается "младенец".

Информацию, например, о семье Конюховых, можно занести в базу д а н н ы х с помощью одного предложения (clause):

family (member_family
("Иван","Конюхов", date (5, "май", 1950), work ("инженер", 13500)), member_family ("Наталья","Конюхов", date (25, "ноябрь", 1950),
work ("врач", 8000)), |
[member_family ("Юрий", "Конюхов", date (5, "июнь", 1974),
work ("не работает", 0)),
member_family ("Татьяна", "Конюхов", date (15, "май", 1978), work ("студентка", 2500))]|).

В базу данных заносятся факты-предложения (clauses), подобные вышеприведенному, о всех семьях, представляющих интерес для отдела социальной защиты.

Пример базы данных "Состав и доходы семей" ПРОГРАММА №6

DOMAINS
/* В данной программе в качестве доменов используются структуры, аргументами которых являются простые базисные домены */

```
Mf = member_family (symbol, symbol, dt, w)
dt = dale (integer, symbol, integer)
w = work( symbol, integer)
```

/* В качестве домена используется составной объект - *список*, который содержит упорядоченную последовательность произвольного числа других объектов, принадлежащих одному и тому же доменному типу - *структурам* */

```
mflist =  mf*
PREDICATES
family ( mf, mf, mflist)     /* семья ( член семьи, член семьи,
                                    список членов семьи */
husband (mf )                 /* муж ( член семьи ) */
wife( mf)                          /* жена ( член семьи ) */
child (mf)                         /* ребенок ( член семьи ) */
belong ( mf, mflist)        /* принадлежит (член семьи, список
членов семьи) */
exist (mf)                         /* существует ( член семьи )
*/
date_birth (mf, dt)         /* дата рождения (член семьи, да-
та) */
income (mf, integer)        /* доход ( член семьи, целые )*/

CLAUSES
family (member_family ("Иван", "Конюхов", date ( 7, "май",
1950),
                          work( "инженер", 13500)),
                                member_family     ("Наталья",
                           "Конюхов",    date ( 25, "ноябрь",
                           1950), work ( "врач", 8000)),
     [ member _family ("Юрий", "Конюхов", date ( 5, "июнь",
  1974 ),
          work ("не работает", 0)),
                member_family ('Татьяна", "Конюхов", date (
            15, "май", 1978 ), work ("студентка", 2500 ))]).
family( member_family ( "Ефим", "Авенс", date ( 14, "сен-
тябрь", 1947 ),
          work ("банкир", 390000)),
          member_family( "Илона", "Авенс", date (12, "август",
       1977),
                work ("не работает", 0)),
                [member_family( "Лора", "Авенс", date (2,
            "январь", 1977 ), work ("младенец", 500))]).
```

family(member_family ("Сергей", "Петров", date (4, "июнь", l967),
work ("слесарь", 10000)),
member_family ("Людмила", "Петров", date (17, "октябрь", 1968),
work ("не работает", 0)),
[member_family ("Илья", "Петров", date (14, "июль", 1989),
work ("учащийся", 500)),
member_family ("Юлия", "Петров", date (27, "апрель", 1991),
work ("учащийся", 500)),
member_family ("Марина", "Петров", date (20, "май", 1995),
work ("младенец", 500))]).

Управление вычислениями в программе № 6

База данных, представленная в виде структур, очень удобна для извлечения необходимой информации. В такой базе можно ссылаться на объекты, не указывая в деталях всех компонент, и задавать только структуру интересующих нас объектов. Применяя ***анонимные переменные***, можно не описывать конкретные объекты, а ограничиться лишь частичным описанием.

Например, для того, чтобы получить информацию о списке всех семей, находящихся в базе данных, достаточно в диалоговом режиме сформулировать запрос:

family (member_family (_, X, _), _, _).

На этот запрос Пролог-программа выдаст решение:

X=Конюхов
X=Авенс
X=Петров
3 Solutions

Для получения информации о неработающих женщинах достаточно в диалоговом окне сформулировать запрос:

family (_, member_family (X, Y, work ("не работает",)), _).

Огвет на этот запрос:

X=Илона У=Авенс

X=Людмила У=Петров
2 Solutions

Необходимо обратить внимание на то, что интересующие нас объекты можно указывать не только по их содержанию, но и по структуре.

Для того, чтобы взаимодействие с базой данных было более удобным, необходимо создать набор правил, который служил бы утилитой и составлял часть пользовательского интерфейса.

Ниже приводится набор правил, с помощью которых можно извлечь из данной базы данных полезную информацию. Эти правила должны быть добавлены к программе № 6. Используемые в этих правилах предикаты описаны в разделе PREDICATES.

```
husband ( X ) :- family (X, _, _).  /*  определяется  понятие «муж» */
wife ( X ) :- family (_, X, _ ).                /* определяется понятие «жена» */
child (X ) :-
family ( Kids ), belong(X, Kids).     /* определяется понятие «дети» */
belong ( X, [X|_]).        /* Предикат belong (...) и рекурсивное */
belong ( X, [ _ | L ]) :-
 belong(X,L).       /*правило определяют принадлежность*/
                              /*элемента X к списку [....].    * /
exist ( Member_family ) :-
                    husband  (  Member_family  );   /*определяет
              существование */
                    wife ( Member_family);              /*любого
              члена семьи в базе */
             child (Member_famiiy).            /*       знаний */
data_birth  (  member_family(_,  _,Date,  _),  Date  ).  /*  дата рождения члена*/
                                                          /*семь
                                                        и */
income ( member_family(_, _, _, work ( _, S )), S ). /* доход члена семьи */
```

С помощью этих правил, обращаясь к системе с соответствующими запросами, можно получить ответы, например, на такие вопросы:

I) Найти имена и фамилии всех детей моложе 7 лет.

Goal: child(X), date_birth (X, date (_, _, Y)), Y >= 2007.

Задание:

а) сформулировать следующие внутренние и внешние запросы к системе :

1) найти фамилии женщин, имеющих по крайней мере двух детей;

2) есть ли в базе данных семьи, не имеющие детей;

3) найти в базе данных людей старше 50 лет;

4) найти всех детей, разница в возрасте родителей которых составим не менее 15 лет;

5) найти в базе данных всех учащихся;

б) описать процессы формирования ответов на указанные запросы.

Контрольные вопросы:

1. Какие структурированные объекты используются в языке Пролог?
2. Как представляются структурированные объекты на Прологе?
3. Для чего используется функтор?
4. Когда применяется анонимная переменная?

ЛАБОРАТОРНАЯ РАБОТА НА ТЕМУ: «ПРЕДСТАВЛЕНИЕ СПИСОЧНЫХ ОБЪЕКТОВ НА ПРОЛОГЕ»

Цель работы: получение навыков представления и оперирования со списочными структурами на Прологе.

Теоретические сведения

Список - это простая структура данных, широко используемая в нечисловом программировании. Список — это последовательность, составленная из произвольного числа элементов, например, **анна, теннис, ольга, лыжи**. На Прологе это записывается так:

[анна, теннис, ольга, лыжи]

Однако таково лишь внешнее представление списков. Как и все структурные объекты Пролога списки —это деревья.

Пустой список обозначается квадратными скобками: []. Непустой список рассматривается как структура, состоящая из двух частей:

(1) первый элемент, называемый ***головой*** списка;
(2)остальная часть списка, называемая ***хвостом***.

Например, для списка

[анна, теннис, ольга, лыжи]

анна —это голова, а хвостом является список

[теннис, ольга, лыжи]

В общем случае, головой может быть что угодно (любой объект Пролога, например, дерево или переменная); хвост же должен быть списком. Голова соединяется с хвостом при помощи специального функтора. Выбор этого функтора зависит от конкретной реализации Пролога; мы будем считать, что это точка:

.(Голова, Хвост)

Поскольку **Хвост —** это список, он либо пуст, либо имеет собственную голову и хвост. Наш список представляется следующим образом:

.(анна, .(теннис, .(ольга, .(лыжи, [])))))

На рис. 1 изображена соответствующая древовидная структура. Заметим, что показанный выше пример содержит пустой список []. Дело в том, что самый последний хвост является одно-

элементным списком:

[лыжи]

Хвост этого списка пуст

[лыжи] = .(лыжи, [])

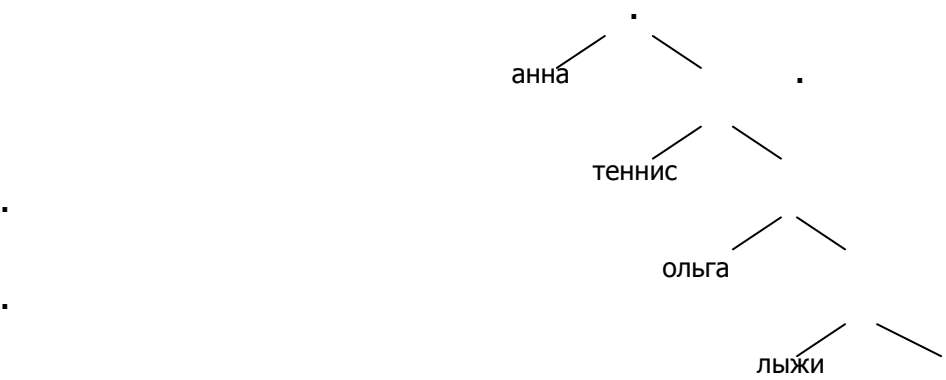


Рис. 1. Представление списка [анна, теннис, ольга, лыжи] в виде дерева.

На практике часто бывает удобным трактовать хвост списка как самостоятельный объект. Для этого в Прологе используется вертикальная черта, отделяющая голову от хвоста. Например, список **L = [a, b, c]** можно представить, отделив голову от хвоста с помощью вертикальной черты:

L = [a | Хвост]

На самом деле вертикальная черта имеет более общий смысл: мы можем перечислить любое количество элементов списка, затем поставить символ "|", а после этого — список остальных элементов. Так, только что рассмотренный пример можно представить следующими различными способами:

[a, b, c] = [a | [b, c]] = [a, b | [c]] = [a, b, c | []]

Выводы:

• Список — это структура данных, которая либо пуста, либо состоит из двух частей: ***головы*** и ***хвоста***. Хвост, в свою очередь, сам является списком.

• Список рассматривается в Прологе как специальный частный случай двоичного дерева. Для повышения наглядности программ в Прологе предусматриваются специальные средства для списковой нотации, позволяющие представлять списки в виде

[Элементl, Элемент2, ...]

или

[Голова | Хвост]

или

[Элемент1, Элемент2, ... | Остальные]

Операции над списками

Списки можно применять для представления множеств, хотя и существует некоторое различие между этими понятиями: порядок элементов множества не существенен, в то время как для списка этот порядок имеет значение; кроме того, один и тот же объект может встретиться в списке несколько раз. Однако наиболее часто используемые операции над списками аналогичны операциям над множествами. Среди них

- проверка, является ли некоторый объект элементом списка, что соответствует проверке объекта на принадлежность множеству;
- конкатенация (сцепление) двух списков, что соответствует объединению множеств;
- добавление нового объекта в список или удаление некоторого объекта из него.

Покажем программы, реализующие эти операции над списками.

Принадлежность к списку

Представим отношение принадлежности как

belong(X, L)

где **X —** объект, а **L —** список. Цель **belong(X, L)** истинна, если элемент **X** встречается в **L**. Например, верно что

belong (b, [a, b, c])

и, наоборот, не верно, что

belong (b, [a, [b,c]])

но

belong ([b, c], [a, [b, c]])

истинно. Составление программы для отношения принадлежности может быть основано на следующих соображениях: X принадлежит списку, если

(1) X есть голова списка, либо

(2) X принадлежит хвосту списка.

Это можно записать в виде двух предложений, первое из

которых есть простой факт, а второе — правило:

belong (X, [X | L]).

belong (X, [Y | L]) :- belong (X, L).

Сцепление (конкатенация)

Для сцепления списков определим отношение

conc (L1, L2, L3)

Здесь **L1** и **L2** — два списка, а **L3** — список, получаемый при их сцеплении. Например,

conc ([a,b], [c,d], [a,b,c,d])

истинно, а

conc ([a,b], [c,d], [a,b,a,c,d])

ложно. Определение отношения **conc**, как и раньше, содержит два случая в зависимости от вида первого аргумента **L1**:

(1) Если первый аргумент пуст, тогда второй и третий аргументы представляют собой один и тот же список (назовем его **L**), что выражается в виде следующего факта:

conc ([], L, L).

(2) Если первый аргумент отношения **conc** не пуст, то он имеет голову и хвост и выглядит так**: [X | L1]**

На рис. 2 показано, как производится сцепление списка **[X | L1]** с произвольным списком **L2**. Результат сцепления — список **[X | L3]**, где **L3** получен после сцепления списков **L1** и **L2**. На Прологе это можно записать следующим образом:

conc ([X | L1], L2, [X | L3]) :— conc (L1, L2, L3).

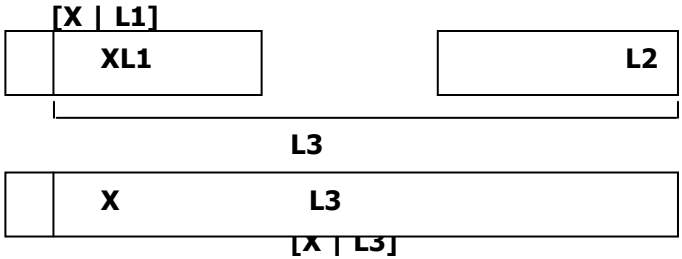


Рис. 2. Конкатенация списков.

Составленную программу можно теперь использовать для сцепления заданных списков, например, при вводе запроса:

conc([a,b,c], [d,e,f], L)

Пролог-система выдаст ответ:

L = **[a,b,c,d,e,f]**

Хотя программа для **conc** выглядит довольно просто, она обладает большой гибкостью и ее можно использовать многими другими способами. Например, ее можно применить как бы в обратном направлении для *разбиения* заданного списка на две части. Так, при вводе запроса:

conc(L1, L2, [a, b, c])

Пролог-система выдаст ответ:

L1 = **[]**
L2 = **[a, b, c];**
L1 = **[a]**
L2 = [b,c];
L1 = [a, b]
L2 = [c];
L1 = [a, b, c]
L2 = [];
no (нет)

Список **[a, b, c]** разбивается на два списка четырьмя способами, и все они были обнаружены нашей программой при помощи механизма автоматического перебора.

Нашу программу можно также применить для поиска в списке комбинации элементов, отвечающей некоторому условию, задаваемому в виде шаблона или образца. Например, можно найти все месяцы, предшествующие данному, и все месяцы, следующие за ним, сформулировав такую цель:

conc(До, [май | После], [янв, фев, март, апр, май, июнь, июль, авг, сент, окт, ноябрь, дек]).

Ответ Пролог-системы:

До = [янв,фев,март,апр]

После = [июнь, июль, авг, сент, окт, ноябрь, дек].

Можно найти месяц, непосредственно предшествующий

маю, и месяц, непосредственно следующий за ним, задав вопрос:

conc(_ , (Месяцl, май, Месяц2 | _], [янв, фев, март, апр, май, июнь,

июль, авг, сент, окт, ноябрь, дек]).

Ответ Пролог-системы:

Месяц1 = апр

Месяц2 = июнь

Можно, например, удалить из некоторого списка **L1** все, что следует за тремя последовательными вхождениями элемента **z** в **L1** вместе с этими тремя **z**. Это можно сделать, задав запрос:

L1 = [a,b,z,z,c,z,z,z,d,e], conc(L2, [z,z,z | _], L1).

Ответ Пролог-системы:

L1 = [a,b,z,z,c,z,z,z,d,e]
L2 = [a,b,z,z,c]

Используя **conc**, можно определить отношение принадлежности следующим эквивалентным способом:

belongl(X, L) : — conc(L1, [X | L2], L).

В этом предложении сказано: «**X** принадлежит **L**, если список **L** можно разбить на два списка таким образом, чтобы элемент **X** являлся головой второго из них. Разумеется, **belongl** определяет то же самое отношение, что и **belong**. Другое имя используется только для того, чтобы различать две разные реализации этого отношения. Заметим, что, используя анонимную переменную, можно записать выше приведенное предложение так:

belongl (X, L) :- conc(_, [X | _], L).

Добавление элемента

Наиболее простой способ добавить элемент в список — это вставить его в начало списка так, чтобы он стал его новой головой. Если **X** — это новый элемент, а список, в который **X** добавляется — **L**, тогда результирующий список **—** это **[X | L]**.

Таким образом, для того, чтобы добавить новый элемент в начало списка, не надо использовать никакой процедуры.

Удаление элемента

Удаление элемента **X** из списка **L** можно запрограммировать в виде отношения

delete (X, L, L1)

где **L1** совпадает со списком **L**, у которого удален элемент **X**. Отношение **delete** можно определить аналогично отношению принадлежности. Имеем снова два случая:

(1) Если **X** является головой списка, тогда результатом удаления будет хвост этого списка.

(2) Если **X** находится в хвосте списка, тогда его нужно удалить оттуда.

```
delete ( X, [ X | L ], L).
delete ( X, [ Y |  L ], [ Y | Ll ] ) :-
delete ( X, L, L1).
```

как и **belong**, отношение **delete** по природе своей недетерминировано. Если в списке встречается несколько вхождений элемента **X**, то **delete** сможет исключить их все при помощи возвратов. Конечно, вычисление по каждой альтернативе будет удалять лишь одно вхождение **X**, оставляя остальные в неприкосновенности. Например, при запросе:

delete (a, [a,b,a,a], L).

ответ Пролог-системы будет:

```
L = [b,a,a];
L = (a,b,a];
L = (a,b,a];
no                                (нет)
```

При попытке исключить элемент, не содержащийся в списке, отношение **delete** потерпит неудачу.

Отношение **delete** можно использовать в обратном направлении для того, чтобы добавлять элементы в список, вставляя их в произвольные места. Например, если мы хотим во все возможные места списка **[1,2,3]** вставить элемент **6**, то мы можем это сделать, задав вопрос: «Каким должен быть список **L**, чтобы после удаления из него элемента **6** получился список **[1,2, 3]**?»:

delete(6, L, [1,2,3]).

Ответ Пролог-системы:

L = **[6,1,2,3];**
L = **[1,6,2,3];**
L **= [1,2,6,3];**
L = [1,2,3,6];
no (нет)

Операция **append** (по внесению **X** в произвольное место некоторого списка **L,** дающее в результате больший список **L1**), может быть определена предложением:

```
append ( X, L, L1) : -
delete ( X, L1, L).
```

Отношение **delete** можно также использовать для проверки на принадлежность. Идея простая: некоторый **X** принадлежит списку **L**, если **X** можно из него удалить:

```
belong2( X, L) : -
delete ( X, L, _ ).
```

Подсписки

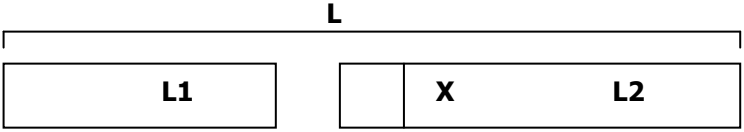
Рассмотрим отношение **sublist** (подсписок). Это отношение имеет два аргумента **—** список **L** и список **S**, такой, что **S** содержится в **L** в качестве подсписка. Так отношение

sublist ([c, d, e], [a, b, c, d, e, f])

имеет место, а отношение

sublist ([c, e], [a, b, c, d, e, f])

нет. Пролог-программа для отношения **sublist** может основываться на той же идее, что **belong**, только на этот раз отношение более общо (см. рис. 3).



long(X, L)

[X | L2]

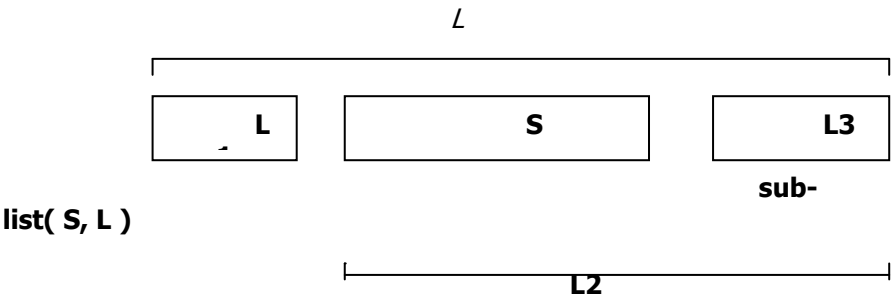


***Рис. 3. Отношения** belong **и** sublist.*

Его можно сформулировать так:** S **является подсписком** L, **если

(1) **L** можно разбить на два списка **L1** и **L2** и

(2) **L2** можно разбить на два списка **S** и **L3**.

Как мы видели раньше, отношение **conc** можно использовать для разбиения списков. Поэтому вышеприведенную формулировку можно выразить на Прологе так:

```
sublist ( S, L) : -
conc ( L1, L2, L),
conc( S, L3, 12).
```

Ясно, что процедуру подсписок можно гибко использовать различными способами. Хотя она предназначалась для проверки, является ли какой—либо список подсписком другого, ее можно использовать, например, для нахождения всех подсписков данного списка. Так, на запрос:

sublist (S, [a,b,c]).

Пролог-система ответит:

S = **[];**
S = **[a,b];**
S = [a,b,c];
S = [b];

...

Перестановки

Иногда бывает полезно построить все перестановки некоторого заданного списка. Для этого мы определим отношение **reput** (перестановка) с двумя аргументами. Аргументы — это два списка, один из которых является перестановкой другого. Мы намереваемся порождать перестановки списка с помощью механизма автоматического перебора, используя процедуру **reput**, подобно тому, как это делается в следующем примере.

Запрос: **reput ([a, b, c], P)**.
Ответ Пролог-системы:
P = **[a,b,c];**
P = **[a,c,b];**
P = **[b,a,c];**
...

Программа для отношения **reput** в свою очередь опять может основываться на рассмотрении двух случаев в зависимости от вида первого списка:

(1) Если первый список пуст, то и второй список должен быть пустым.

(2) Если первый список не пуст, тогда он имеет вид **[X | L]**, и перестановку такого списка можно построить так, как это показано на рис. 4: вначале получить список **L1 —** перестановку **L**, а затем внести **X** в произвольную позицию **L1**.

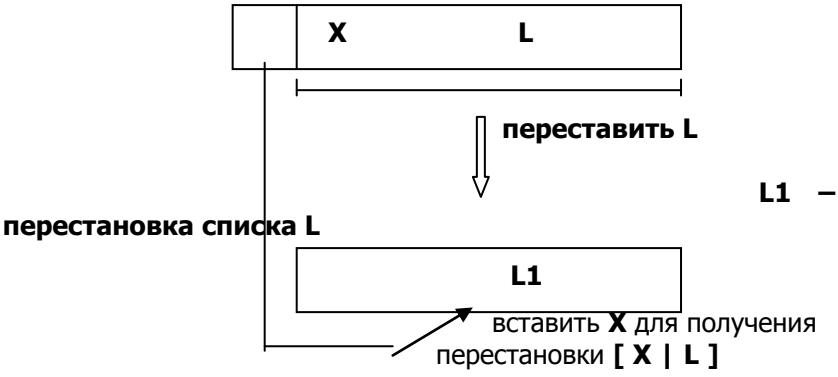


Рис. 4. Один из способов построения перестановки списка **[X | L]**.

Два предложения на Прологе, соответствующие этим двум случаям, таковы:

```
 reput ( [], []).
 reput ( [X | L], P) : -
reput ( L, L1),
append ( X, L1, P).
```

Другой вариант этой программы мог бы предусматривать удаление элемента X из первого списка, перестановку оставшейся его части — получение списка P, а затем добавление X в начало списка P. Соответствующая программа такова:

```
reput2( [], []).
                    reput2( L, [X | P] ) :-
delete ( X, L, L1),
reput2( L1, P).
```

Запрос к программе **reput** мог бы быть таким:
reput ([red, blue,green], P).

В результате будут построены все шесть перестановок:

```
P = [red, blue, green];
 P = [red, green, blue];
  P = [blue, red, green];
  P = [blue, green, red];
 P = [green, red, blue];
 P = [green, blue, red];
  no                                    (нет)
```

Задание на лабораторную работу

1. Изучить приведенный выше теоретический материал.
2. Выполнить следующее упражнение:

(a) Используя отношение **conc**, напишите цель, соответствующую вычеркиванию трех последних элементов списка L, результат — новый список L1. Указание: L — конкатенация L1 и трехэлементного списка.

(b) Напишите последовательность целей для порождения списка L2, получающегося из списка L вычеркиванием его трех первых и трех последних элементов.

3. Составить программу на Прологе, реализующую один из следующих вариантов задания по указанию преподавателя.

1) Определите предикат **палиндром (Список).**

Список называется палиндромом, если он читается одинаково, как слева направо, так и справа налево. Например, **[м, а, д, а, м]**.

2) Определите отношение

перевод (Списокl, Список2)

для перевода списка чисел от 0 до 9 в список соответствующих слов. Например,

перевод ([3,5,1,3], [три, пять, один, три])

Используйте в качестве вспомогательных следующие отношения:

означает (0, нуль).
означает (1, один).
означает (2, два).
...

3) Определите отношение

подмножество (Множество, Подмножество)

где **Множество и Подмножество —** два списка, представляющие два множества. Желательно иметь возможность использовать это отношение не только для проверки включения одного множества в другое, но и для порождения всех возможных подмножеств заданного множества. Например:

? - подмножество ([a,b,c], S).
S = [a, b, c];
S = [b, c];
S = [c];
S = [];
S = [a, c];
S = [a];
...

4) Определите отношение

разбениесписка (Список, Список1, Список2)

так, чтобы оно распределяло элементы списка между двумя списками **Список1** и **Список2,** и чтобы эти списки были примерно одинаковой длины. Например,

разбениесписка ([a,b,c,d,e], [a,c,e],[b,d]).

5) Определите отношение

последний (Элемент, Список)

так, чтобы **Элемент** являлся последним элементом списка **Список**. Напишите два варианта определения: (a) с использованием отношения **conc**, (b) без использования этого отношения.

6) Определите предикат

сумспис (Список, Сумма)

так, чтобы **Сумма** равнялась сумме чисел, входящих в **Список**.

7) Определите предикат

упорядоченный (Список)

который принимает значение «истина», если **Список** представляет собой упорядоченный список чисел. Например: **упорядоченный ([1, 5, 6, 6, 9, 12]).**

8) Определите предикат

подсумма (Множ, Сумма, ПодМнож)

где **Множ** – это список чисел, **ПодМнож** – подмножество этих чисел, а сумма чисел из **ПодМнож** равна **Сумма**. Например:

? – подсумма ([1, 2, 5, 3, 2], 5, SM).

SM = [1, 2, 2];

SM = [2, 3];

SM = [5];

...

9) Определите два предиката

chet(N) и **nechet(N)**

таким образом, чтобы они были истинными, если их аргументом является список четной или нечетной длины соответственно(здесь N – список). Например, список [a,b,c,d] имеет четную длину, [a,b,c] – нечетную.

10) Напишите программу подсчета суммы произведений элементов двух векторов X и Y ($\sum_{i=1}^{n} x_i \cdot y_i$), представленных списками X = [x_1, x_2, … , x_n] и Y = [y_1, y_2, … , y_n].

11) Определите отношение

n_элемент(N, Список, X)

которое выполняется, если X является N-м элементом спис-

ка **Список**.

12) Определите отношение **max(X, Y, Max)** так, чтобы **Max** равнялось наибольшему из двух чисел X и Y. Определить предикат

максспис (Список, Max)

так, чтобы **Max** равнялось наибольшему из чисел, входящих в **Список**. (Отношение **максспис** можно определить, используя отношение **max**).

13) Определите отношение **length(Список, X)** так, чтобы значение **X** равнялось длине списка **Список**.

Содержание отчета

1. Наименование и цель выполняемой работы.
2. Формулировка задания на лабораторную работу.
3. Листинг программы.
4. Примеры запросов и результаты их выполнения.

Контрольные вопросы

1. Что такое список и как он задается на языке Пролог?
2. Как записывается на Прологе пустой список?
3. Из каких частей состоит непустой список?
4. Как представить список в виде дерева?

ЛАБОРАТОРНАЯ РАБОТА НА ТЕМУ: «ПРЕДСТАВЛЕНИЕ МНОЖЕСТВ ДВОИЧНЫМИ ДЕРЕВЬЯМИ НА ПРОЛОГЕ»

Цель работы: получение навыков представления множеств двоичными деревьями на Прологе.

Теоретические сведения

Списки часто применяют для представления множеств. Такое использование списков имеет тот недостаток, что проверка принадлежности элемента множеству оказывается довольно неэффективной. Обычно предикат **принадлежит(X, L)** для проверки принадлежности X к L программируют так:

принадлежит (X, [X | L]).

принадлежит (X, [Y | L]):- принадлежит (X, L).

Для того чтобы найти X в списке L, эта процедура последовательно просматривает список элемент за элементом, пока ей не встретится либо элемент X, либо конец списка. Для длинных списков такой способ крайне неэффективен.

Для более эффективной реализации отношения принадлежности применяют различные древовидные структуры, в частности, двоичные деревья.

Двоичное дерево либо пусто, либо состоит из следующих трех частей:

- корень
- левое поддерево
- правое поддерево

Корень может быть чем угодно, а поддеревья должны сами быть двоичными деревьями. На рис. 1 показано представление множества [a, b, c, d] двоичным деревом. Элементы множества хранятся в виде вершин дерева. Пустые поддеревья на рис. 1 не показаны. Например, вершина b имеет два поддерева, которые оба пусты.

Существует много способов представления двоичных деревьев на Прологе. Одна из простых возможностей — сделать корень главным функтором соответствующего терма, а поддеревья — его аргументами. Тогда дерево рис. 1 примет вид

a(b, c(d))

Такое представление имеет среди прочих своих недостатков то слабое место, что для каждой вершины дерева нужен свой

функтор. Это может привести к неприятностям, если вершины сами являются структурными объектами.

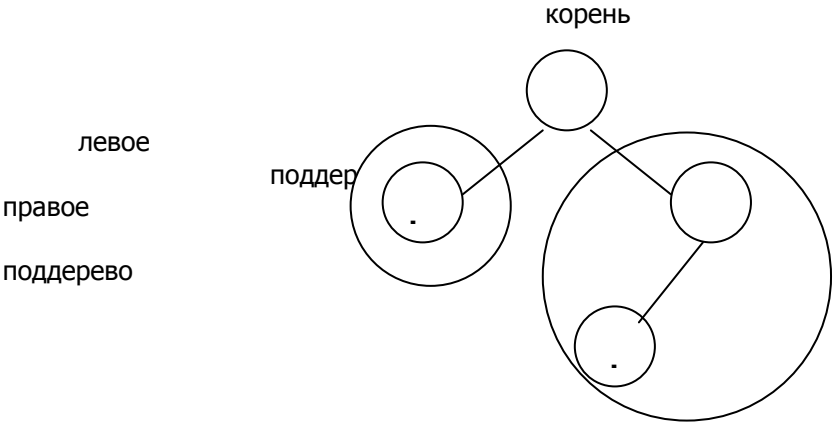


Рис.1. Двоичное дерево.

Существует более эффективный и более привычный способ представления двоичных деревьев: нам нужен специальный символ для обозначения пустого дерева и функтор для построения непустого дерева из трех компонент (корня и двух поддеревьев). Относительно функтора и специального символа сделаем следующий выбор:

Пусть атом **empty** представляет пустое дерево.

В качестве функтора примем **tree**, так что дерево с корнем X, левым поддеревом L и правым поддеревом R будет иметь вид терма **tree(L, X, R)** (см. рис.2).

В этом представлении дерево рис. 1 выглядит как

tree (tree(empty, b, empty), a,
tree (tree (empty, d, empty), c, empty)).

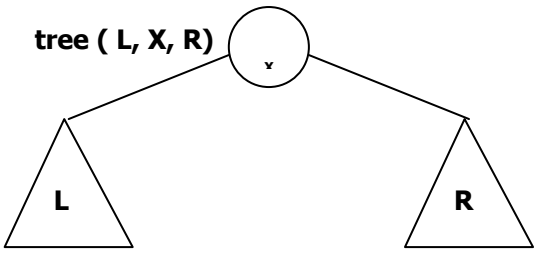


Рис. 2. Представление двоичных деревьев.

На Прологе определение дерева в разделе **domains** может выглядеть следующим образом:

```
domains
        der = tree ( der, k, der ) ; empty
        k = integer
```

Рассмотрим отношение принадлежности, которое будем обозначать **внутри**. Цель **внутри (X, T)** истинна, если **X** есть вершина дерева **T**. Отношение **внутри** можно определить при помощи следующих правил:

X есть вершина дерева **T**, если

- корень дерева T совпадает с X, или
- X — это вершина из левого поддерева, или
- X — это вершина из правого поддерева.

Эти правила непосредственно транслируются на Пролог следующим образом:

```
внутри ( X, tree ( _, X, _ ) ).
внутри ( X, tree ( L, _, _ ) ) :- внутри ( X, L).
внутри ( X, tree ( _, _, R) ) :- внутри ( X, R).
```

Очевидно, что цель **внутри (X, empty)** терпит неудачу при любом X.

Если между элементами множества существует отношение порядка, то можно упорядочить данные в дереве слева направо в соответствии с этим отношением. Будем говорить, что непустое дерево **tree (Лев, X, Прав)** упорядочено слева направо, если

(1) все вершины левого поддерева **Лев** меньше **X**;
(2) все вершины правого поддерева **Прав** больше **X;**
(3) оба поддерева упорядочены.

Будем называть такое двоичное дерево ***двоичным справочником.*** Пример показан на рис.3.

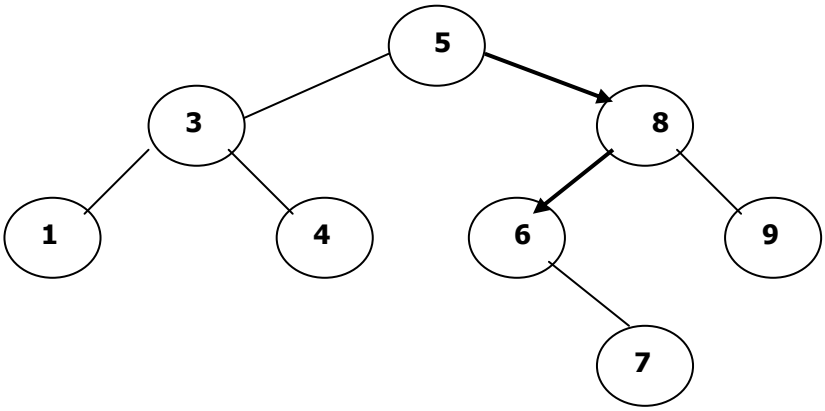


Рис. 3. Двоичный справочник. Элемент 6 найден после прохода по отмеченному пути

Преимущество упорядочивания состоит в том, что для поиска некоторого объекта в двоичном справочнике всегда достаточно просмотреть не более одного поддерева. Экономия при поиске объекта X достигается за счет того, что, сравнив X с корнем, мы можем сразу же отбросить одно из поддеревьев. Например, пусть мы ищем элемент 6 в дереве, изображенном на рис. 3. Мы начинаем с корня 5, сравниваем 6 с 5, получаем 6 > 5. Поскольку все элементы данных в левом поддереве должны быть меньше, чем 5, единственная область, в которой еще осталась возможность найти элемент 6, — это правое поддерево. Продолжаем поиск в правом поддереве, переходя к вершине 8, и т.д.

Общий метод поиска в двоичном справочнике состоит в следующем:

Для того чтобы найти элемент X в справочнике Д, необходимо:

- если X — это корень справочника Д, то считать, что X уже найден, иначе
- если X меньше, чем корень, то искать X в левом поддереве, иначе
- искать X в правом поддереве;
- если справочник Д пуст, то поиск терпит неудачу.

Эти правила запрограммированы в виде следующей процедуры:

```
внутри ( X, tree ( _, X, _ )).
внутри ( X, tree ( Лев, Корень, Прав )):-
        больше ( Корень, X ),          % Корень больше, чем X
        внутри ( X, Лев ).             % Поиск в левом поддереве

внутри ( X, tree ( Лев, Корень, Прав )):-
        больше ( X, Корень ),          % X больше, чем корень
        внутри ( X, Прав ).            % Поиск в правом поддереве
```

Отношение **больше (X, Y)**, означает, что **X** больше, чем **Y**. Если элементы, хранимые в дереве, — это числа, то под «больше, чем» имеется в виду просто **X >Y**.

Вообще говоря, поиск элемента в справочнике эффективнее, чем поиск в списке. Пусть n **—** число элементов множества. Если множество представлено списком, то ожидаемое время поиска будет пропорционально его длине n. В среднем нам придется просмотреть примерно половину списка. Если множество представлено двоичным деревом, то время поиска будет пропорционально глубине дерева. Глубина дерева — это длина самого длинного пути между корнем и листом дерева. Однако следует помнить, что глубина дерева зависит от его формы.

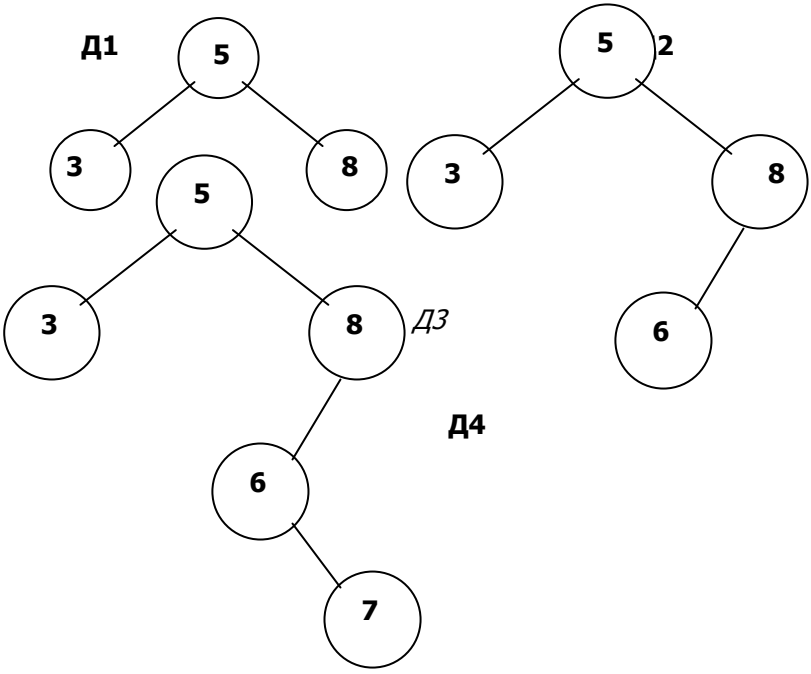
Говорят, что дерево (приближенно) сбалансировано, если для каждой вершины дерева соответствующие два поддерева содержат примерно равное число элементов. Если дерево хорошо сбалансировано, то его глубина пропорциональна log n. В этом случае говорят, что дерево имеет логарифмическую сложность. Сбалансированный справочник лучше списка настолько же, насколько log n меньше n. К сожалению, это верно только для приближенно сбалансированного дерева. Если происходит разбалансировка дерева, то производительность падает. В случае полностью разбалансированных деревьев, дерево фактически превращается в список. Глубина дерева в этом случае равна n, а производительность поиска оказывается столь же низкой, как и в случае списка.

Добавление и удаление элемента в двоичном дереве

Если мы имеем дело с динамически изменяемым множеством элементов данных, то нам может понадобиться внести в него новый элемент или удалить из него один из старых. В связи с этим набор основных операций, выполняемых над множеством **S** таков:

внутри (X, S) % X содержится в S
добавить(S, X, S1) % Добавить X к S, результат – S1
удалить(S, X, S1) % Удалить X из S, результат - S1.

Определим отношение ***добавить***. Простейший способ: ввести новый элемент на самый нижний уровень дерева, так что он станет его листом. Место, на которое помещается новый элемент, выбрать таким образом, чтобы не нарушить упорядоченность дерева. На рис. 4 показано, какие изменения претерпевает дерево в процессе введения в него новых элементов. Назовем такой метод вставки элемента во множество **доб_лист (Д, X, Д1)**



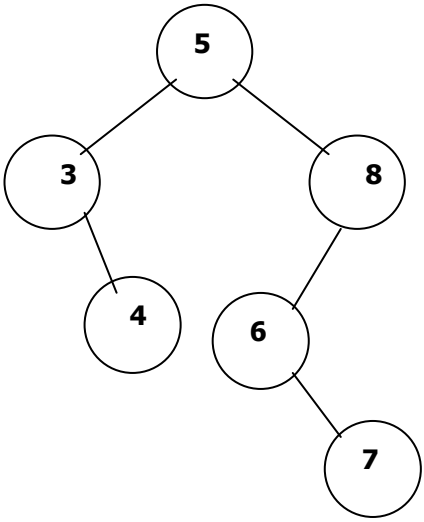


Рис. 4. Введение в двоичный справочник нового элемента на уровне листьев. Показанные деревья соответствуют следующей последовательности вставок:

добавить (Д1,6,Д2), добавить (Д2,7,Д3), добавить (Д3, 4, Д4)

Правила добавления элемента на уровне листьев таковы:

• Результат добавления элемента X к пустому дереву есть дерево **tree (empty, X, empty).**

• Если X совпадает с корнем дерева Д, то Д1 = Д (во множестве не допускается дублирование элементов).

• Если корень дерева Д больше, чем X, то X вносится в левое поддерево дерева Д; если корень меньше, чем X, то X вносится в правое поддерево.

Соответствующая программа имеет вид:

доб_лист (empty, X, tree (empty, X, empty)).
доб_лист (tree (Лев, X, Прав), X, tree (Лев, X, Прав)).

доб_лист (tree (Лев, Кор, Прав), X, tree (Лев1, Кор, Прав)): —
больше (Кор, X),
доб_лист (Лев, X, Лев1).
доб_лис (tree (Лев, Кор, Прав), X, tree (Лев, Кор, Прав1)):-

больше (X, Кор),
доб_лист (Прав, X, Прав1).

Теперь рассмотрим операцию ***удалить***. Лист дерева удалить легко, однако удалить какую—либо внутреннюю вершину — дело не простое. К сожалению, если X — это внутренняя вершина, то возникает проблема, иллюстрацией к которой служит рис. 5. Вершина X имеет два поддерева Лев и Прав. После удаления вершины X в дереве образуется «дыра», и поддеревья Лев и Прав теряют свою связь с остальной частью дерева. К вершине A оба эти поддерева присоединить невозможно, так как вершина A способна принять только одно из них.

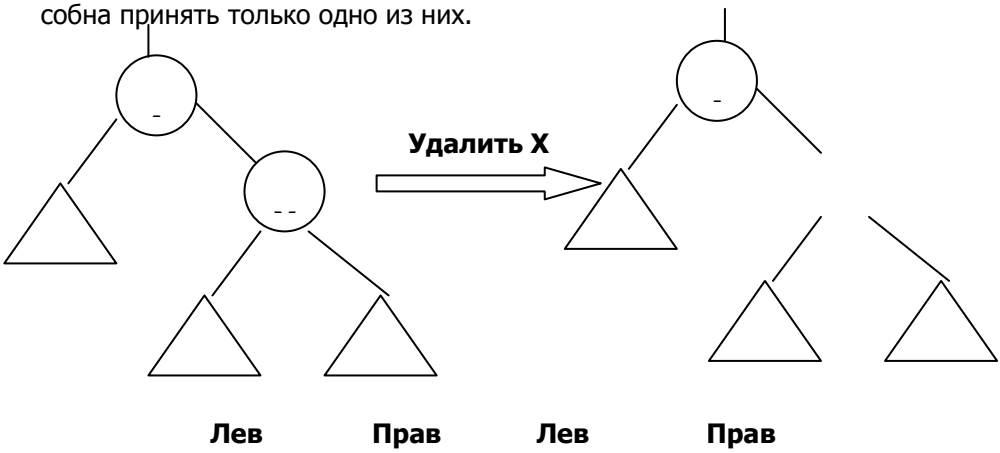


Рис. 5. Удаление X из двоичного справочника. Возникает проблема наложения «заплаты» на место удаленного элемента X.

Если одно из поддеревьев **Лев** и **Прав** пусто, то существует простое решение: подсоединить к A. непустое поддерево. Если же оба поддерева не пусты, то можно использовать следующую идею (рис. 6): если самую левую вершину Y поддерева **Прав** переместить из ее текущего положения вверх и заполнить ею пробел, оставшийся после X, то упорядоченность дерева не нарушится. Разумеется, та же идея сработает и в симметричном случае, когда перемещается самая правая вершина поддерева **Лев**.

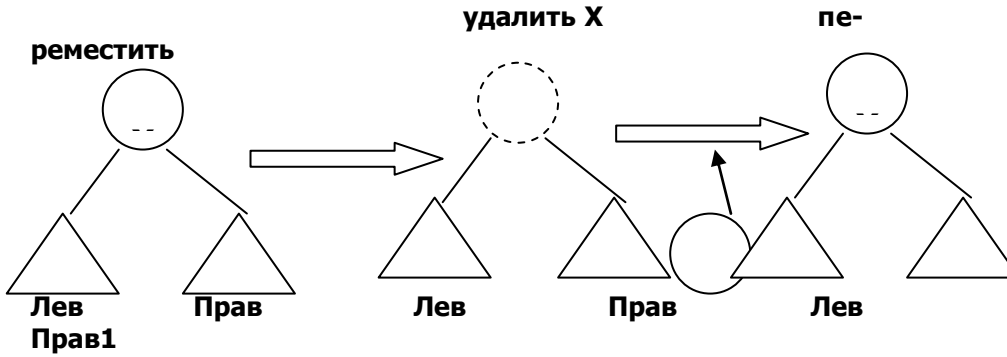


Рис. 6. Заполнение пустого места после удаления X.

Программа, реализующая операцию удаления элементов в соответствии с изложенными выше соображениями, имеет вид:

```
уд ( tree ( empty, X, Прав ), X, Прав ).

уд ( tree ( Лев, X, empty ), X, Лев ).

уд ( tree ( Лев, X, Прав ), X, tree ( Лев, Y, Правl ) ) :-
уд_мин ( Прав, Y, Правl ).
уд ( tree ( Лев, Кор, Прав ), X, tree ( Лев1, Кор, Прав )
):-
больше ( Кор, X ),
уд ( Лев, X, Лев1 ).

уд ( tree ( Лев, Кор, Прав ), X, tree ( Лев, Кор, Прав1 )
) :-
        больше ( X, Кор ),
        уд ( Прав, X, Правl ).

уд_мин ( tree ( empty, Y, Прав ), Y, Прав ).

уд_мин ( tree ( Лев, Кор, Прав), Y, tree ( Лев1, Кор,
Прав ) ) : -
         уд_мин ( Лев, Y, Лев1 ).
```

Основную работу по перемещению самой левой вершины

выполняет отношение

уд_мин (Дер, Y, Дер1)

Здесь **Y** — минимальная (т.е. самая левая) вершина дерева **Дер**, а **Дерl** — то, во что превращается дерево **Дер** после удаления вершины **Y**.

Отображение деревьев

Так же, как и любые объекты данных в Прологе, двоичное дерево T может быть непосредственно выведено на печать при помощи встроенной процедуры write. Однако цель

write(T)

хотя и отпечатает всю информацию, содержащуюся в дереве, но действительная структура дерева никак при этом не будет выражена графически. Довольно утомительная работа — пытаться представить себе структуру дерева, рассматривая прологовский терм, которым она представлена. Поэтому во многих случаях желательно иметь возможность отпечатать дерево в такой форме, которая графически соответствует его структуре.

Существует простой способ это сделать. Уловка состоит в том, чтобы изображать дерево растущим слева направо, а не сверху вниз, как обычно. Дерево нужно повернуть влево таким образом, чтобы корень стал его крайним слева элементом, а листья сдвинулись вправо (рис.7).

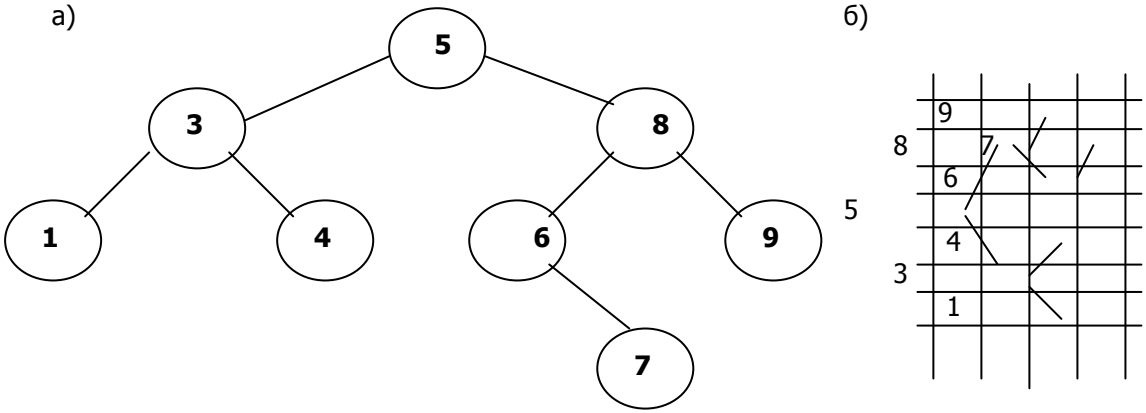


Рис. 7. (а) Обычное изображение дерева. (б) То же дерево, отпечатанное процедурой **draw** (дуги добавлены для ясности).

Определим процедуру

draw (T)

так, чтобы она отображала дерево в форме, показанной на рис. 7. Принцип работы этой процедуры:

Для того, чтобы отобразить непустое дерево T, необходимо:

(1) отобразить правое поддерево дерева отступом вправо на расстояние H;

(2) отпечатать корень дерева T;

(3) отобразить левое поддерево дерева T отступом вправо на расстояние H.

Величина отступа H, которую можно выбирать по желанию, — это дополнительный параметр при отображении деревьев. Введем процедуру

draw2(T, H)

печатающую дерево T с отступом на **H** пробелов от левого края листа. Связь между процедурами **draw** и **draw2** такова:

draw (T) :— draw2(T, 0).

Программа отображения имеет вид: (в этой программе предусмотрен сдвиг на 2 позиции для каждого уровня дерева)

```
draw ( T):- draw2 ( T, 0 ).

draw2 ( empty, _ ).
draw2 ( tree ( L, X, R ), Отступ ):-
        Отступ2 = Отступ + 2,
        draw2 ( R, Отступ2 ),
        tab ( Отступ), write( X), nl,
          draw2 ( L, Отступ2).
```

В программе использован встроенный оператор **nl** – перевод строки. Процедуру **tab**, выполняющую табуляцию на заданное число позиций легко определить.

Задания на лабораторную работу

С помощью процедур работы с деревьями, приведенными выше, построить дерево, представленное на рис. 3. Отобразить это дерево на экране с помощью процедуры **draw**, определив предварительно процедуру **tab**. Выполнить одно из ниже приве-

денных заданий по указанию преподавателя.

1. Определите процедуру
glubina (D, X)

вычисляющую глубину X двоичного дерева D в предположении, что глубина пустого дерева равна 0, а глубина одноэлементного дерева равна 1.

2. Определите отношение
lineariz (D, L)

соответствующий «выстраиванию» всех вершин дерева D в список L.

3. Определите отношение
max_element (D, X)

таким образом, чтобы переменная **X** приняла значение наибольшего из элементов, хранящихся в дереве **D**.

4. Внесите изменения в процедуру
внутри (Элемент, Справочник)

добавив в нее третий аргумент **Путь** таким образом, чтобы можно было бы получить путь между корнем справочника и указанным элементом.

5. Определите отношение
min_element (D, Element)

таким образом, чтобы переменная **Element** приняла значение наименьшего из элементов дерева **D**.

6. Определите предикат
member(X, Tree),

истинный, если элемент **X** является одной из вершин дерева **Tree**.

7. Определите предикат
subtree(S, T),

определяющий, является ли **S** поддеревом **T**.

8. Определите отношение
sum_tree(T, Sum),

истинное, если число **Sum** равно сумме всех вершин дерева **T**.

9. Определите отношение
num_tree(T, L),

истинное, если число **L** равно количеству элементов дерева **T**.

10. Определите отношение
disbalans_tree(Tree),

истинное, если в дереве **Tree** число элементов в поддеревьях различное.

Содержание отчета

1. Наименование и цель выполняемой работы.
2. Формулировка задания на лабораторную работу.
3. Листинг программы.
4. Примеры запросов и результаты их выполнения.

Контрольные вопросы

1. Что такое двоичное дерево?
2. Как упорядочены элементы в двоичном дереве?
3. Как двоичное дерево задается на языке Пролог?

ЛАБОРАТОРНАЯ РАБОТА НА ТЕМУ: «ПРЕДСТАВЛЕНИЕ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ НА ПРОЛОГЕ».

Цель работы: изучить способы представления содержимого реляционной базы данных и реляционных операций в программе на языке логического программирования **Пролог**.

РУКОВОДСТВО К ВЫПОЛНЕНИЮ

Модель реляционной базы данных и операции реляционной алгебры легко выражаются в логическом программировании, в частности, на языке Пролог.

Факты логической программы соответствуют отношениям реляционной базы данных, при этом арность отношений совпадает с арностью предикатов, представляющих факты. Пять основных операций определяют реляционную алгебру: объединение, разность, декартово произведение, проекция и выборка. Покажем, как каждая из них выражается в логической программе на Прологе.

Операция объединения строит одно *n*-арное отношение из двух совместимых по типу *n*-арных отношений ***r*** и ***s***. Новое отношение, которое обозначим ***r_union_s***, является объединением ***r*** и ***s***. Это отношение непосредственно задается логической программой, состоящей из двух правил:

r_union_s $(X_1, \ldots, X_n)$ ***:- r*** $(X_1, \ldots, X_n)$***.***
r_union_s $(X_1, \ldots, X_n)$ ***:- s*** $(X_1, \ldots, X_n)$***.***

Определение разность использует понятие отрицания, которое задается встроенным предикатом ***not***. Разность определяется как *n*-арное отношение ***r_diff_s*** для *n*-арных отношений ***r*** и ***s***:

r_diff_s $(X_1, \ldots, X_n)$ ***:- r*** $(X_1, \ldots, X_n)$***, not s*** $(X_1, \ldots, X_n)$***.***

Декартово произведение может быть определено одним правилом. Если ***r*** - *m*-арное отношение, а ***s*** – *n*–арное отношение, то (*m*+*n*)-арное отношение ***r_x_s*** определяется так:

r_x_s $(X_1, \ldots, X_{m+n})$ ***:- r*** $(X_1, \ldots, X_n)$***, s*** $(X_{m+1}, \ldots, X_{m+n})$***.***

Проекция состоит в построении отношения, использующего лишь некоторые аргументы исходного отношения. Определение в любом конкретном случае не вызывает сложностей. Например, проекция ***r13*** оставляет первый и третий аргументы трехарного отношения, т.е.

r13 (X_1, X_3) :- r (X_1, X_2, X_3).

Также просто описывается любой конкретный случай выборки. Рассмотрим отношение, описывающее наборы, в которых третьи элементы больше вторых, и отношение, в котором первый элемент есть Смит или Джонс. В обоих случаях для иллюстрации используется трехарное отношение ***r***. Первый пример состоит в построении отношения ***r1***:

r1 (X_1, X_2, X_3) :- r (X_1, X_2, X_3), $X_2 > X_3$.

Второй пример состоит в построении отношения ***r2***, использующего дизъюнктивное отношение ***смит_или_джонс***:

r2 (X_1, X_2, X_3) :- r (X_1, X_2, X_3), смит_или_джонс (X_1).
смит_или_джонс (смит).
смит_или_джонс (джонс).

Некоторые производные операции реляционной алгебры непосредственно выражаются в конструкциях логического программирования. Например, операция пересечения. Если ***r*** и ***s*** – некоторые совместимые по типу *n*–арные отношения, то их пересечение ***r_meet_s*** тоже является *n*–арным отношением, задаваемым единственным правилом:

r_meet_s (X_1, …, X_n) :- r (X_1, … , X_n), s (X_1, … , X_n).

Иногда возникает необходимость собрать данные из базы данных в список для последующей их обработки. Пролог содержит встроенный предикат **findall**, позволяющий справиться с этой задачей.

Описание предиката **findall** выглядит следующим образом:

findall (Var_name, Predicate_expression, List_name).

Здесь **Var_name** обозначает объект входного предиката

Predicate_expression, а **List_name** является именем переменной выходного списка. Переменная должна относиться к домену списков, объявленному в разделе **domains**.

Для пояснения рассмотрим предикат базы данных
football(name, points)

Этот предикат порождает 5 утверждений:

football("Спартак",116.0).
football("ЦСКА",121.0).
football("Динамо",114.0).
football("Локомотив",99.0).
football("Сатурн",122.0).

Эти утверждения содержат сведения об очках, набранных командами. Предположим, что необходимо сложить все очки и усреднить их. Сбор очков в список осуществляется при помощи встроенного предиката **findall**:

findall (Points, football(_,Points), Points_list)

Здесь **Points** является свободной переменной для значений набранных командами очков, а **Points_list** – списочной переменной, элементы которой принадлежат к тому же домену, что и **Points**, в данном случае домену **real**.

Предикат **findall** работает следующим образом. Он просматривает все утверждения с предикатом **football**, начиная с первого. Значение переменной **Points (116)**, взятое из этого утверждения, присваивается голове списка **Points_list**. Остальные значения **Points** помещаются в список на последующие позиции. По завершении работы **findall** переменная **Points_list** принимает значение

[116.0, 121.0, 114.0, 99.0, 122.0]

Для подсчета среднего значения набранных очков применяется рекурсивное правило

sum([],0,0).
sum([H|T],S,N):- sum(T,S1,N1), S = H + S1, N = N1 + 1.

Это правило заносит в переменную **S** сумму элементов

списка, являющегося первым аргументом предиката **sum**, а в переменную **N** – число элементов списка.

Для получения суммы всех элементов списка **Points_list** это правило необходимо задать в качестве подцели **sum(Points_list, S, N).**

После применения правила **S = 572**, **N = 5**.

Правило для нахождения среднего значения выглядит так: **Avg = S/N.** Правило **Avg** использует подсчитанные правилом **sum** значения переменных **S** и **N**. Результатом является присвоение переменной **Avg** частного от деления **S** и **N**, т.е. **114.4**.

Следующая программа демонстрирует использование предиката **findall** для сбора данных из базы данных в список.

```
/*  Программа Очки */
domains
  name = string
  points = real
  list = points*
predicates
  football(name, points)
  sum(list, points, integer)
  report
clauses
  football("Спартак",116.0).
  football("ЦСКА",116.0).
  football("Динамо",116.0).
  football("Локомотив",116.0).
  football("Сатурн",116.0).

report:-
  findall( Points, football( _ ,Points), Points_list),
  sum( Points_list, S, N),
  Avg = S/N,
  write("Очки футбольных команд:"), nl,
  write("  Среднее значение =  ", Avg).
sum([],0,0).
sum([H|T], S, N):- sum(T, S1, N1), S = H + S1, N = N1 + 1.

goal
  report.

/***  конец программы ***/
```

Программа подсчитывает сумму очков команд и их среднее значение. Внутренняя цель программы есть **report**. Цель представляет собой правило, содержащее подцели **findall**, **sum**, **Avg**, а также предикаты, осуществляющие вывод полученных результатов в нужной форме.

Начиная свою работу, программа пытается удовлетворить подцель **findall** в том виде, в котором она была описана. Когда подцель удовлетворена, делается попытка удовлетворить подцель **sum**, а затем **Avg**. Переменной **Avg** при этом присваивается значение **114.4**, которое используется затем предикатом **write**. Теперь все подцели удовлетворены, следовательно, удовлетворена и цель программы.

Задание на лабораторную работу.

1. Представить в программе на Прологе следующую базу данных.

Дана реляционная БД поставщиков, деталей и проектов. Поставщики поставляют определенные детали для определенных проектов. В БД определено три объекта - «поставщик», «деталь» и «проект» - и отношение «поставка» между этими тремя объектами.

Для каждого поставщика (таблица S) заданы: номер (s_n), имя (sname), статус (status), город (s_city), где поставщик находится.

Для детали (таблица P) заданы: номер (p_n), название (pname), цвет (color), вес (weight), город (p_city), где деталь хранится.

Для проекта (таблица J) заданы: номер (j_n), название (jname), город (j_city).

Каждая поставка (таблица SPJ) задается номером поставщика (s_n), номером детали (p_n), номером проекта (j_n), количеством поставляемых деталей (qty).

Таблицы БД имеют следующий вид (первичные ключи таблиц выделены двойной линией):

S P

S_n	Sname	Status	S_city
S1	Смит	20	Лондон
S2	Джонс	10	Париж

S3	Блэк	30	Париж
S4	Кларк	20	Лондон
S5	Адамс	30	Афины

J

P_n	Pname	Color	Weight	P_city
P1	Муфта	красный	12	Лондон
P2	Рама	Зеленый	17	Париж
P3	ось	Синий	17	Рим
P4	Колпак	Красный	14	Лондон
P5	Диск	Синий	12	Париж
P6	Вал	красный	19	Лондон

SPJ

J_n	Jname	J_ city
J1	Sorter	Париж
J2	Display	Рим
J3	OCR	Афины
J4	Console	Афины
J5	RAID	Лондон
J6	EDS	Осло
J7	Tape	Лондон

S_n	P_n	J_n	qty
S1	P1	J1	200
S2	P3	J1	400
S2	P3	J2	200
S2	P3	J3	200
S2	P3	J4	500
S2	P3	J5	600
S2	P3	J6	400
S2	P3	J7	800
S2	P5	J2	100
S3	P3	J1	200
S4	P6	J1	300
S4	P6	J7	200
S5	P2	J2	100
S5	P2	J4	500
S5	P5	J5	100
S5	P5	J7	200
S5	P6	J2	100
S5	P1	J4	200
S5	P3	J4	800
S5	P4	J4	400
S5	P5	J4	500
S5	P6	J1	300

2. Определить в программе на Прологе процедуры в соответствии с вариантом задания.

Замечание. Для выполнения некоторых заданий необходимо использовать процедуры обработки списков, рассмотренные

в предыдущих лабораторных работах.

Варианты заданий

1. Соединить таблицы S и P по равенству значений атрибутов S_city и P_city.

Выбрать номера поставщиков, поставляющих деталь P1 для проектов в Афинах.

2. Соединить таблицы P и J по равенству значений атрибутов P_city и J_city.

Выбрать номера поставщиков, поставляющих деталь P5 в количестве не менее 300.

3. Соединить таблицы S и J по равенству значений атрибутов S_city и J_city.

Выбрать номера проектов, расположенных в Лондоне, для которых поставляется детали, хранящиеся в Лондоне.

4. Выбрать номера деталей, поставляемых для проекта J4 поставщиками из Лондона. Получить общее число деталей, поставляемых для проекта J4.

5. Выбрать имена поставщиков, обеспечивающих более одного проекта.

6. Выбрать номера проектов, обеспечиваемых всеми поставщиками.

7. Получить номера деталей, поставляемых Смитом или Блэком.

8. Получить номера проектов, для которых поставляются все детали.

9. Получить общее количество деталей, поставляемых для каждого проекта.

10. Получить имена поставщиков, поставляющих самые тяжелые детали.

11. Получить имена поставщиков, выполняющих поставки для всех проектов.

12. Получить номера поставщиков, поставляющих одну и ту же деталь для одного и того же проекта.

13. Получить общее количество деталей, поставляемых каждым поставщиком.

14. Получить имена поставщиков, поставляющих все детали.

15. Получить число номеров деталей, поставляемых поставщиком S5 для проекта J4.

16. Получить номера проектов, размещенных в Риме или Афинах, для которых поставляется деталь P5.
17. Получить число проектов, размещенных в каждом городе.
18. Получить цвета деталей, поставляемых поставщиком S1 или S3.
19. Получить средний объем поставок для проекта J4.
20. Получить общее количество деталей, поставляемых для проектов в Лондоне.
21. Получить имена поставщиков, выполняющих поставки максимального объема.
22. Получить число проектов в каждом городе.
23. Получить общее количество деталей, поставляемых поставщиками из Лондона.
24. Получить средний объем поставок деталей из Лондона.
25. Получить имена проектов, для которых выполняются поставки максимального объема.
26. Получить число номеров деталей, хранящихся в каждом городе.
27. Получить номера поставщиков, выполняющих поставки минимального объема.
28. Получить общий объем поставок деталей, хранящихся в Лондоне.
29. Получить имена поставщиков, не выполняющих поставки для проектов в Афинах.
30. Получить номера проектов, для которых не поставляются детали из Парижа.
31. Получить номера деталей, которые не поставляются поставщиками из Парижа.
32. Получить число номеров деталей, поставляемых для каждого проекта.
33. Получить списки номеров деталей, поставляемых для каждого проекта.
34. Получить число поставщиков для каждого проекта.
35. Получить списки номеров поставщиков для каждого проекта.
36. Получить число номеров деталей, поставляемых каждым поставщиком.
37. Получить списки номеров деталей, поставляемых каждым поставщиком.

Содержание отчета

1. Наименование и цель выполняемой работы.
2. Формулировка задания на лабораторную работу.
3. Листинг программы.
4. Примеры запросов и результаты их выполнения.

Контрольные вопросы

1. Каким образом можно представить на Прологе строки таблицы базы данных?
2. Какой встроенный предикат Пролога позволяет собрать данные из базы данных в список?
3. Какова структура предиката **findall** ?

ЛАБОРАТОРНАЯ РАБОТА НА ТЕМУ: РАЗРАБОТКА ЭКСПЕРТНОЙ СИСТЕМЫ НА ЯЗЫКЕ CLIPS

Цель работы: ознакомиться с особенностями языка CLIPS, получить практические навыки разработки экспертных систем, основанных на использовании продукционной модели представления знаний.

Общие сведения

CLIPS располагает тремя механизмами представления знаний: процедурным, эвристическим и объектно-ориентированным. Рассмотрим первые два механизма.

Процедурный механизм позволяет пользователю при помощи встроенных в язык функций разрабатывать или конструировать новые функции, выполняющие некоторые действия или возвращающие какие-либо значения. В этом смысле CLIPS напоминает такие известные языки программирования, как C, C++ или Pascal. Так, для создания пользовательских функций используется конструктор deffunction, имеющий следующий синтаксис:

```
(deffunction имя_функции
[необязательный комментарий]
(список формальных параметров)
(действие_1)
(действие_2)
.....
(действие_N))
```

Например, определим функцию om(x,y), которая возвращает целую часть частного от деления переменной y на переменную x:

```
(deffunction om
( ?x ?y)
(div ?y ?x))
```

Обратите внимание на то, что в CLIPS имя переменной начинается с символа " ? ", и что для вызова функции (в данном случае – встроенной функции деления нацело div) используется префиксная нотация, а также на то, что вся конструкция представляет собой список, состоящий из четырех полей.

Эвристический механизм представления знаний в CLIPS ре-

ализуется при помощи правил в форме

ЕСЛИ условие_1 и ... и условие_N выполняются,
ТО
ВЫПОЛНИТЬ действие_1 и ... и действие_N.

Список условий называется левой частью правила (Left-Hand Side или LHS). Список действий называется правой частью правила (Right-Hand Side или RHS). Возможность применить конкретное правило определяется тем, выполняются ли условия, которые сформулированы в его левой части. Выполнение или невыполнение условий определяется в момент их сопоставления с так называемыми фактами, которые образуют ни что иное, как базу данных. В CLIPS такая база данных может представлять некоторую

предметную область, исходное или текущее состояние какой-либо проблемы, может моделировать в пространстве или во времени поведение какой-либо системы или любой сущности, которую можно описать посредством множества записей в виде списков.

Существует несколько способов создания базы данных, один из них – использование конструктора deffacts. Его синтаксис таков:

(deffacts имя_базы_данных
[необязательный комментарий]
(факт_1)
(факт_2)
.....
(факт_N))

Каждый факт в базе данных представляет собой запись в виде списка. Список может содержать одно или несколько полей, принимающих символьные либо числовые значения. Список также может быть пустым.

Если каждое условие в левой части правила находит себя среди фактов – происходит активизация правила и выполняются ВСЕ действия, записанные в его правой части. В противном случае правило не активизируется.

Работа правила напоминает условный оператор if-then, присутствующий во многих процедурных языках программирования. Принципиальная разница заключается в том, что оператор if-then

выполнится в любом случае, когда до него дойдет очередь в программе. Что касается правила, то интерпретатор CLIPS еще "подумает", выполнять его или нет. Так, при старте программы, содержащей множество фактов и правил, интерпретатор CLIPS запускает машину логического вывода, которая выясняет, какие из правил можно активизировать. Это выполняется циклически, причем каждый цикл состоит из трех шагов:

- сопоставление фактов и правил;
- выбор правила, подлежащего активизации;
- выполнение действий, предписанных правилом.

Таким образом, правила, взаимодействующие с базой данных в виде фактов, вносят в нее функциональность и образуют вместе с ней базу знаний.

Для создания правила используется конструктор defrule, который имеет следующий синтаксис:

```
(defrule имя_правила
[необязательный комментарий]
[необязательное объявление]
(условие_1)
(условие_2)
.....
(условие_M)
=>
(действие_1)
(действие_2)
.....
(действие_N))
```

Обратите внимание: левая часть правила отделяется от правой комбинацией символов "=>", а количество условий и действий в правиле в общем случае не совпадает. Для пояснения вышесказанного рассмотрим несколько примеров.

Пример 1

Рассмотрим предметную область, которая представляет участников некоторой конференции, приехавших из разных городов. На подобных мероприятиях все участники обычно проходят регистрацию. Пусть эта процедура представляет собой ввод сведений об участниках в базу данных, в которой на каждого участника выделяется одна запись (факт), состоящая из списка с тремя

полями. Пусть первое поле имеет символьное значение rep – сокращение от representative (представитель). В общем случае это значение может быть любым, а поле может отсутствовать. Во втором поле списка хранится фамилия участника, а в третьем – город, из которого участник прибыл. Содержимое фактов базы данных с именем rep может быть, например, таким:

```
(deffacts rep
(rep Alejnov Odessa)
(rep Ladak Odessa)
(rep Slobodjanjuk Lvov)
(rep Klitka Lvov)
(rep Bojko Kiev)
(rep Pustovit Odessa)
(rep Spokojnij Odessa)
(rep Shamis Odessa)
(rep Lobovko Kiev)
(rep Zadorozhna Lvov)
(rep Javorskij Lvov))
```

Используя любой текстовый редактор, создадим и сохраним базу данных в виде текстового ASCII-файла с именем, повторяющим имя базы данных (то есть rep). Это позволяет легко редактировать данные, независимо от каких-либо других программных модулей, добавляя новых участников или удаляя выбывших.

После окончания конференции организаторы подводят итоги, определяя массу показателей. В частности, пусть требуется определить количество представителей от каждого города. Алгоритм решения такой задачи прост. Для каждого города задаем счетчик и последовательно просматриваем списки в записях файла rep. Если в записи третье поле списка имеет значение Kiev, то содержимое соответствующего счетчика увеличиваем на единицу. Для других городов – аналогично. Программа на языке CLIPS, реализующая указанный алгоритм, может быть, например, такой:

```
(defglobal ?*odessa* = 0)
(defglobal ?*kiev* = 0)
(defglobal ?*lvov* = 0)
(defrule start
(initial-fact)
=>
(printout t crlf «REPRESENTATIVES» crlf)
```

```
(defrule odessa
(rep ? Odessa)
=>
(bind ?*odessa* (+ ?*odessa* 1)))
(defrule kiev
(rep ? Kiev)
=>
(bind ?*kiev* (+ ?*kiev* 1)))
(defrule lvov
(rep ? Lvov)
=>
(bind ?*lvov* (+ ?*lvov* 1)))
(defrule result
(declare (salience -1))
(initial-fact)
=>
(printout t «from Odessa: « ?*odessa* crlf)
(printout t «from Kiev: « ?*kiev* crlf)
(printout t «from Lvov: « ?*lvov* crlf))
```

В первых трех строках программы при помощи конструктора defglobal объявляются три глобальные переменные: ?*odessa*, ?*kiev* и ?*lvov*. Эти переменные являются счетчиками. В CLIPS переменная может быть и локальной – но тогда она связывается только с тем правилом, в котором объявляется.

Далее следует правило с именем start, левая часть которого представляет собой запись (initial-fact). Так обозначается системный начальный факт, который создается в рабочей памяти интерпретатора CLIPS по команде (reset) до запуска программы на выполнение. Для чего он нужен? Дело в том, что в CLIPS-программах распространенными правилами являются такие, которые добавляют факты в базу данных, либо, наоборот, удаляют их. Типичной является ситуация, когда при старте программы в базе данных нет фактов, удовлетворяющих хотя бы одному правилу. В этом случае программа ничего не выполнит. Для того чтобы начать вычисления и используется системный начальный факт, который, независимо от фактов в базе данных, активизирует некоторое правило, добавляющее такие факты, которые, в свою очередь, активизируют правила, условия которых не выполнялись в начальный момент.

В данной программе (initial-fact) запускает правило start, которое активизируется независимо от фактов в файле rep и при-

сутствует в программе только с одной целью – вывести заголовок. Для этого в его правой части вызывается встроенная функция printout с ключом t, выводящая на стандартное устройство вывода (монитор) заголовок, заключенный в кавычки. Комбинация символов crlf является аналогом endl в C++ и служит для перевода курсора на следующую строку.

Следующие три правила с именами odessa, kiev и lvov можно назвать ядром программы. В них производится подсчет количества участников – соответственно, из Одессы, Киева и Львова.

Рассмотрим правило lvov. Оно активизируется в том случае, когда в базе данных находится факт (rep ? Lvov). Не трудно догадаться, что символ " ? " во втором поле этого списка означает символ универсальной подстановки и заменяет собой любую фамилию. Отсюда следует, что правило lvov активизируется столько раз, сколько раз факт (rep ? Lvov) присутствует в базе данных. При этом столько же раз выполнятся действия, содержащиеся в правой части правила. Встроенная функция bind – аналог оператора присваивания. Следовательно, содержимое переменной ?*lvov* увеличивается на единицу и результат сохраняется в этой же переменной. Аналогично работают правила odessa и kiev.

Действия, которые выполняются в последнем правиле программы, отражены в его названии. Правая часть правила особых комментариев не требует, в то время как левая часть заслуживает подробного рассмотрения. В CLIPS существует несколько стратегий очередности выполнения правил, а сами правила могут иметь приоритет, который задается встроенной функцией declare с параметром salience (особенность). Этот параметр может принимать целочисленные значения от –10000 до +10000. По умолчанию для всех правил величина salience равна нулю. Если в правиле result не указать приоритет, оно будет конфликтовать с правилом start за очередность выполнения, так как у этих правил одинаковая левая часть. Для устранения конфликта в правиле result приоритет указан явно и со знаком минус, в связи с чем это правило выполнится последним.

Используя любой текстовый редактор, наберем и сохраним текст программы в ASCII-файле со стандартным для CLIPS-программ расширением .clp и с именем represent. Командой clips вызовем интерпретатор CLIPS, командой (load имя_файла) загрузим в интерпретатор файлы rep и represent.clp, командами (reset) и (run) запустим программу represent.clp на выполнение.

```
CLIPS> (load rep)
.........
TRUE
CLIPS> (load represent.clp)
.........
TRUE
CLIPS> (reset)
CLIPS> (run)

REPRESENTATIVES
from Odessa: 5
from Kiev: 2
from Lvov: 4

CLIPS>
```

Сообщение интерпретатора TRUE означает, что в файле нет синтаксических ошибок и команда загрузки выполнена корректно.Многоточием представлены другие сообщения интерпретатора, которые в данном случае опущены.

Как следует из описанных действий, в интерпретаторе CLIPS находятся два файла. Первый, с именем rep, является базой данных. Второй, с именем represent.clp, содержит сведения (правила) о том, как эти данные могут быть использованы. Таким образом, вместе файлы образуют базу знаний, которая содержит, по крайней мере, два знания. Первое – общий состав участников конференции. Его можно посмотреть, не выходя из интерпретатора по команде (facts). Второе знание – количество участников от каждого города.

В рассмотренном примере база знаний состоит из двух программных модулей. Однако ничто не мешает использовать одну программу, сохраненную в одном файле. В следующем примере показано, как это делается. В нем же эвристический механизм представления знаний используется вместе с процедурным.

Пример 2

Пусть требуется подобрать резистор для участка цепи схемы электрической принципиальной некоторого радиоэлектронного устройства. Резистор характеризуется сопротивлением, которое определяется по измеренным или рассчитанным значениям электрического тока, проходящего через резистор, и падению

напряжения на нем. Программа с именем resistor.clp, решающая эту задачу, может быть, например, такой

```
(deffacts resistors; база данных резисторов
(resistor Ra 2)
(resistor Rb 5)
(resistor Rc 7))
(deffunction om; функция om(x,y)
( ?x ?y)
(div ?y ?x))
(defrule input; начальное правило
(initial-fact)
=>
(printout t crlf "Input current value: ")
(bind ?i (read))
 (printout t "Input strait value: ")
(bind ?u (read))
(assert (numbers ?i ?u)))
(defrule take; подобрать резистор из БД
(numbers ?i ?u)
(resistor ?r =(om ?i ?u))
=>
(printout t crlf "You must take resistor « ?r».” crlf crlf)
(reset)
(halt))
(defrule nothing; если в БД нет подходящего резистора
(numbers ?i ?u)
(resistor ?r ~=(om ?i ?u))
=>
(printout t crlf "There is nothing for You in my database!" crlf
crlf)
(reset)
(halt))
```

Программа состоит из нескольких частей: базы данных с именем resistors, объявления пользовательской функции om и трех правил с именами input, take и nothing.

В базе данных содержатся сведения о резисторах. Они представлены в виде списков, состоящих из трех полей. Первое поле имеет значение resistor, которое отражает тип радиодетали.

Во втором поле списка содержится тип резистора. Последнее поле хранит значение сопротивления.

О функции om подробно говорилось ранее. В данном случае она используется для представления процедурного знания – закона Ома.

Правило input предназначено для ввода исходных данных. Оно активизируется системным начальным фактом и требует от пользователя ввести ток и напряжение. Встроенная функция read возвращает значение, введенное со стандартного устройства ввода (клавиатуры), которое сохраняется в переменных ?i и ?u.

В правой части правила выполняется еще одно действие. Команда assert добавляет в рабочую память интерпретатора CLIPS факт (numbers ?i ?u) для того, чтобы можно было обращаться к локальным переменным ?i и ?u, связанным с правилом input, из других правил программы.

В следующих двух правилах пользователю либо предлагается тип подходящего резистора (правило take), либо сообщается об отсутствии такового (правило nothing).

Рассмотрим правило take. Его левая часть состоит из двух условий, поэтому правило активизируется, если оба условия будут выполнены. Первое условие выполняется, так как соответствующий факт уже создан правилом input. Второе условие выполнится, если будет точно соответствовать какому- либо факту (списку) в базе данных. Первое поле условия вопросов не вызывает. Во втором поле условия находится переменная ?r, которая может принять значение Ra, либо Rb, либо Rc – в зависимости от содержимого третьего поля условия. В этом поле осуществляется вызов функции om и сохраняется возвращаемое функцией значение. Так, если возвращаемое значение будет равно 7, то условие выполнится, переменная ?r примет значение Rc, правило активизируется и выведет на экран монитора предложение выбрать резистор Rc. Если возвращаемое функцией om значение равно 5, то пользователю будет предложен резистор Rb и т.д.

В левой части правила nothing вроде бы полная аналогия – за исключением одной маленькой модификации. В третьем поле второго условия перед вызовом функции om стоит символ " ~ ", означающий логическое отрицание. Таким образом, условие выполнится и правило активизируется, если возвращаемое функцией om значение будет не 2, не 5 и не 7.

Находясь в интерпретаторе CLIPS, командой (clear) очистим его от данных предыдущего примера, загрузим файл resistor.clp и запустим программу на выполнение:

```
CLIPS> (clear)
CLIPS> (load resistor.clp)
.............
TRUE
CLIPS> (reset)
CLIPS> (run)

Input current value: 3
Input strait value: 15
You must take resistor Rb.

CLIPS> (run)

Input current value: 3.5
Input strait value: 5.44
There is nothing for You in my database!

CLIPS>
```

Таким образом, файл resistor.clp также представляет собой базу знаний, поскольку содержит и базу данных, и сведения (правила) о том, как данные могут быть использованы. Эта база располагает, по крайней мере, тремя знаниями. Первое – общий список резисторов с указанием типа и сопротивления. Второе – закон Ома. Третье знание – предлагаемый тип резистора.

Интеллект базы знаний можно существенно повысить добавлением новых данных и правил. Так, вместо закона Ома можно использовать более серьезные методики определения сопротивления резистора.

Другой путь – добавление новых типов резисторов в базу данных. Например, интересный результат получается при внесении в базу данных записи (resistor Rd 2). При некоторой доработке правил take и nothing, если возвращаемое функцией om значение равно 2, правило take отработает два раза и предложит резисторы Ra и Rd. Затем можно пойти дальше: добавить правило (правила), которое выберет из резисторов Ra и Rd

предпочтительный для некоторых конкретных условий и т.д.

Порядок выполнения работы

1. Изучить теоретическую часть.
2. Запустить «Пример 2» и разобраться, как он работает.
3. Разработать на основе примера (файл auto.clp) экспертную систему для требуемой предметной области согласно заданию.
4. Получить распечатки примеров работы программы.
5. Оформить отчет.

Содержание отчета

1. Вариант задания, включающий описание предметной области.
2. Исходный текст разработанной программы.
3. Графическое представление алгоритма работы программы.
4. Распечатки экрана с результатами работы программы.
5. Выводы по работе.

Контрольные вопросы

1. Основные компоненты экспертных систем.
2. Какая форма записи используется в CLIPS для выражений?
3. Как организована база знаний в CLIPS?
4. Какой механизм используется для вывода новых знаний?

ЛИТЕРАТУРА

1. Попов Э. В. Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. – М.: Наука, 1987.

2. Уотермен Д. Руководство по экспертным системам. Пер. с англ. – М.:Мир, 1989.

3. Частиков А. П., Гаврилова Т. А., Белов Д. Л. Разработка экспертных систем. Среда CLIPS. – СПБ: БХВ-Петербург, 2003.