



*Томский межвузовский центр  
дистанционного образования*

**Т.Д. Карминская**

# **ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ**

Учебное пособие

Томск - 2000

Министерство образования Российской Федерации

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ  
УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

Т.Д. Карминская

# **ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ**

Учебное пособие

**2000**

Рецензент:

ведущий специалист отдела информатизации  
Администрации Томской области Омельченко М.В.

**Карминская Т.Д.**

Информационное обеспечение: Учебное пособие. - Томск: Томский  
межвузовский центр дистанционного образования, 2000. - 82 с.

Пособие содержит лекционный материал по курсам информационного обеспечения систем автоматизированного проектирования и систем управления. В пособии рассматриваются основные концепции организации информационного фонда, различные модели данных, вопросы проектирования баз данных, языковые средства систем управления базами данных. Рассмотрена диалоговая среда реляционной СУБД FoxPro.

© Карминская Татьяна Дмитриевна, 2000

© Томский межвузовский центр  
дистанционного образования, 2000

## СОДЕРЖАНИЕ

ГЛАВА 1. Основные понятия. Информация и данные .....	5
1.1. Понятия информационного обеспечения. Специфика данных САПР. ....	5
1.2. Состав и способы ведения информационного фонда САПР .....	5
1.3. Этапы представления данных. Организация информационного обеспечения на основе концепции баз данных. ....	7
ГЛАВА 2. Архитектура систем баз данных. Уровни представления данных. Модели данных .....	9
2.1. Трехуровневая архитектура представления данных .....	10
ГЛАВА 3. Стандарты СУБД .....	16
3.1. Принципы разработки СУБД. ....	16
3.2. Обобщенное определение СУБД. ....	17
ГЛАВА 4. Представление данных .....	18
4.1. Основные определения .....	18
4.2. Виды взаимосвязей между данными. ....	19
4.3. Основные модели данных. ....	20
4.3.1. Иерархические модели данных. Описание и терминология ..	21
4.3.2. Сетевые системы .....	24
4.3.3. Отличия иерархий от сети .....	26
4.3.4. Реляционные модели. Реляционные СУБД. ....	27
ГЛАВА 5. Базисные средства манипулирования реляционными данными. ....	29
5.1. Реляционная алгебра. ....	29
5.2. Реляционное исчисление. ....	34
ГЛАВА 6. Проектирование баз данных. ....	35
6.1. Функциональные зависимости. ....	35
6.2. Теория нормальных форм. ....	37
6.2.1. Первая нормальная форма (1НФ). ....	37
6.2.2. Вторая нормальная форма (2НФ). ....	38
6.2.3. Третья нормальная форма (3НФ). ....	39
6.2.4. Четвертая нормальная форма (4НФ). ....	40
6.3. Языковые средства СУБД. ....	41
6.3.1. Языки описания данных. ....	41
6.4. Язык запросов SQL. ....	48
6.5. Язык запросов QBE. ....	61
ГЛАВА 7. Система FoxPro .....	64
7.1. Основные ограничения .....	64
7.2. Типовые технологии создания и обработки файлов данных .....	67

7.3. Диалоговая среда системы FoxPro .....	70
7.3.1. Блоки диалога .....	70
7.4. Опции главного меню системы FoxPro .....	74
7.4.1. Меню File .....	74
7.4.2. Меню Database .....	76
7.4.3. Меню Record .....	80
7.4.4. Меню Program .....	80
7.4.5. Меню Window .....	81
8. Список рекомендуемой литературы .....	82

## ГЛАВА 1. ОСНОВНЫЕ ПОНЯТИЯ. ИНФОРМАЦИЯ И ДАННЫЕ

### 1.1. Понятия информационного обеспечения. Специфика данных САПР

Система автоматизированного проектирования (САПР) - сложная и многокомпонентная система, процессы преобразования данных в которой разнообразны. Это приводит к различным трактовкам термина "данные" для различных компонент САПР:

- для *управляющего монитора* САПР в состав данных входит совокупность программных модулей, которые реализуют функции проектирования;
- для *системы диалогового обеспечения* САПР данными являются множество взаимосвязанных информационных и управляющих кадров экрана дисплея;
- для *функциональных программных модулей* к данным относится совокупность исходных и результирующих чисел, необходимых для выполнения конкретной проектной процедуры;
- пользователю САПР в качестве данных требуется иметь в своем распоряжении исходную проектную документацию, справочные данные, типовые проектные решения и т.д.

При этом данные, являющиеся *результатом* одного процесса преобразования, могут быть *исходными* для другого процесса (промежуточные данные).

Для обозначения совокупности данных, используемых всеми компонентами САПР, введем понятие **информационный фонд** (ИФ) САПР.

Ведение информационного фонда, т.е. обеспечение создания, поддержки и организации доступа к данным – функция *информационного обеспечения САПР* (ИО САПР). Это важное понятие для вопросов ведения данных и именно так называется учебный курс, который Вам предстоит изучить.

Назначение ИО САПР - реализация информационных потребностей всех основных компонентов САПР. **Информационное обеспечение САПР** - есть совокупность информационного фонда и средств его ведения.

### 1.2. Состав и способы ведения информационного фонда САПР

Проектирование - связанная совокупность процессов преобразования одних данных в другие. Важно выяснить состав, назначение и физическую организацию структур данных, составляющих информационный фонд:

1. *Программные модули* - хранятся в виде символических и объектных текстов. Как правило, эти данные мало изменяются в течение жизненного цикла САПР, имеют фиксированные размеры и появляются на этапе создания

информационного фонда САПР. Потребителями этих данных являются мониторы (управляющие программы) различных подсистем САПР.

2. *Исходные и результирующие данные* необходимы при выполнении программных модулей в процессе преобразования. Эти данные часто меняются в процессе проектирования, однако их тип постоянен и полностью определяется соответствующим программным модулем. При организации промежуточных данных возможны конфликтные ситуации в процессе согласования между собой данных различных типов.

3. *Нормативно-справочная проектирующая документация (НСПД)* – включает в себя справочные данные о материалах, элементах схем, унифицированных узлах и конструкциях. Эти данные, как правило, хорошо структурированы и могут быть отнесены к фактографическим. Кроме того, к НСПД относятся государственные и отраслевые стандарты (ГОСТы ОСТы), руководящие материалы и указания, типовые проектные решения, регламентирующие документы (слабоструктурированные документальные данные).

4. *Содержание экранов дисплеев* - представляет собой последовательную совокупность данных, задающих форму кадра, и, следовательно, позволяющих отображать на экран дисплея информацию с целью организации диалогового взаимодействия в ходе проектирования. Обычно эти данные не изменяются в течение жизненного цикла САПР, имеют фиксированный размер и по своим характеристикам занимают промежуточное место между программными модулями и исходными данными.

5. *Текущая проектная документация* - отражает состояние и ход выполнения проекта. Как правило, эти данные слабоструктурированы, часто изменяются в процессе проектирования и представляются в форме текстовых документов.

Проблему организации и ведения данных нужно рассматривать в двух аспектах:

- *содержательного* - интерес представляет причина возникновения необходимости тех или иных данных, получение его значения, определяется принятой методикой проектирования, разработанными алгоритмами решения частных задач. Эти вопросы носят общеметодологический характер;
- *организационного* - определяются принципы и способы ведения информационного фонда, структурирования данных, алгоритмы управления массивами данных. В настоящее время существуют следующие **способы ведения информационного фонда САПР**:
- использование файловой системы; построение библиотек; использование банков и баз данных; создание информационных программ адаптеров.

Каждый из этих способов имеет свои достоинства и недостатки и ориентирован на конкретный класс задач.

Использования файловой системы и библиотек достаточно широко распространены в организации информационного обеспечения, они представляют собой традиционный подход - т.е. такой, когда каждый из программистов создает свой файл данных. Подход, основанный на "независимых файлах" имеет ряд серьезных недостатков:

- избыточность информации;
- нерациональное расходование ресурсов (времени, затрачиваемое на ввод дублей и памяти для их хранения);
- возможности возникновения противоречий (например, разные обозначения одной и той же физической величины).

Эти и другие проблемы могут быть успешно разрешены путем использования банков и баз данных, которые позволяют:

- централизовать информационный фонд САПР;
- произвести структурирование данных в виде, удобном для проектировщика;
- обеспечить поиск информативно-справочной литературы и проектной документации;
- упростить организацию межмодульного интерфейса путем унификации промежуточных данных.

Для класса задач, которые решаются в сложных системах управления и проектирования, наибольшее распространение получил способ ведения данных, основанный на *концепции баз данных*. Именно такой подход обеспечивает *многоуровневую организацию данных*, когда в процессе проектирования данных задействованы как заказчики-пользователи, так и разработчики-программисты. *Т.о. организация данных ведется на двух уровнях, каждый из которых характеризуется своей моделью данных*

### **1.3. Этапы представления данных. Организация информационного обеспечения на основе концепции баз данных**

*Модель данных - это абстрактное описание объектов и их взаимосвязей, (которое дает возможность увидеть "лес" информационного содержания данных, а не "отдельные деревья" -конкретные значения данных).*

В зависимости от уровня семантической (содержательной) интерпретируемости, обеспечиваемой используемой моделью, выделяют следующие модели данных:

- **инфологические** (информационные);
- **датологические** (физические).

Эти модели характеризуются своими аспектами.



**Инфологический аспект** употребляется при рассмотрении вопросов, связанных со смысловым содержанием данных независимо от способов их представления в памяти системы, реализуется на информационном этапе проектирования данных.

На этапе инфологического проектирования информационной системы должны быть решены вопросы:

- о каких объектах или явлениях реального мира требуется накапливать и обрабатывать информацию в системе;
- какие их основные характеристики и взаимосвязи между собой будут учитываться;
- уточнения вводимых в информационную систему понятий об объектах "явлениях", их характеристиках и взаимосвязях.

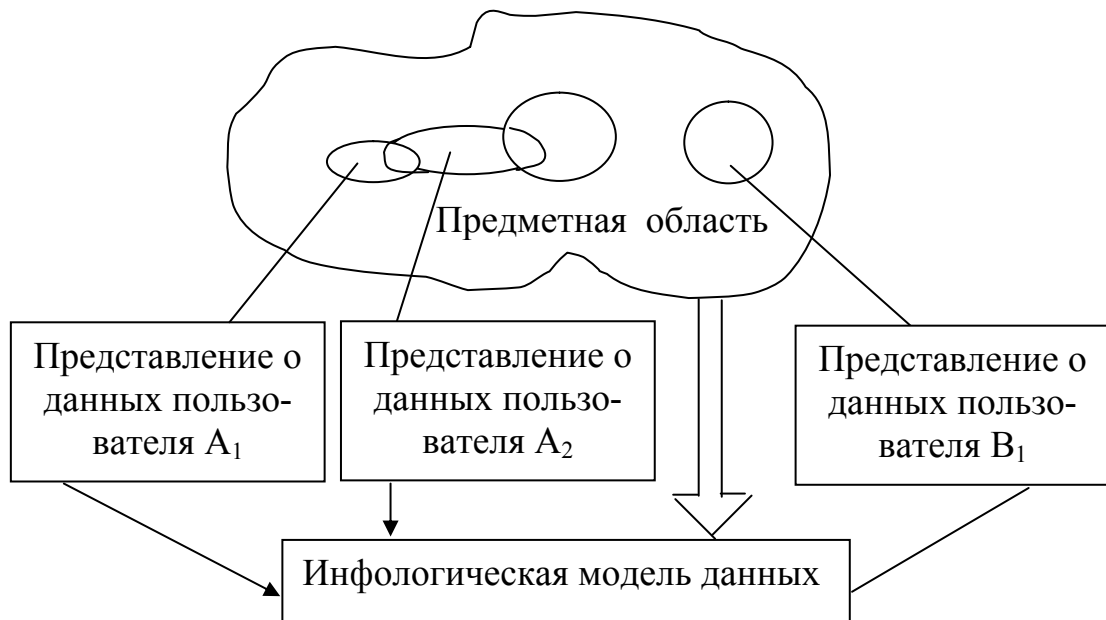


Рис. 1.1. Формирование инфологической модели данных

Описание предметной области, выполненное с использованием естественного языка, математических формул, таблиц, и т.д. называется **инфологической моделью**. Такая человеко-ориентированная модель полностью независима от физических параметров среды хранения данных. Инфологическая модель не должна изменяться до тех пор, пока не будет изменений в реальном мире (рис. 1.1). Представление инфологической модели данных с учетом правил порождения структур данных подводит к понятию **ДАТОЛОГИЧЕСКОЙ МОДЕЛИ ДАННЫХ** и соответствующего аспекта.

**Датологический аспект** употребляется при рассмотрении вопросов представления данных в памяти информационной системы.

Датологические модели отражают специфику той физической среды, в которой они реализуются, а сами данные соответствуют зарегистрированным фактам об объектах или явлениях реального мира. Чтобы в дальнейшем ис-

пользовать данные, требуется их *смысловое содержание - семантика*. Поэтому в информационной системе должны быть сформулированы правила смысловой интерпретации данных, которые реализуются средствами языка программирования.

*Датологическая модель - это формальное описание инфологической модели, выполненное на языке описания данных. Т.е. компьютеро-ориентированная модель, которая позволяет программам и конечным пользователям осуществлять доступ к хранимым данным лишь по именам, совершенно не заботясь о физическом расположении искомых данных.*

Форма преобразование инфологической модели в датологическую не является однозначной. *Физическая реализация инфологической модели данных может быть осуществлена с использованием разных языков программирования и/или разных систем управления базами данных (СУБД) и не является единственно возможной.*

*В структурном плане модель данных* характеризуется как совокупность взаимосвязанных понятий и правил порождения структур данных, ограничений целостности (логико-семантических свойств данных), а также операций над ними.

Теория моделей данных в своем развитии использует концепцию баз данных.

**Концепция баз данных** – связана с проблемами обеспечения логико-семантической и физической целостности, а также эффективности многоцелевого использования данных в условиях непрерывной и естественной эволюции приложений – процессов, оперирующих данными.

Конкретная реализация этой концепции приводит к архитектуре *многоуровневого представления данных*, которая и является *отличительным признаком БД*.

**База данных** – совокупность данных и метаданных, т.е. описаний свойств данных, интерпретируемая в среде специальной программной системы – системы управления базами данных, (СУБД) – как многоуровневая организация, обеспечивающая независимость представления данных на различных уровнях.

## ГЛАВА 2. АРХИТЕКТУРА СИСТЕМ БАЗ ДАННЫХ. УРОВНИ ПРЕДСТАВЛЕНИЯ ДАННЫХ. МОДЕЛИ ДАННЫХ

Выше отмечалось, что *организация данных ведется на двух уровнях, каждый из которых характеризуется своей моделью данных.*

*Возникновение проблемы определения требований логического проектирования и реализации базы данных в рамках всего жизненного цикла базы данных, вызвало необходимость введения концептуального этапа проекти-*

рования данных. Таким образом, возникла необходимость введения в рассмотрение трехуровневой архитектуры представления данных.

## 2.1. Трехуровневая архитектура представления данных

В соответствии с предложениями исследовательской группы по системам управления базами данных Американского национального института стандартов, выделяют три уровня представления данных:

- концептуальный;
- внешний;
- внутренний.

**Концептуальный уровень представления данных** в форме концептуальной модели данных (схемы КМД) отражает объективные, не зависящие от конкретного приложения свойства данных, описывающих предметную область. Его еще называют "промежуточный уровень" между двумя другими (внешним и внутренним) уровнями.

**Внешний уровень представления данных (схема ВМД)** - отражает субъективные взгляды приложений на данные.

В большинстве практических случаев внешнее представление ограничивается выделением некоторого подмножества концептуального представления.

Т.е. внешний уровень наиболее близок к пользователям, т.е. связан с тем, как отдельные пользователи представляют себе эти данные. Поэтому внешняя модель состоит из различных типов "внешних записей".

**Внутренний уровень представления данных (ВнМД)** - определяет машинно-ориентированное, физическое представление данных.

Т.е. внутренний уровень наиболее близок к физической памяти, т.е. связан со способом *фактического хранения данных*.

**ВЫВОД:** если *внешний* уровень связан с **частными представлениями пользователей**, то *концептуальный уровень* - **обобщенное представление пользователей**. Другими словами, может быть много "внешних представлений", каждое из которых состоит из более или менее абстрактного представления некоторой части базы данных, и может быть единственное "концептуальное представление", состоящее из абстрактного представления базы данных в целом. (Действительно, большинство пользователей интересуется не вся база данных, а лишь некоторая ограниченная её часть).

Также есть *единственное "внутреннее представление"* отражающее всю базу данных, как действительно хранимую.

### Трехуровневая архитектура систем баз данных

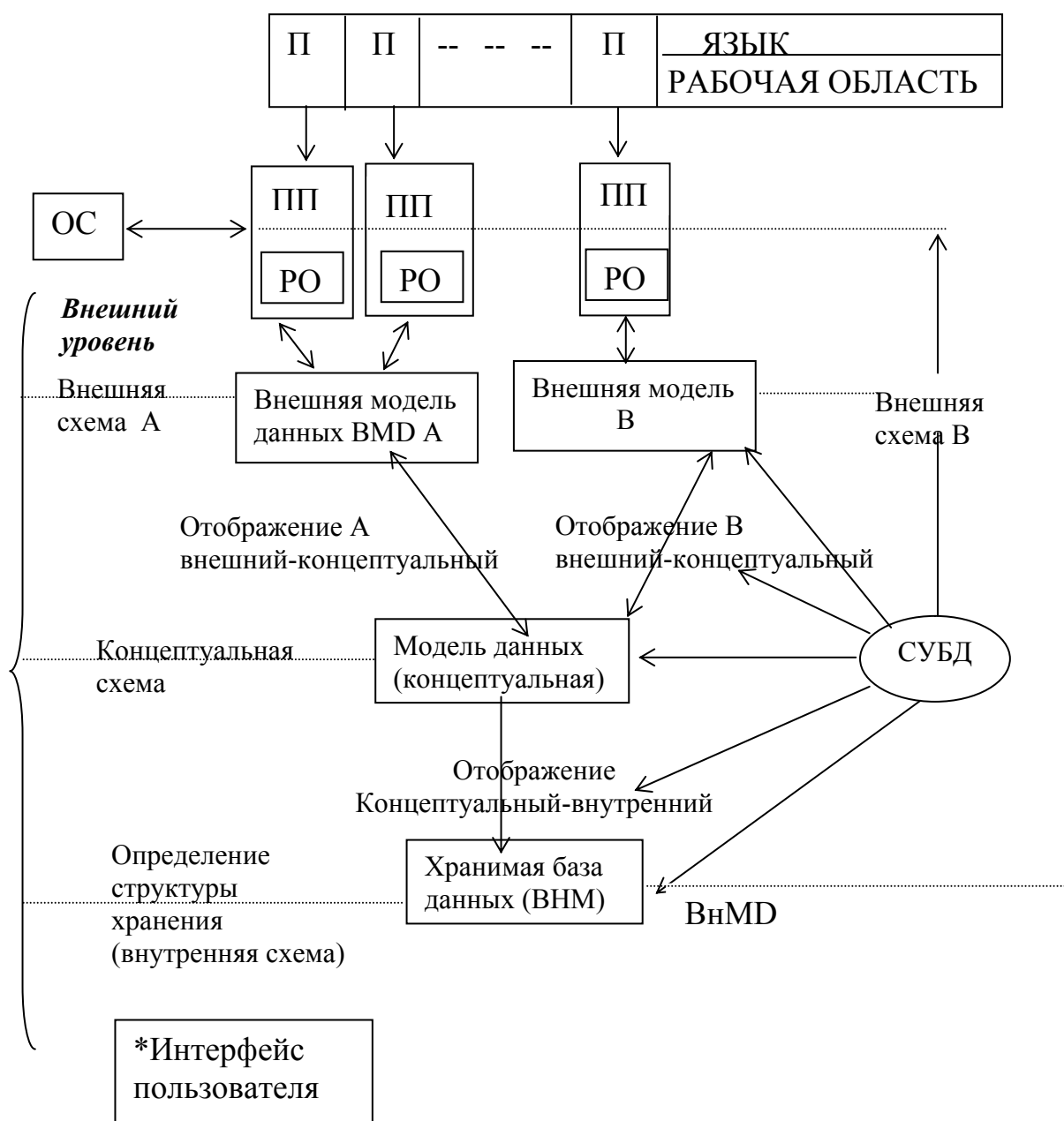


Рис. 2.1

## Анализ архитектуры

**Пользователи системы (ПП):** являются либо прикладные программисты, либо пользователи, работающие с удаленных терминалов и имеющие различный уровень профессиональной подготовки.

В распоряжении каждого пользователя есть **язык общения**:

- для прикладного программиста - это язык программирования;
- для пользователя - язык, разработанный с учетом его потребностей.

Каждый язык пользователя включает **подъязык данных** - подмножество языка пользователя для выборки и запоминания информации в базе данных. Мы говорим, что *подъязык данных* содержится в **базовом языке**, хотя для пользователя оба этих языка неразличимы. Фактически, по отношению к существующим языкам программирования, элементы **подъязыка данных** - есть обращение к стандартным подпрограммам.

### **Рабочая область (РО).**

Каждый пользователь обеспечивается **рабочей областью, необходимой для приема или передачи данных между пользователем и базой данных**. (Для прикладного программиста рабочая область является область ввода-вывода, для пользователя - видеотерминал).

### **Внешний уровень.**

Запросы пользователя (П) БД в виде отдельных приложений - прикладных программ (ПП), записанные в терминах внешней модели данных и отражающие их конкретные нужды, поступают в систему и обрабатываются СУБД и операционной системой (ОС) ЭВМ. Тем самым составляется внешний уровень, который состоит из различных экземпляров различных типов **внешних записей**.

### **Концептуальный уровень.**

**Концептуальная модель** - есть представление полного информационного содержания базы данных в несколько абстрактной форме по сравнению со способом их физического представления.

Концептуальная модель состоит из множества экземпляров различных типов **концептуальных записей**. Концептуальная модель определяется посредством **концептуальной схемы**, которая включает определения каждого типа концептуальных записей.

Для достижения независимости данных эти определения не должны каким-либо образом учитывать структуру хранения или стратегию доступа, они должны быть определениями **информационного содержания**.

Таким образом, **концептуальная модель** есть представление общего содержания базы данных, а **концептуальная схема** информационное определение этого представления.

### **Внутренний уровень.**

**Внутренняя модель** есть представление самого низкого уровня всей базы данных: он состоит из различных экземпляров типов **внутренних записей**.

Термин ANSI (Американский национальный институт стандартов) "**внутренняя запись**" применяется для конструкции, которую мы называем "**хранимой записью**". Таким образом, внутренняя модель является шагом в сторону от физического уровня, т.к. она не строится в терминах физических записей или полей.

Внутренняя модель описывается посредством **внутренней схемы**, которая не только определяет различные типы хранимых записей, но и то, какие индексы существуют, как представлены хранимые поля, какова физическая последовательность хранимых записей.

Т.е. хранимая база данных есть внутренняя схема.

### **Уровни отображения**

#### **Отображение "концептуальный внутренний"**

- определяет соответствие между моделью данных (концептуальная МД) и хранимой базой данных;
- указывает, как концептуальные записи и поля отображаются в их хранимые копии.

Если структура хранимой базы данных изменяется, т.е. если изменяется определение структуры хранения, отображение "концептуальный внутренний" должно быть соответственно изменено так, чтобы концептуальная схема оставалась неизменной. Влияние подобных изменений должно быть нейтрализовано ниже концептуального уровня, с тем, чтобы могла быть достигнута **независимость данных**.

Отображение "**внешний - концептуальный**" - определяет соответствие между конкретной внешней моделью и моделью данных (концепт.).

В общем случае между двумя этими уровнями могут существовать те же виды различий, что между моделью данных и базой данных. Например, поля могут иметь различные типы данных; различные внешние модели, и др. Описание отображения и есть **схема систем управления базой данных (СУБД)** - программа, которая управляет всем доступом к базе данных.

### **Схема действия СУБД**

- Пользователь выдает запрос на доступ, используя конкретный подязык данных (ПЯД);
- СУБД воспринимают запрос и интерпретирует его;
- СУБД обследует по очереди внешнюю схему, отображение "внешний-концептуальный", концептуальную схему, отображение "концептуальный-внутренний" и определение структуры хранения;

- СУБД выполняет необходимые операции над хранимой базой данных.

### **Администратор базы данных (АБД)**

**АБД** - это человек или группа людей, вооруженных специальными программными средствами и отвечающих за взаимодействие между пользователями и системой, создание и обслуживание БД, оперативную поддержку защиты и целостности и эффективную работу системы.

### **Функции АБД**

1. Определение информационного содержания БД.

АБД определяет, какая информация должна содержаться в БД, т.е. он идентифицирует объекты представляющие интерес для пользователя. Т.е. АБД составляет концептуальную схему (используя специальный язык).

**Объектная** (скомпилированная) форма этой схемы будет использована СУБД при ответе на запросы.

**Исходная** (не компилированная) форма используется как справочный документ пользователя системы.

2. Определение структуры хранения и стратегия доступа. АБД должен представить данные в базе, написав определение структуры хранения.

Дополнительно должно быть задано соответствующее отображение между определением структуры хранения и концептуальной схемой.

3. Взаимодействие с пользователями. Установление связи с пользователями посредством написания внешних схем. Каждая внешняя схема, подобно концептуальной будет существовать как в исходной, так и в объектной форме.

4. Определение контроля полномочий и процедур проверки достоверности.

Эти процедуры рассматриваются как логическое продолжение концептуальной схемы.

5. Определение стратегии дублирования и восстановления БД.

Эффективность работы любого предприятия сильно зависит от этой функции, например:

- доступность к данным, которые не были повреждены, должна быть сохранена;
- периодическая разгрузка данных на резервные магнитные ленты и процедуры перезагрузки соответственно в части БД с последней магнитной ленты.

6. Управление эффективностью и реакция на изменения в требованиях.

Научная организация системы и настройка системы при изменении требований к ней. (Т.е. при любых изменениях концептуальная схема должна оставаться неизменной). Т.е. должны существовать следующие программы.

### Программное обеспечение АБД

- Программы загрузки (для создания первоначальной версии БД);
- программы реорганизации (для реорганизации ДБ с целью освободить место, занятое ненужными данными);
- программы ведения журнала (для внесения сведений, о каждой операции, использующей базу данных, и о пользователе, выполнившем эту операцию, а также для записи состояния до и после изменения);
- программы восстановления (для восстановления БД в первоначальное состояние после сбоев оборудования или программ);
- программы анализа статистики (для управления производительностью системы).

Одним из наиболее важнейших инструментов базы данных является **словарь данных (data dictionary)**.

Словарь данных сам является БД, которая содержит «данные о данных» или, другими словами, описания других объектов в системе.

- 1) В словаре физически хранятся все схемы, как в исходной, так и в объектной форме, а также определения отображений, процедуры проверки достоверности.
- 2) Находится информация о перекрестных ссылках, показывая например какие программы используют какие массивы данных.

Словарь позволяет определять влияние любых изменений в системе.

**Интерфейс пользователя** – это граница в системе, за которой пользователь ничего не видит, т.е. является внешним уровнем.

Подводя итог сказанному выше можно сделать вывод, что в целом представление пользовательского интерфейса является назначением СУБД и схематически сущность СУБД можно представить следующим образом:



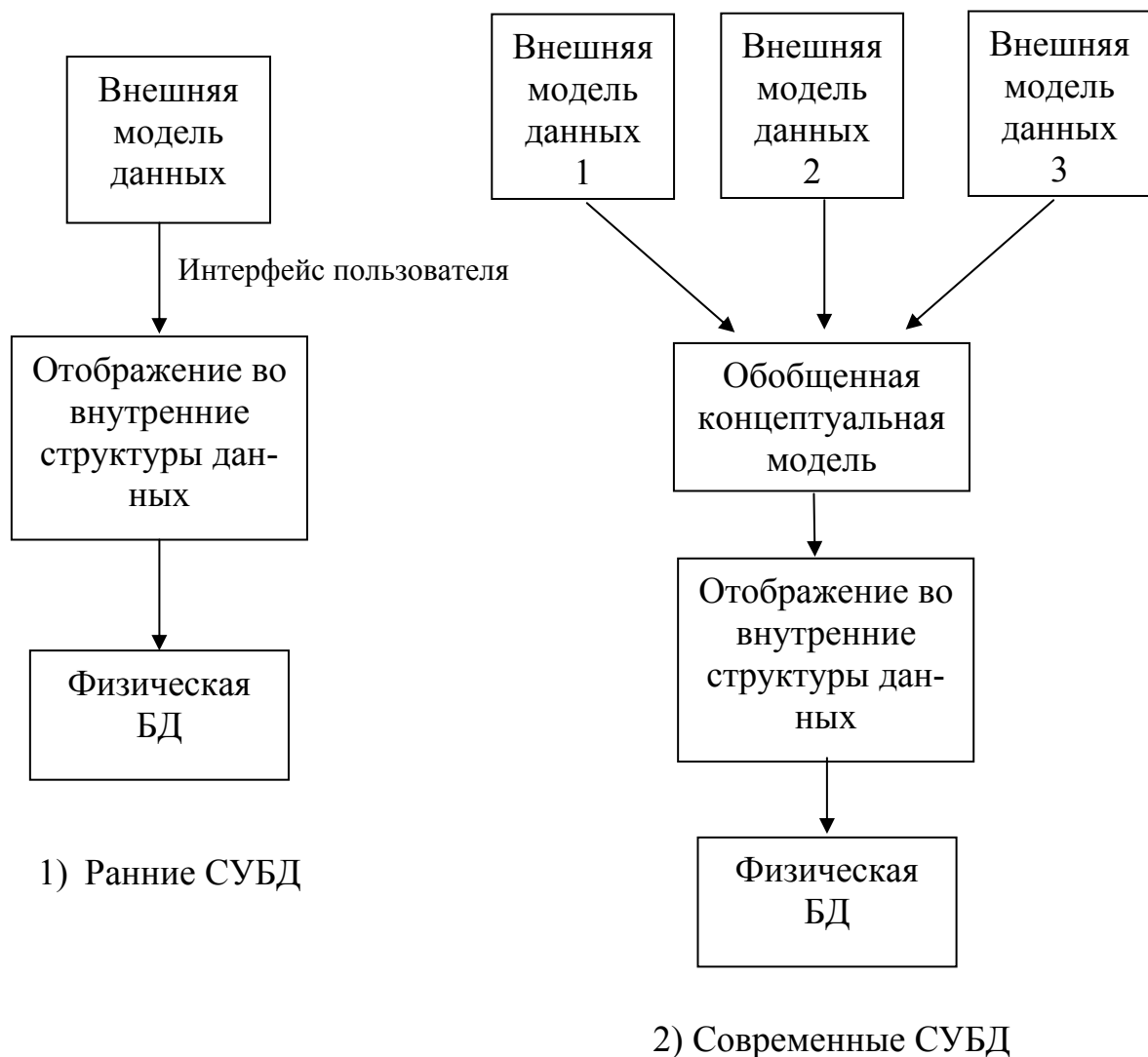


Рис. 2.2

## ГЛАВА 3. СТАНДАРТЫ СУБД

Современные системы управления базами данных (СУБД) разработаны с целью устранения недостатков файловых систем. СУБД-это программная система, являющаяся связующим звеном между пользователем и базой данных(БД), представляет пользователю возможность работы с данными, не вникая в детали на уровне аппаратного обеспечения.

### 3.1. Принципы разработки СУБД

Принципы разработки СУБД представляют собой перечень и обоснование свойств, которыми должны обладать системы управления базами данных, способные реализовать многоуровневую организацию данных и манипулиро-

вание ими. Приведем основные свойства СУБД и их функциональную интерпретацию (назначение):

1) **Независимость данных** – СУБД должна обеспечивать независимость пользователя от организации физических данных.

Изменения физической организации и/или параметров запоминающих устройств воспринимаются СУБД и не влияют на пользователя или, точнее, на прикладную программу. А также изменение пользовательского представления и/или добавление нового представления также поддерживается СУБД и не требует затрат на реорганизацию и изменение механизма доступа к файлам физических данных.

**Независимость данных** – это возможность функционирования с обеих сторон (т.е. со стороны пользователя и физических данных).

2) **Универсальность**. СУБД должно обладать мощными средствами поддержки концептуальной модели данных для отображения пользовательских логических связей.

3) **Совместимость**. СУБД должна сохранять работоспособность при развитии программного и аппаратного обеспечения.

4) **Не избыточность данных**. В отличие от файловых систем БД должна представлять собой единую совокупность интегрированных данных. **Не избыточность=интеграция**.

5) **Защита данных**. СУБД должна обеспечивать защиту от несанкционированного доступа.

6) **Целостность данных**. СУБД должна предотвращать нарушения базы данных пользователями, т.е. поддержка согласованности одних и тех же данных при работе различных пользователей.

7) **Управление одновременной работой**. СУБД должна предохранять базу данных от рассогласований в режиме коллективного пользования.

Для обеспечения согласованного состояния БД все **транзакции** (запросы пользователя на изменение, удаление или добавление данных) должны выполняться в определенном порядке.

### 3.2. Обобщенное определение СУБД

**СУБД** – это набор программных средств, позволяющих:

1) Обеспечить пользователей языковыми средствами определения и манипулирования данными. Подобными средствами являются:

ЯОД – язык определения данных (DDL-Data Direct Language);

ЯМД – язык манипулирования данными (DML- Data Managment Language).

Термин **язык данных** обозначает либо оба, либо один из названных языков, а слово **данные** отличает язык данных от других типов языков (PL1, Паскаль). Язык данных может быть включен в универсальный язык. В этом

случае универсальный язык программирования и язык данных называется соответственно **включающим языком и подязыком данных (ПЯД)**.

**Автономный язык данных** – (т.е. не включенный в универсальный язык), называют **языком запросов**.

2) Обеспечивать поддержку моделей данных пользователя.

Модель данных – это средство для определения логического представления физических данных, относящихся к некоторому приложению.

3) Обеспечить программу, реализующую функции ЯОД (языка определения данных) и ЯМД (языка манипулирования), допускающую определение, создание и манипулирование логическими данными (т.е. выборку, обновление, включение и удаление). Эта программа отображает перечисленные операции в соответствующие операции над физическими данными.

4) Обеспечить защиту и целостность данных.

## ГЛАВА 4. ПРЕДСТАВЛЕНИЕ ДАННЫХ

### 4.1. Основные определения

При обработке информации человек начинает процесс с формирования понятий об интересующих его фактах, явлениях, предметах и событиях.

Для обозначения прообраза понятия любой природы используется термин **СУЩНОСТЬ** (множество однотипных предметов реального мира - множество служащих, студентов).

Сущность в свою очередь характеризуется своими основными свойствами: примерами свойств сущности "студент" является фамилия, факультет, номер группы и т.д.

В информационной модели сущность представляется **типом записи** - это определенный формализм, который позволяет логическое представление сущности описывать в терминах структуры данных информационной модели.

Термин **тип записи** используется всегда при информационном представлении сущности, хотя тип записи в различных моделях может играть различную роль:

Например:

- в реляционной модели - типу записи соответствует отношение;
- в иерархической модели - сегмент или узел;
- в сетевой модели - записи - владельцы или записи - члены.

**Свойства, характеризующие сущность, называются АТРИБУТАМИ.**

Рассмотрим физическую структуру, соответствующую логическому представлению, она описывает данные, хранящиеся в памяти ЭВМ. Терминология, относящаяся к этому уровню представления, заимствована из традиционных файловых систем.

- **Тип записи** реализуется как множество экземпляров записей, один экземпляр записи (**или просто запись**) содержит данные об одном объекте (например, об одном студенте).
- **Сущность** (или объектное множество), представляет тип записи "студент", множество сходных объектов, обладающих одним и тем же набором **свойств (или) атрибутов**. Термин, эквивалентный термину **файл - тип записи**.
- Экземпляр записи состоит из элементов данных, которые соответствуют значениям атрибутов (**студент** - это тип записи, **группа** - это атрибут, **588** - это элемент данных или значение атрибута.)

#### 4.2. Виды взаимосвязей между данными

Пусть имеются множество  $\{A\}$  и  $\{B\}$ . Отношение  $A R B$  указывает на связь между отдельными элементами этих множеств.

Связь - это соответствие или отображение между элементами двух (или более) множеств.

Существуют связи: **межатрибутные** - внутрисущностный тип, **межсущностные** - "ассоциации".

##### МЕЖАТРИБУТНЫЕ:

Сущность	свойства
студенты	номер группы номер студента ф.и.о.

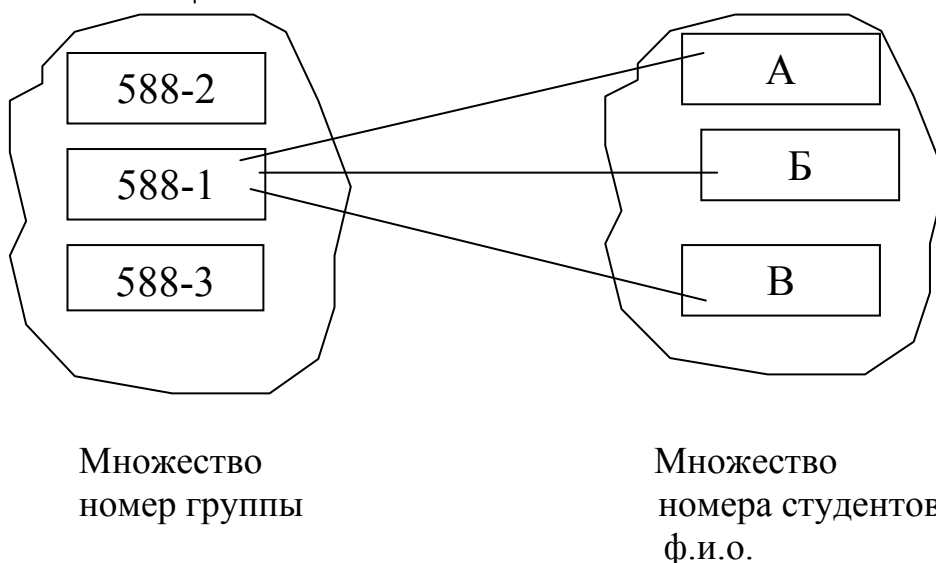


Рис. 4.1

## МЕЖСУЩНОСТНЫЕ:

<u>Сущности</u>	<u>Свойства</u>
Поставщики	номер поставщика
Детали	имя город номер детали, описание, наличие

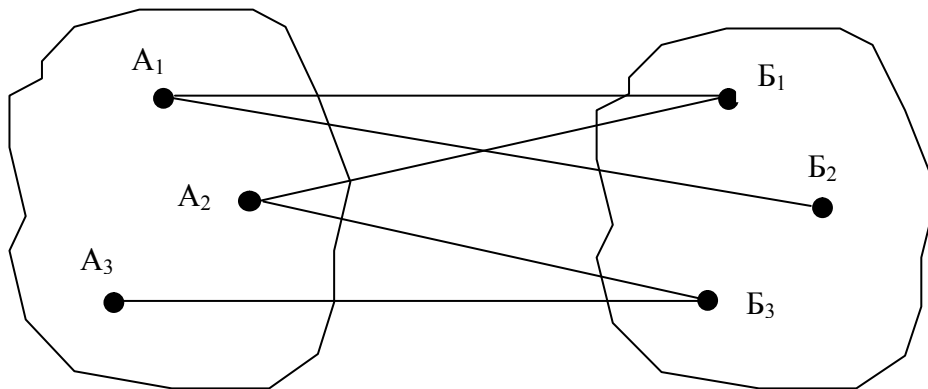


Рис. 4.2.

**По способам отражения связей** между данными на логическом уровне различают модели: иерархическую, сетевую, реляционную.

Модель называют **сетевой**, если данные и их связи имеют структуру графа.

Если структура отражаемых связей представляется в виде дерева, то модель называют **иерархической**.

Представление данных в форме таблиц соответствует **реляционной модели данных**.

### 4.3. Основные модели данных

В СУБД данные организованы таким образом, чтобы пользователи и прикладные программы могли работать с данными по определенным алгоритмам. Организация данных и способы доступа к ним определяют такие алгоритмы и представляет суть модели данных. Любая конкретная СУБД ориентирована на определенную модель данных и поддерживает эту модель. Поэтому существует понятие СУБД иерархической, сетевой либо реляционной модели (системы).

*Вид модели определяется типом связей между данными.* Существуют отображения, которые реализуют тип связи между данными:

- один к одному (1:1);

- один ко многим (1:M);
- многие ко многим (M:N).

Рассмотрим основные модели.

#### 4.3.1. Иерархические модели данных. Описание и терминология

Основным типом логической структуры, поддерживаемой иерархической (СУБД), является иерархия (или дерево), поэтому иерархические структуры зачастую обозначаются как древовидные структуры

Дерево представляет собой иерархию элементов, называемых **узлами**.

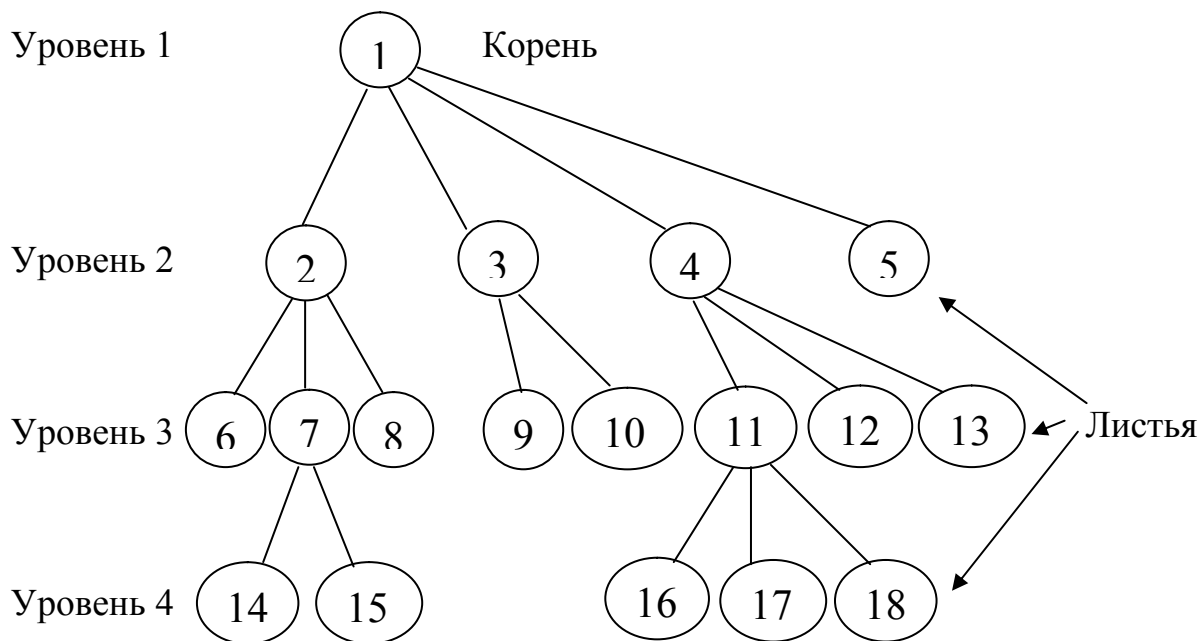


Рис. 4.3

1) На самом верхнем уровне иерархии имеется только один узел – корень.

2) Каждый узел (кроме корня) связан с одним узлом на более высоком уровне, называемым **исходным узлом** для данного узла. Ни один элемент не имеет более одного исходного.

3) Каждый элемент может быть связан с одним или несколькими элементами на более низком уровне. Они называются **порожденными** (или зависимыми).

4) Элементы, расположенные в конце ветви, т.е. не имеющие порожденных элементов, называются **листьями**.

**Термины описания диаграммы дерева:**

- 1) Высота - число уровней (на рисунке равна 4).
- 2) Момент - число узлов (равен 18).
- 3) Вес - число листьев (равен 12).
- 4) Основание- число корней (в иерархических системах, как правило, равно 1).

Порядок хранения записей в физической базе данных зависит от особенностей реализации, определяющих способ отображения логических деревьев в физические структуры файлов конкретной системы.

Наиболее простой способ состоит в **линеаризации** дерева, при этом экземпляры типов записей (узлов) хранятся последовательно: в порядке обхода дерева сверху вниз слева направо.

Причем, если разность уровней двух связанных узлов равна единице, то связь является **непосредственной** (т.е. без промежуточных узлов).

Если любые две вершины дерева, принадлежащие одной ветви, **транзитивно** связаны друг с другом – связь является опосредованной

Для обозначения свойств иерархических систем, необходимо ввести понятия:

**функциональная связь** (или отображение) - это связь, удовлетворяющая определению математической функции, а именно любому аргументу соответствует конкретное значение функции;

**кардинальное число** - число элементов в экземпляре связи.

**Свойства иерархической системы:**

- 1) Иерархическая структура реализует отображения  $N:1$  между порожденным и исходным типом записей.
- 2) Отображение (связи) в иерархических системах функционально, т.к. дерево не может содержать порожденный узел без исходного узла (за исключением корня). Следовательно, отображения  $1:1$  и  $1:N$  могут непосредственно представляться иерархическими структурами.
- 3) Для представления отображения типа  $M:N$  необходимо дублирование деревьев.

В базе «поставщики-детали» имеется отображение  $M:N$  между типами записей. Если неизвестен наиболее часто используемый тип записи, то в целях общности и гибкости необходимо при наличии ресурсов памяти хранить оба варианта.

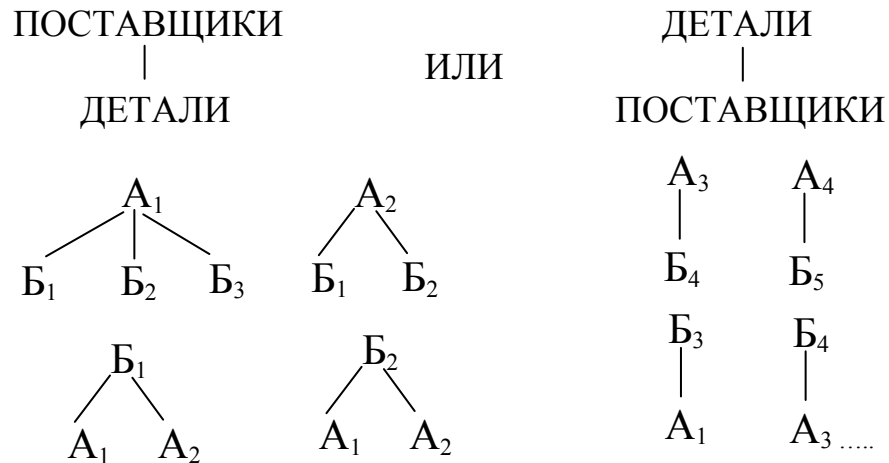


Рис. 4.4

Следовательно, в иерархических СУБД реализация сложных связей требует дублирования данных и/или больших затрат на поиск.

4) В иерархических СУБД способ реализации поиска ориентирован на древовидную структуру. В связи с этим поиск начинается с корня и продолжается в направлении порожденных узлов.

Если в конкретной реализации кроме последовательного просмотра всего дерева отсутствует какой-либо способ прямого доступа к конкретному типу записи, то положение узла в дереве имеет важное значение для доступа к нему. Другой проблемой иерархий является невозможность хранения в базе данных порожденного узла без соответствующего исходного, в этом случае необходимо ввести **пустой исходный узел, что приводит к возникновению проблемы избыточности**.

Соответственно удаление данного исходного узла влечет удаление всех порожденных узлов (поддеревьев), связанных с ним.

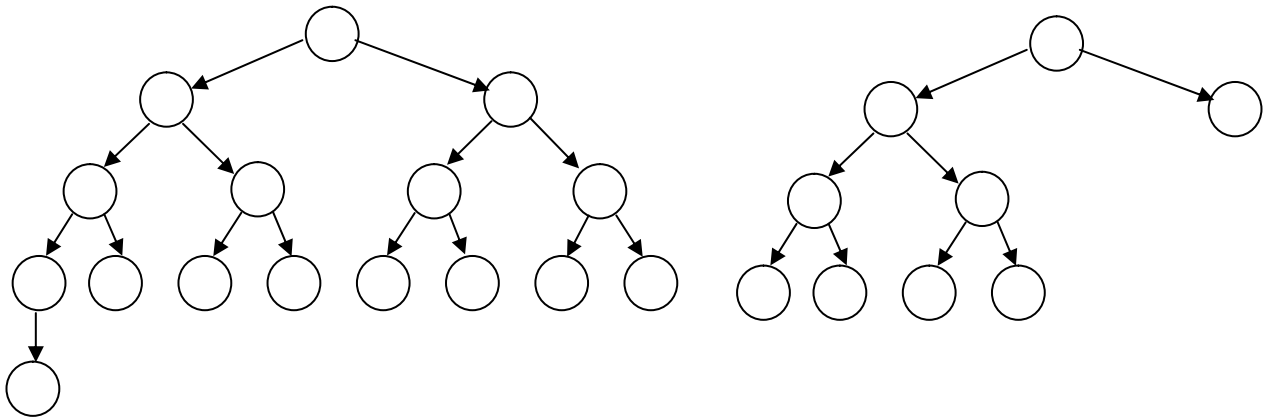
Эти ограничения создают проблемы для некоторых приложений, усложняя процесс проектирования схемы.

При физической организации данных иерархических систем вводится понятие *сбалансированного и двоичного дерева*, как основных графических структур для разработки алгоритмов работы с данными.

### Сбалансированные и двоичные деревья

**Сбалансированное дерево** – дерево, каждый узел которого имеет одинаковое количество ветвей. Причем, процесс включения новых ветвей в узлы идет сверху вниз, а на каждом уровне дерева – слева направо.





а) сбалансированное дерево

б) несбалансированное дерево

Рис. 4.5

**Двоичное дерево** – особая категория сбалансированных древовидных структур, в которой допускается не более двух ветвей для одного узла. Двоичные деревья, подобно другим сбалансированным деревьям, представляют интерес для Физического, а не для логического представления данных.

#### 4.3.2. Сетевые системы

Если в отношении между данными порожденный элемент имеет более одного исходного элемента, то это отношение уже нельзя описать как древовидную или иерархическую структуру.

В этом случае используется **сетевая структура**.

*Модель называют сетевой, если данные и их связи имеют структуру ГРАФА.*

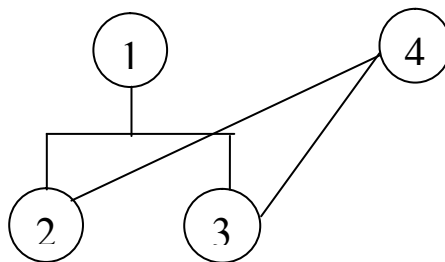


Рис. 4.6

С точки зрения теории графов сетевой модели соответствует произвольный граф (возможно, имеющий циклы и петли).

В узлах графа помещаются типы записей, а ребра интерпретируются как *связи* между типами записей. В сетевых моделях термин «тип записи» уточняется и вводится понятие: «запись-владелец» и «запись-член». Для определения наиболее устойчивых связей вводится понятие набор (Set).

**Set (набор) – это поименованное двухуровневое дерево.**

Каждый тип набора представляет собой отношение между двумя или несколькими типами записей.

Для каждого типа набора один тип записи может быть объявлен его **владельцем** (это исходный тип записи) и один или несколько других типов записей – **членами** набора (**порожденный тип записи**). Каждый экземпляр набора должен содержать один экземпляр записи, имеющий тип записи – владельца и любое количество экземпляров каждого типа записей – членов набора.

Тип набора определяет связь между типами записи-члена и записи-владельца, т.е. это экземпляр поименованной совокупности записей.

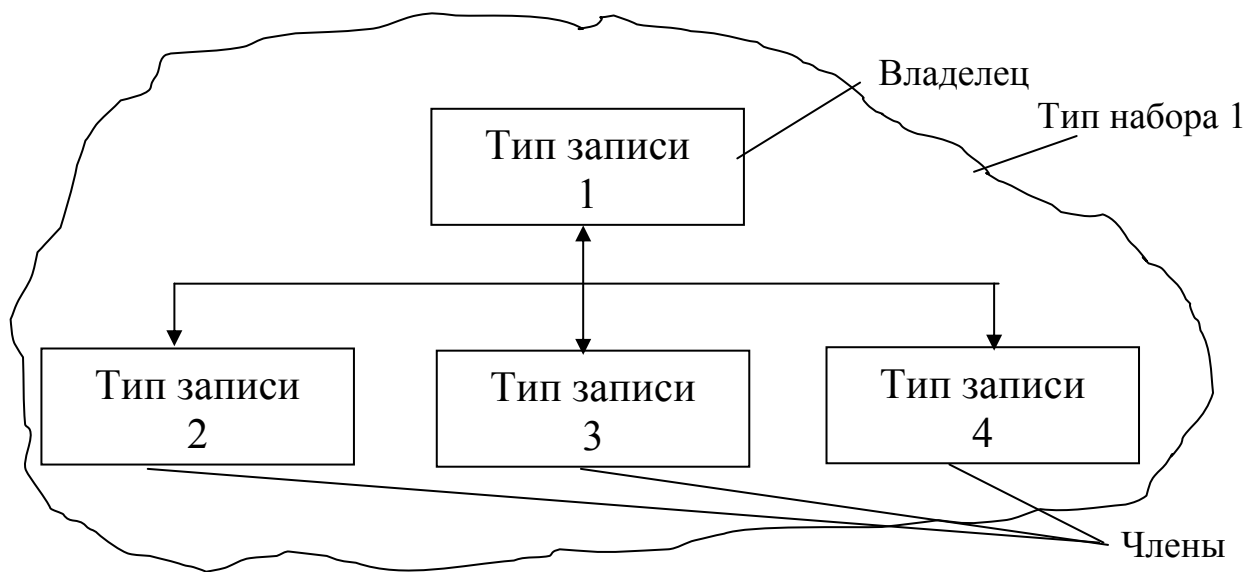


Рис. 4.7

У многоуровневых деревьев тип записи, являющейся записью-владельцем на нижнем уровне дерева, должен быть также объявлен членом набора более высокого уровня.

Рабочая группа по базам данных (РГБД) ассоциации КОДАСИЛ сформулировала ограничения на сетевую модель данных, которые позволяют усвоить структуру сетевых моделей:

- 1) тип набора определяет отображение 1:N между типом записи-владельца и типами записей-членов набора;
- 2) экземпляр типа записи-члена может участвовать только в одном экземпляре данного типа набора;
- 3) тип записи-владельца в типе набора не может совпадать с типом записи-члена.

### 4.3.3. Отличия иерархий от сети

1. В сетевых моделях связи позволяют образовывать более сложную модель, отображая соотношение «многие ко многим», что невозможно в иерархии.
2. Обычно в сети «связи» именованы, а в иерархии безымянны. Т.е. появляется еще один тип «связывающая запись», которая содержит данные, описывающие эту связь (В примере «Поставщик-детали» - это количество деталей).
3. В отличие от иерархической системы в сетевых системах допускаются записи-члены, не участвующие в наборах (связях). Это соответствует порожденным узлам дерева, не имеющим исходные узлы. Таким образом, вместо полной функциональной зависимости допускается частичная функциональная зависимость.
4. В сетевой модели РГБД можно определить несколько типов наборов между двумя типами записей, так что между ними могут быть заданы различные отношения в отличие от единственного возможного отношения исходный – порожденный в иерархических структурах.

#### Приведение сетевых структур к более простому виду

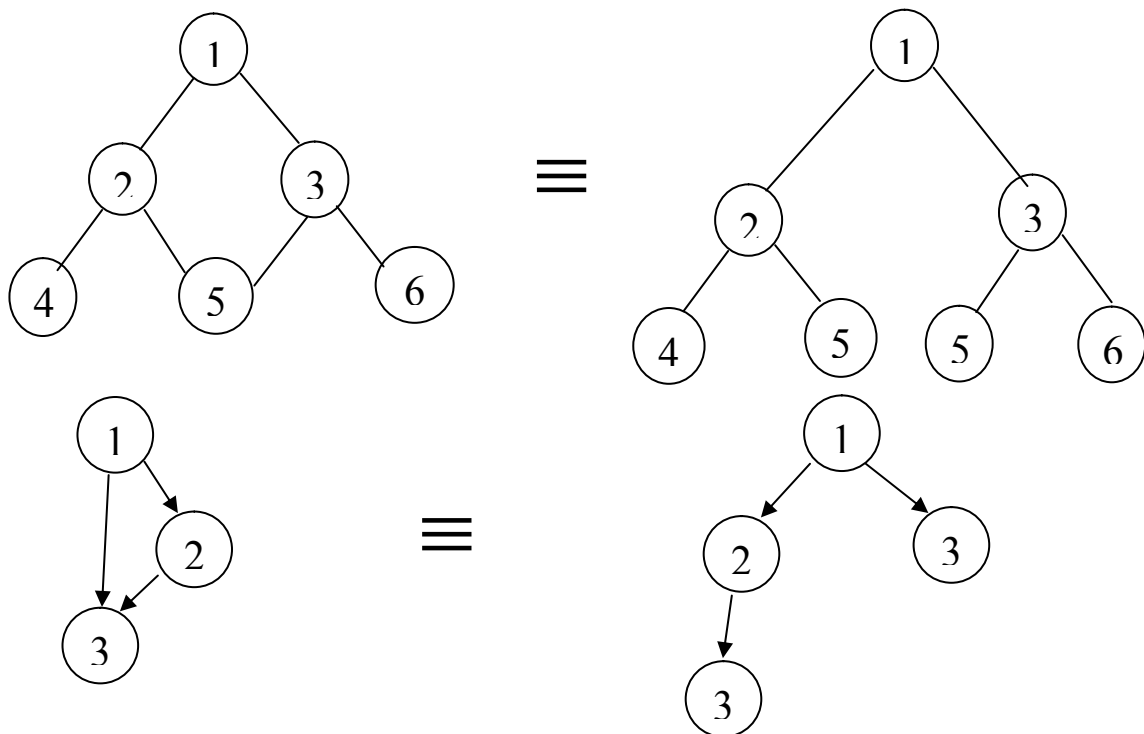


Рис. 4.8

#### 4.3.4. Реляционные модели. Реляционные СУБД

**Реляционная база данных - это такая база данных, которая воспринимается её пользователями как совокупность таблиц.**

Такая форма представления данных в значительной степени обеспечила популярность реляционных моделей, т.к. она привычна для специалиста, пользующегося различной справочной литературой, ведущего экономические расчеты и др.

**Все значения данных в таблицах являются атомарными, т.е. в каждой позиции на пересечении строки и столбца всегда имеется в точности ОДНО значение данных и никогда не бывает множества значений.**

Например:

Таблица 1

База данных "Меню"				
Блюдо	Диеты	Рецепт	Состав	
			Продукты	Масса
Уха из судака	1	Рыбу очистить...	Рыба	100
	2		Морковь	20
	9		Лук	20
	11		Зелень	2
	15		Масло	5

Таблица 1, имеющая многозначное поле "Диеты" и повторяющуюся группу "Состав" не может использоваться в реляционных СУБД.

Для реляционной СУБД эту таблицу необходимо преобразовать в нормализованную таблицу.

Блюдо	Диеты	Рецепт	Продукты	масса
Уха из судака	1	Рыбу ...	Рыба	100
Уха из судака	2	Рыбу ...	морковь	20
Уха из судака	9	Рыбу ...	Лук	20
Уха из судака	11	Рыбу ...	Зелень	2

**Полное информационное содержание базы данных представляется в виде явных значений данных, и такой метод представления данных является единственным.**

В частности, не существует каких-либо специальных "связей" или указателей, соединяющих одну таблицу с другой.

Для этого составляются отдельные таблицы связей.

Всем столбцам таблицы присваиваются однозначные имена и в каждой из них размещаются однородные значения данных (например, **ДИЕТЫ**-числовые, **Продукты**-символьные и т.д.).

Все строки таблицы обязательно отличаются друг от друга хотя бы единственным значением, что позволяет однозначно идентифицировать любую строку такой таблицы.

При выполнении операций с таблицей ее строки и столбцы можно обрабатывать в любом порядке безотносительно к их информационному содержанию.

Почему же БД, удовлетворяющая перечисленным признакам, называется реляционной?

Relation (англ. – «отношение») – математический термин для обозначения таблицы специального вида (точнее «подмножества декартова произведения»).

В 1968 году сотрудник фирмы IBM Э.Р. Кодд (математик по образованию) впервые понял, что для упрощения представления и обработки данных целесообразно использовать аппарат теории множеств.

С помощью языка реляционной алгебры пользователь мог бы получать новые таблицы из уже имеющихся в БД, указывая соединяемые таблицы, выделяемые столбцы и строки и т.п. (работать по принципу «режь и клей»). Это проще и понятнее, чем создание программ для получения новых таблиц из отдельных значений (в лучшем случае строк) существующих таблиц, что приходится делать при использовании любых не реляционных СУБД.

Так как в конце 60-х годов не существовало четкого определения многих общественных терминов, то для определенности рассмотренный тип таблицы был назван Э.Ф.Коддом «отношением» (Relation), строка такой таблицы – «кортежем»; столбец – «атрибутом»; совокупность однотипных значений данных – «доменом». Важно понимать разницу между «доменом» и «атрибутом». «Атрибут» представляет использование домена внутри отношения (примерами являются домен денежных сумм, домен имен, домен целых чисел и т.п.). Несколько атрибутов представляющих свойства объекта, могут получать значения из одного домена.

Таким образом, значения домена по-разному интерпретируются в разных атрибутах, используемых при описании информационной структуры.

Например, атрибуты «зарплата» и «подходный налог» объекта (отношения) «служащие» получают значения из домена денежных сумм.

В реляционной модели предполагается, что все отношения должны быть **нормализованными**. В связи с этим перечислим свойства нормализованных отношений:

- 1) Отношение называется нормализованным, если каждая компонента строки (кортежа) является простым, **атомарным** (неделимым) значением, не состоящим из группы значений. Это не позволяет заме-

нять значение атрибута другим отношением, (что привело бы к сетевому или иерархическому подходу).

- 2) Нормализованное отношение представляется в виде табличной структуры. Имя таблицы соответствует имени отношения, имена столбцов - именам атрибутов, а строки таблицы - кортежам.
- 3) Упорядочение кортежей теоретически несущественно, однако оно может влиять на эффективность доступа к кортежам.
- 4) Все строки (кортежи) отношения должны быть различными.
- 5) Реляционная база данных является совокупностью **изменяющихся** во времени нормализованных отношений различных степеней, которые могут быть связаны друг с другом через общие домены.

Рассмотрим подробнее 5) свойство. Это свойство позволяет выявить основные различия между **математическим отношением и отношением базы данных**. Оно состоит в том, что состояние последнего может меняться со временем при добавлении и/или удалении отдельных кортежей.

Число атрибутов, входящих в отношение, называются **степенью** отношения. А число кортежей отношения - **кардинальным числом** или **мощностью** отношения.

Операции над отношениями выполняются методами реляционного исчисления и реляционной алгебры.

## ГЛАВА 5. БАЗИСНЫЕ СРЕДСТВА МАНИПУЛИРОВАНИЯ РЕЛЯЦИОННЫМИ ДАННЫМИ

Манипуляция данными является частью реляционной модели данных. В работе с данными используются два базовых математических механизма: реляционная алгебра и реляционное исчисление.

### 5.1. Реляционная алгебра

Основные операции реляционной алгебры позволяют создавать новую таблицу, используя в качестве операндов одну (проекция и селекция) или две (объединение, разность и декартово произведение) существующие таблицы. Т.е. операндами реляционной алгебры являются отношения - нормализованные таблицы.

**Проекция, объединение, разность, декартово произведение и селекция** - эти операции являются основными.

Другие, часто используемые операции **пересечения, соединения и деления** можно выразить через выше названные основные операции.

Выберем конкретные отношения (таблицы), которые будем использовать в демонстрационных примерах:

R			S			T		S1		
A	B	C	A	B	C	D	E	C	A	B
a	1	G	b	3	H	1	%	H	b	3
d	7	F	d	7	F	7	&	F	d	7
c	1	H								
h	7	F								

### 1. Объединение.

$R \cup S$  содержит те строки, которые есть либо в R, либо в S, либо в обеих таблицах.

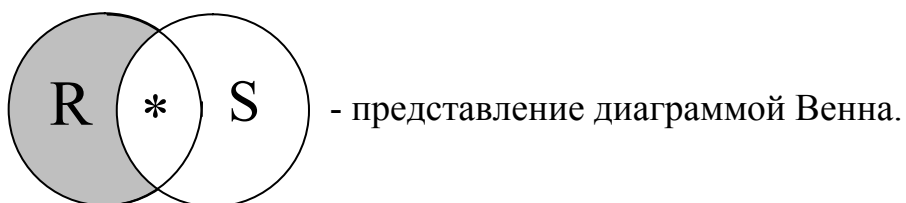
Для того чтобы объединение было возможным, отношения - операнды (R,S) должны быть совместными по объединению, т.е. их атрибуты должны быть определены над совместимыми (однотипными) доменами.

$$R \cup S = \begin{bmatrix} a & 1 & G \\ d & 7 & F \\ c & 1 & H \\ h & 7 & F \end{bmatrix} \cup \begin{bmatrix} b & 3 & H \end{bmatrix} = \begin{bmatrix} a & 1 & G \\ d & 7 & F \\ c & 1 & H \\ h & 7 & F \\ b & 3 & H \end{bmatrix}$$

(можно привести пример с R и S1).

### 2. Разность.

$R - S$  содержит только те строки, которые есть в R, но отсутствуют в S.



$$R - S = \begin{bmatrix} a & 1 & G \\ d & 7 & F \\ c & 1 & H \\ h & 7 & F \end{bmatrix} - \begin{bmatrix} b & 3 & H \\ d & 7 & F \end{bmatrix} = \begin{bmatrix} a & 1 & G \\ c & 1 & H \\ h & 7 & F \end{bmatrix}$$

### 3. Декартово произведение $R \otimes S$

$R \times T$  содержит все возможные строки, составленные сцеплением строки из  $R$  (начало создаваемой строки) со строкой из  $T$ .

Следовательно, число строк произведения равно произведению числа строк, а их длина - сумме длин строк сомножителей.

( $R \times T$  - содержит шесть столбцов и восемь строк).

Степень ( $R \otimes T$ ) = Степень  $R$  + степень  $T$ .

Мощность ( $R \otimes T$ ) = Мощность  $R \times$  мощность  $T$ .

Отсюда следует, что результирующее отношение может иметь очень большие размеры.

$$R \times T = \begin{bmatrix} A & B & C \\ a & 1 & \sigma \\ d & 7 & F \\ c & 1 & H \\ h & 7 & F \end{bmatrix} \times \begin{bmatrix} 1 & \% \\ 7 & \& \end{bmatrix} = \begin{bmatrix} a & 1 & \delta & 1 & \% & 1 \\ a & 1 & \delta & 7 & \& & 2 \\ d & 7 & F & 1 & \% & 3 \\ d & 1 & F & 7 & \& & 4 \\ c & 1 & H & 1 & \% & 5 \\ c & 1 & H & 7 & \& & 6 \\ h & 7 & F & 1 & \% & 7 \\ h & 7 & F & 7 & \& & 8 \\ 1 & 2 & 3 & 4 & 5 & \end{bmatrix}$$

4. **Проекция** - вертикальное подмножество  $\{R[. .]\}$  - выполняется над одним отношением.

$R$  создается из указательных столбцов  $R$  (в заданном порядке) с последующим исключением, если это необходимо, избыточных дубликатов строк.

$R[A,C]$

A	C
a	G
d	F
c	H
h	F

$R[C,B]$

C	B
G	1
F	7
H	1

$R[A]$  - это выборка из каждого кортежа отношения значений атрибутов, входящих в  $A$ , и удаление из полученного отношения повторяющихся строк.

5. **Селекция** (Ограничение или горизонтальное подмножество).

$R$  создается из строк  $R$ , удовлетворяющих заданным условиям, соединяемым логическими операторами. AND, OR, NOT AND и NOT OR.



Каждое условие может состоять из имени какого либо столбца R, оператора сравнения ( $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ,  $>$ ,  $\geq$ ) и константы или имени другого столбца R. При этом сравниваемые столбцы должны быть определены на одном и том же домене.

Пример:

$R[C=F]$

или

$R[B>3 \text{ OR } A=c]$ ,

A	B	C
d	7	F
h	7	F

A	B	C
d	7	F
c	1	H
h	7	F

т.е. можно в общем виде записать:

а)  $R[A\theta v]$

б)  $R[A\theta B]$ ,

где  $v$  - обозначает константу, а B- атрибут отношения R, отличный от A.

Символ  $\theta$  используется для обозначения одной из операций сравнения ( $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ,  $>$ ,  $\geq$ ).

$R[C=P]=\emptyset$  - пустое множество, поскольку в отношении отсутствуют кортежи, где  $C=P$ .

Существует еще несколько чрезвычайно полезных операций, которые хотя и могут быть выражены в терминах основных операций, но имеют специальные названия - пересечение, соединение.

**Пересечение**  $R \cap S$  - содержит только те строки, которые есть и в R, и в S  $R \cap S = R - (R - S)$  - это соответствует области, отмеченной \* на диаграмме Венна для операции разности.

$$R \cap S = \begin{bmatrix} a & 1 & G \\ d & 7 & F \\ c & 1 & H \\ h & 7 & F \end{bmatrix} \cap \begin{bmatrix} b & 3 & H \\ d & 7 & F \end{bmatrix} = \begin{matrix} A & B & C \\ |d & 7 & F| \end{matrix}$$

**Соединение** R с T по столбцам i и j ( $R[i\theta j]T$ ) содержит те строки декартова произведения  $R \times T$ , в которых значение из i-го столбца R находится в отношении  $\theta$  ( $<$ ,  $\leq$ ,  $=$ ,  $\neq$ ,  $>$ ,  $\geq$ ) со значением из j-го столбца T.

При этом сравниваемые столбцы должны быть определены на одном и том же домене.

Таким образом, операция соединения имеет сходство с декартовым произведением. Однако здесь добавлено **условие**, согласно которому вместо полного произведения всех строк в результирующее отношение включаются только строки, удовлетворяющие определенному соотношению между атрибутами соединения (A,B) соответствующих отношений.  $R[A\theta B]S$ .

Имеется несколько вариантов соединения:

а) **Тетта - соединение**. Так называют соединения, полученные при любом операторе  $\theta$ , кроме оператора « $=$ ».

Например,  
 $R[B \leq D]T =$

$$\left[ \begin{array}{ccc} \underline{A} & \underline{B} & \underline{C} \\ a & 1 & \sigma \\ d & 7 & F \\ c & 1 & H \\ h & 7 & F \end{array} \right] \begin{array}{cc} D & E \\ \left| \begin{array}{cc} 1 & \% \\ 7 & \& \end{array} \right| \end{array} = \begin{array}{ccccc} A & B & C & D & E \\ \left| \begin{array}{ccccc} a & 1 & \sigma & 1 & \% \\ a & 1 & \sigma & 7 & \& \\ d & 7 & F & 7 & \& \\ c & 1 & H & 1 & \% \\ c & 1 & H & 7 & \& \\ h & 7 & F & 7 & \& \end{array} \right| \end{array}.$$

б) **Эквисоединение** - это соединение, полученное при  $\theta$ , являющимся оператором « $=$ ».

Например,  $R[B = D]T$ .

A	B	C	D	E
a	1	$\delta$	1	%
d	7	F	7	&
c	1	H	1	%
h	7	F	7	&

в) **Естественное соединение** - это эквисоединение, в котором исключен столбец  $j$ . (В нашем случае столбцу  $j$  соответствует атрибут "Д").

$R[B = D]T$ .

A	B	C	E
a	1	$\sigma$	%
d	7	F	&
c	1	H	%
h	7	F	&

г) **Композиция** - это эквисоединение, в котором исключены столбцы  $i$  и  $j$ , по которым проводилось соединение. Следует отметить, что  $i$  и  $j$  могут рассматриваться не только как определенные столбцы, но и как совместимые множества столбцов.

Т.е. это соединение отличается от естественного тем, что из результирующего отношения удаляются оба атрибута соединения. Поэтому степень результирующего отношения на две единицы меньше суммы степеней операндов.

## 5.2. Реляционное исчисление

Если реляционная алгебра представляет собой совокупность операций высокого уровня над отношениями, то **реляционное исчисление** - простой набор *правил* для записи выражения, определяющего некоторое новое отношение в терминах заданной совокупности отношений.

Другими словами, *реляционное исчисление есть метод определения того отношения, которое нам желательно получить (как ответ на запрос) в терминах уже имеющихся отношений (отношений в базе данных).*

Возьмем пример данных в реляционной форме:

S

S#	Симя	Область	Город
S1	Иванов		С-Петербург
S2	Петров		Томск
S3	Сидоров		Москва

SP

S#	P#	Кол-во
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S3	P2	200

P

P#	Римя	Цвет	Вес	Город	Кол-во
P1	Втулка	Красный	12	Ленинград	
P2	Болт	Зеленый	17	Москва	
P3	Винт	Голубой	17	Томск	
P4	Винт	Красный	14	Ленинград	

Рассмотрим запрос: «Для каждой поставляемой детали найти P# (номер детали) и названия всех городов, из которых она поставляется».

Определение в реляционном исчислении результата ответа на запрос имеет вид:  $\{(SP, P\#, S, ГОРОД) : SP, S\# = S, S\#\}$

где: «{ }» - указывает, что данное выражение является определением некоторого множества (отношения);

«:» - означает «такое, что» или «где»;

терм,: предшествующий двоеточию, представляет типичный член (кортеж) множества;

а: терм, следующий за двоеточием, является условием или (*предикатом*), описывающим «определяющее» свойство множества.

Значением полного выражения является множество всех пар (P#, ГОРОД), таких, что значение P# берется из кортежа отношения SP, а значение ГОРОД, берется из кортежа отношения S, где значения S# в этих двух кортежах равны!

Причем: (P#, ГОРОД) - данное определение множества полностью не процедурно. Оно просто устанавливает, каким будет результат ответа на запрос, но не указывает, как его получить.

В алгебраическом решении той же задачи, напротив, указывается, каким образом можно построить результат: в данном случае соответствующее решение содержит последовательность двух операций: соединение и проектирование (проекцию).

## ГЛАВА 6. ПРОЕКТИРОВАНИЕ БАЗ ДАННЫХ

### 6.1. Функциональные зависимости

В реляционных базах данных схема содержит как структурную, так и семантическую (смысловую) информацию.

**Структурная информация** связана с объявлением отношений.

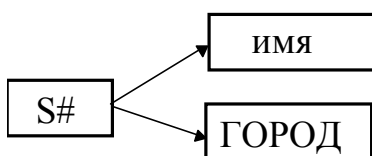
**Семантическая информация** - выражается множеством известных функциональных зависимостей между атрибутами отношений, объявленных в схеме.

Что же такое **функциональная зависимость**? (Ведь мы говорим, что в реляционной БД все связи функциональны).

*Если задано отношение R, то мы говорим, что атрибут Y отношения R функционально зависит от атрибута X отношения R тогда и только тогда, когда каждое значение X в отношении R в каждый момент времени связано точно с одним значением Y (т.е. если известно X, то известно Y).*

Однако одно и то же значение X может появиться в нескольких различных кортежах отношения R. Если Y функционально зависит от X, то по определению каждый из этих кортежей должен содержать также одно и то же значение Y.

Например, в отношении S нашей БД каждый из атрибутов Симя, Город, функционально зависит от атрибута S#.



Т.е. для определенного значения S# существует в точности одно соответствующее значение каждого из атрибутов Симя и Город.

Рис. 6.1. Обозначение функциональных связей

Однако некоторые функциональные зависимости могут быть нежелательными из-за побочных эффектов или аномалий, которые они вызывают при модификации данных.

В связи с этим возникает вопрос о корректности представленной схемы. **Корректной** считается схема, в которой отсутствуют нежелательные функциональные зависимости.

В противном случае приходится прибегать к процедуре, называемой **декомпозицией** (разложением), при которой данное множество отношений заменяется другим множеством отношений (число их возрастает), являющихся проекциями первых.

**Цель этой процедуры** - устранить нежелательные функциональные зависимости, а, следовательно, и аномалии, что составляет суть процесса **нормализации**.

**Нормализация** - это пошаговый обратимый процесс замены данной схемы (или совокупности отношений) другой схемой, в которой отношения имеют более простую и регулярную структуру.

*В теории нормальных форм определяются различные нормальные формы, которые ограничивают типы допустимых функциональных зависимостей отношения.*

Для приведения отношения к какой-либо нормальной форме прибегают к декомпозиции. При этом декомпозиция должна сохранять **эквивалентность** схем при замене одной схемы на другую (т.е. возможность восстановления исходной схемы).

Введем необходимые понятия:

Идентификатор – свойство объекта, однозначно определяющее данный объект, рассматривается как **ключ** записи.

Атрибут, входящий в ключ, называется **первичным**; в противном случае он называется **не первичным**.

Функциональная зависимость  $A \rightarrow B$  называется **полной функциональной зависимостью**, если  $B$  зависит от всей группы атрибутов  $A$ , а не от ее части (подмножества) т.е. от всего множества  $A$ , но не зависит ни от какого подмножества  $A$ .

Например, если  $A1 = \{A_1, A_2, \dots, A_N\}$  и  $A_1, A_2 \rightarrow B$ , то функциональная зависимость  $B$  от  $A$  неполная.

Например, в отношении  $S$ , атрибут ГОРОД функционально зависит от составного атрибута ( $S\#, SI\text{мя}$ ). Однако он не находится в полной функциональной зависимости, т.к. также функционально зависит от одного  $S\#$  (рис. 6.1).

## 6.2. Теория нормальных форм

При рассмотрении реляционных моделей данных, мы выяснили, что отношения в реляционной базе данных всегда нормализованы в том смысле, что содержит только «скалярные» значения. Однако некоторые отношения все еще могут обладать нежелательными свойствами. Процесс дальнейшей нормализации далее называется просто *нормализацией*, основывается на концепции нормальных форм. Нормализация выполняет задачи:

- ограничивает определенный тип функциональной зависимости;
- устраняет аномалии при выполнении операций над отношениями БД.

В зависимости от ограничений, накладываемых на типы связей между данными, различают следующие нормальные формы (для обозначения нормальных форм используют обозначения: 1НФ, 2НФ, 3НФ, НФБК, 4НФ, 5НФ).

1НФ, 2НФ и 3НФ - нормальные формы ограничивают зависимость не первичных атрибутов от ключей.

Нормальная форма Бойса-Кодда (НФБК) - ограничивает также зависимость первичных атрибутов.

4НФ - формулирует ограничения на виды многозначных зависимостей.

5НФ - вводит другие типы зависимостей, называемых зависимыми соединениями.

### 6.2.1. Первая нормальная форма (1НФ)

Отношение находится в 1НФ тогда и только тогда, когда все входящие в него домены содержат только атомарные значения.

Это определение просто устанавливает, что любое нормализованное отношение находится в 1НФ.

Ненормализованному отношению соответствует многоуровневая таблица (иерархия) в отличие от однородной табличной структуры нормализованного отношения.

Пример.

Рейсы (номер, пункт отправления, пункт назначения, расписание).

Расписание (день, время вылета).

ТУ 154 Томск-Москва понедельник 9.40

вторник 9.40

пятница 10.30

ТУ 154 Москва-Томск понедельник 7.30

вторник 7.30

пятница 7.30

Для преобразования этого ненормализованного отношения в 1НФ необходимо в составном отношении РЕЙСЫ заменить отношение РАСПИСАНИЕ соответствующими атрибутами:

Рейс (номер, пункт отправления, пункт назначения, день, время вылета).

ТУ 154 Томск-Москва понедельник 9.40

ТУ 154 Томск-Москва вторник 9.40

ТУ 154 Томск-Москва пятница 10.30

ТУ 154 Москва-Томск понедельник 7.30.

### 6.2.2. Вторая нормальная форма (2НФ)

Пусть имеется отношение ПОСТАВКИ, содержащее данные о поставщиках, идентифицируемые номером П#, поставляемых ими товарах и ценах. ПОСТАВКИ (П#, ТОВАР, ЦЕНА).

Предположим, что поставщик может поставлять различные товары, а один и тот же товар могут поставлять разные поставщики.

Таким образом, **ключ** отношения будет состоять из атрибутов ( П#, ТОВАР). Известно, что цена любого товара зафиксирована, т.е. все поставщики поставляют товар по одной и той же цене.

Семантика отношения включает следующие зависимости:

П#, ТОВАР→ЦЕНА (по определению ключа) ТОВАР→ЦЕНА.

Можно отметить *неполную* функциональную зависимость атрибута ЦЕНА от ключа (присутствует частичная зависимость).

Кроме того что цена зависит от группы атрибутов (П# и ТОВАР), она еще может зависеть только от атрибута ТОВАР.

*Это приводит к следующим аномалиям:*

**1. Аномалия включения:** Если у поставщика появляется новый товар, информация о товаре и его цене не сможет храниться в БД до тех пор, пока поставщик не начнет поставлять его.

**2. Аномалия удаления:** Если поставки некоторого товара прекращаются, из базы данных придется удалить сведения о товаре и его цене, даже если он имеется в наличии у поставщиков.

**3. Аномалия обновления:** При изменении цены товара, необходим полный просмотр отношения, с целью найти все поставки товара и изменение цены, было отражено для всех поставщиков.

Таким образом, изменение значения атрибута одного объекта влечет необходимость изменений в нескольких кортежах отношения: в противном случае база данных окажется несогласованной.

*Причиной этих аномалий является неполная функциональная зависимость атрибута ЦЕНА от ключа, что обусловлено объединением в отношении ПОСТАВКИ двух семантических факторов в одной структуре.*

Разложение отношения ПОСТАВКИ на два отношения устраняют неполную функциональную зависимость.

Итак, отношение находится во второй нормальной форме (2НФ), если оно находится в 1НФ, и каждый не первичный атрибут функционально полно зависит от ключа, а первичный тот, который входит в ключ.

Следующее разложение приводит к отношению во 2НФ:

ПОСТАВКИ (П#, ТОВАР)

ЦЕНА\_ТОВАР (ТОВАР, ЦЕНА).

Цену товара конкретной поставки можно определить путем соединения двух отношений по атрибуту ТОВАР. Изменение цены товара вызовет модификацию лишь одного кортежа второго отношения.

Отношение, которое находится в 1НФ и не находится в 2НФ, всегда можно преобразовать в эквивалентную совокупность отношений, находящуюся во 2НФ.

Заметим, что если отношение находящееся в 1НФ не находится во 2НФ, то оно как правило имеет *составной* первичный ключ.

### 6.2.3. Третья нормальная форма (3НФ)

Рассмотрим транзитивную (т.е. опосредованную) зависимость следующего типа:

Если  $A \rightarrow B$ ,  $B \rightarrow A$  (т.е.  $B$  не является ключом) и  $B \rightarrow C$ , то  $A \rightarrow C$ .

Пусть имеется отношение ХРАНЕНИЕ (ФИРМА, СКЛАД, ОБЪЕМ), которое содержит информацию о фирмах, получающих товары со складов, и объемах этих складов.

В отношении имеются следующие зависимости:

ФИРМА  $\rightarrow$  СКЛАД (фирма получает товары только с одного склада).

СКЛАД  $\rightarrow$  ОБЪЕМ

*Аномалии:*

1. **Включения:** если на данный момент отсутствует фирма, получающая товар со склада, то в базу данных нельзя ввести информацию об объеме склада.

2. **Обновления:** если объем склада изменяется, необходимы просмотр всего отношения и изменения картежей для фирм, связанных со складом.

3. **Удаления:** если последняя фирма перестает получать товар со склада, данные о складе и его объеме нельзя сохранить в базе данных.

Преобразование отношения в 3НФ устраняет рассмотренные аномалии.

Отношение находится в 3НФ, если оно находится во 2НФ и в нем отсутствуют транзитивные зависимости не первичных атрибутов от ключа.

Следующее разложение приводит к отношениям в 3НФ:

ХРАНЕНИЕ (ФИРМА, склад)

С\_ОБЪЕМ (СКЛАД, объем).



Итак, уровень нормализации данного отношения определяется *семантикой*, а не значениями данных, которые появились в отношении в некоторый конкретный момент времени.

Невозможно по содержанию таблицы данного отношения в данный момент сказать, в какой нормальной форме находится это отношение.

Перед тем, как сделать такое заключение, необходимо знать содержание данных, т.е. имеющиеся зависимости между ними. Все, что необходимо сообщить СУБД об этих зависимостях, заключается только в указании атрибута/(ов), составляющего первичный ключ.

Введем новое понятие - **детермината** - это атрибут (возможно составной) от которого какой-либо другой атрибут зависит функционально (полно).

Тогда можно дать строгое определение 3НФ:

**нормализованное отношение R находится в третьей нормальной форме (3НФ), если каждая детермината является *возможным* (т.е. первичным) ключом.**

Это определение отходит от понятия 1 и 2 НФ и не связано с термином транзитивной связи, оно было введено Бойсом и Коддом, поэтому отношение, удовлетворяющее этому определению, называют отношением в нормальной форме *Бойса/Кодда* (*БКНФ*).

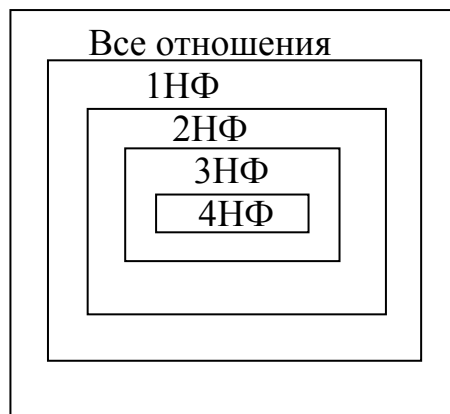


Рис. 6.2

#### 6.2.4. Четвертая нормальная форма (4НФ)

Отношение находится в 4НФ, если оно находится в НФБК, но в нем отсутствуют многозначные зависимости, которые не являются функциональными.

Что же такое многозначная зависимость?

Говорят, что A многозначно определяет B в R (или что B многозначно зависит от A), обозначая указанную зависимость  $A \twoheadrightarrow B$ , если каждому значению A соответствует множество (возможно и пустое) значений B, никак не связанных с другими атрибутами R.

Т.е. отношение R кроме атрибутов A и B содержит другие атрибуты. Например, отношение «ПРОФЕССОР»

ИД#	СТУДЕНТ	КУРСЫ	ДОЛЖНОСТЬ
588	Семен	10(физ)	Доцент
588	Иван	12(мат)	Доцент
588	Семен	12	Доцент
588	Иван	10	Доцент
589	Мария	10	Ассистент

Если обновляется многозначная зависимость атрибутов СТУДЕНТЫ или КУРСЫ от атрибута ИД#, т.е. каждому значению атрибута ИД# должно соответствовать фиксированное множество значений атрибутов СТУДЕНТЫ или КУРСЫ.

Другими словами, возможно изменение значения этих атрибутов в любой строке отношения. Замена значения КУРСЫ в кортеже <<588 Семен 12 ДОЦЕНТ>, даст <<588 Семен 10 ДОЦЕНТ >> - таким образом, другие значения кортежа никак не связаны со значениями многозначных атрибутов.

То есть полученный кортеж имеется уже в отношении.

По другому определению 4НФ требуется, чтобы в отношении для любой нетривиальной многозначной зависимости  $X \twoheadrightarrow Y$ , X обязательно содержал ключ отношения.

Следующие отношения находятся в 4НФ

R1 (ИД#, СТУДЕНТЫ)

R2 (ИД#, КУРСЫ)

R3 (ИД#, ДОЛЖНОСТЬ)

4НФ показывает, что отношение может находиться в НФБК и, тем не менее, могут существовать некоторые аномалии, особенно при обновлениях.

Например, если у профессора появляется еще один студент, в отношение необходимо добавить не один кортеж, а столько, сколько профессор читает курсов.

Итак, нормальное отношение R находится в 4НФ тогда и только тогда, когда при существовании многозначной зависимости R, скажем атрибута B от атрибута A, все атрибуты отношения R также функционально зависят от A.

### 6.3. Языковые средства СУБД

#### 6.3.1. Языки описания данных

Как программисты, так и АБД должны иметь возможность точно описывать и определять структуры данных.

Для этой цели существуют различные языки описания данных. **Язык описания данных (DDL)** - является средством объявления системе управления базами данных структур, которые будут использоваться.

#### *Функции языка описания данных (ЯОД)*

ЯОД, используемый для логического описания данных (т.е. это концептуальный уровень), должен выполнять следующие функции:

1. Идентифицировать типы подразделений данных, такие как элемент данных, сегмент, запись, файл базы данных (типы подразделений данных различны в разных языках).

2. Присваивать уникальное имя каждому типу элемента данных, записи, файлу, базе данных и другим подразделениям данных.

3. Специфицировать для таких типов, как аргумент данных, запись и другие подразделения данных, какие типы элементов данных входят в них, указывать порядок элементов и повторяющиеся группы.

4. Специфицировать, какие типы элементов данных, их частей или комбинаций типов используются в качестве ключей.

5. Специфицировать, каким образом можно установить отношения между типами сегментов (поименованный квант данных, состоящий из одного или несколько полей) или записей для создания структур, подобных рассмотренным выше.

6. ЯОД также можно специфицировать. Тип кодирования, который будет использоваться программистами для представления данных (двоичный, символьный и т.п.). Это не следует путать с кодированием, применяемым для физического представления данных.

7. Длину элементов данных.

8. Диапазон значений, допустимых для элемента данных.

9. Количество элементов данных, т.е. размер и число измерений массива.

10. Порядок записей в файле или порядок записей в базе данных.

11. Замки секретности для предотвращения несанкционированного доступа (чтения или модификации) данных.

В логическом описании данных не должны специфицироваться методы адресации, индексирования, поиска или способы размещения данных на внешних запоминающихся устройствах, поскольку это относится к вопросам физической, а не логической организации данных.

Логические описания могут содержать указания о том, какие данные будут использоваться или как будет организован их поиск. Эти указания позволяют оптимальным образом выбирать методы физической организации данных, но такие указания не должны ограничиваться только логическим уровнем описания данных.

### Три типа описания данных

В самом начале мы видели, что существует несколько уровней представления данных

- представление прикладного программиста о данных выражается под-схемой и часто отличается от представления, заданного общей глобальной схемой (т.е. концептуального уровня).

Последнее представление в свою очередь отличается от физического размещения данных. Следовательно, должны существовать ТРИ описания данных, отражающие:

1. Представление прикладного программиста - описание подсхемы.
2. Общее логическое представление данных - описание схемы.
3. Организацию физического хранения - описание физических записей и связей между ними.

Таким образом СУБД использует описания данных для получения логических записей из физических, а уже из логических записей формируются записи, запрашиваемые конкретными прикладными программами.

а) **Язык описания данных на уровне прикладных программ - это язык подсхем.**

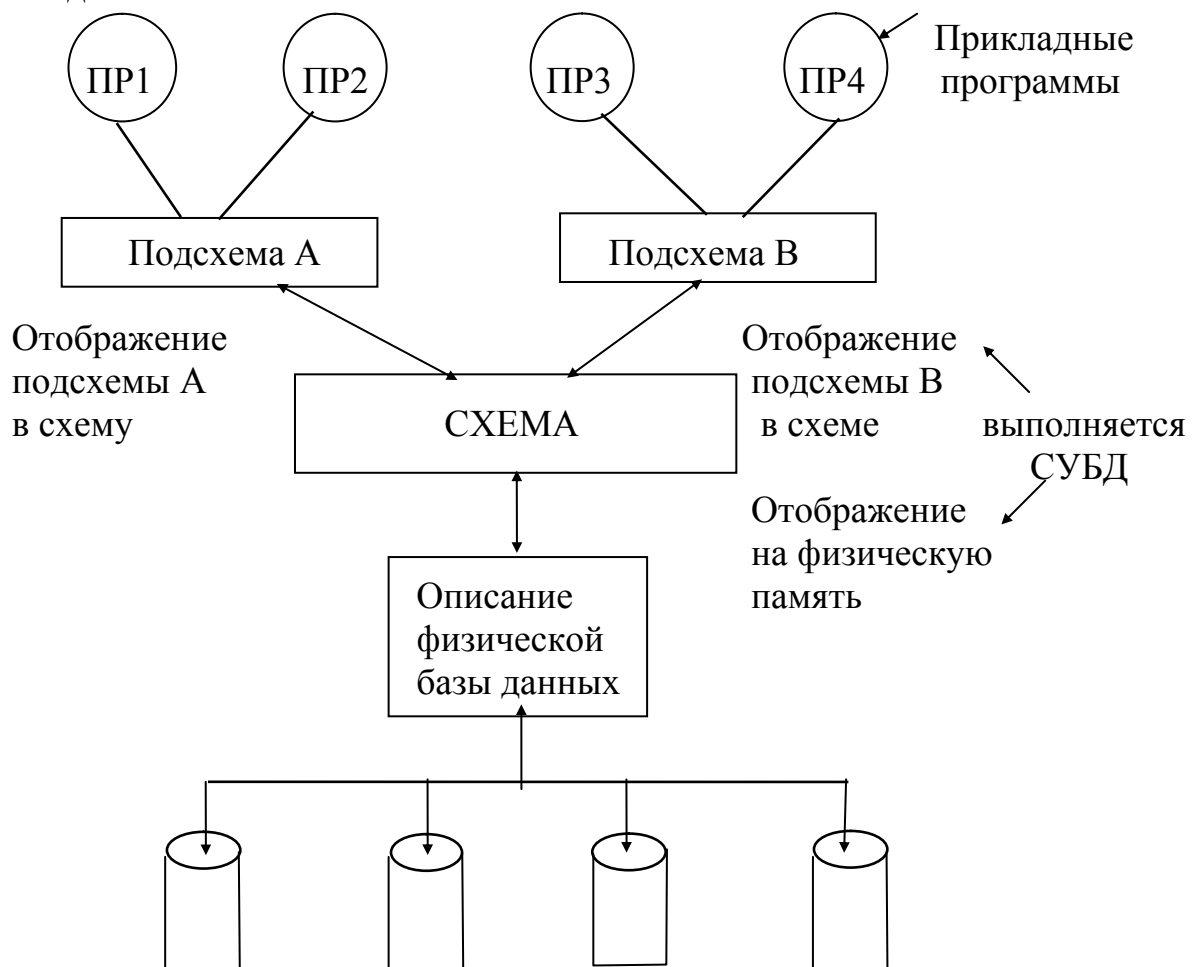


Рис. 6.3

В настоящее время во многих СУБД язык, используемый программистами для описания данных, отличен от языка, используемого администратором данных, и представляет собой язык высокого уровня.

#### **б) Языки схемы.**

В большинстве СУБД для определения схем используются свои собственные языки описания данных.

Обычно они отличны от языков программирования, которые не имеют средств для определения всего множества отношений, допустимых в схемах.

Как пример можно назвать язык DL/1 (Data Language 1) - язык, который используется в системе IMS фирмы IBM, или язык запросов.

Мы учили три типа схем: реляционный подход, сетевой, иерархический.

Прежде чем переходить к изучению языков запросов рассмотрим некоторые характерные для каждого из подходов операции.

#### **1) Реляционный подход.**

При рассмотрении реляционных операторов отметим, что единообразие представления данных (в виде отношений) ведет к соответствующему единообразию в наборе операторов: так как информация представлена одним и только одним способом, то нам необходим только один оператор для каждой из основных функций:

- включить (INSERT)
- удалить (DELETE)
- выбрать (GET)
- обновить (UPDATE) и т.д.

Этого единообразия нет в более сложных ситуациях с более сложными структурами, когда информация может быть представлена несколькими способами и, следовательно, требуется несколько наборов операторов.

#### **Рассмотрим некоторые характерные операторы.**

1) Основным оператором, необходимым для выборки, является оператор GET NEXT WHERE (получить следующий, где), который будет выбирать следующую строку таблицы, удовлетворяющую определенному условию.

«Следующий (NEXT)» - понимается по отношению к текущей позиции (обычно это последняя выбранная строка; при первом выполнении этой операции предполагается, что текущая позиция предшествует первой строке таблицы).

Этот оператор проиллюстрируем на примере, где в общих чертах покажем алгоритмы, требуемые для выполнения двух конкретных запросов, к известной базе данных «поставщики - детали».

Эти два запроса сформулированы симметрично

S (S#, СИМЯ, СГОРОД);

SP (S#, P#, Кол-во);

P (P#, РИМЯ, РВЕС, ГОРОД).

Запрос 1. Найти N поставщиков, поставляющих деталь P2	Запрос 2. Найти номера деталей поставляемых поставщиком S2
NEXT: Получить следующую поставку, где P# = P2. Поставка найдена? Если нет, то выход из программы Печать S# Переход на NEXT	NEXT: Получить следующую поставку, где S# = S2. Поставка найдена? Если нет, то выход из программы Печать P# Переход на NEXT

SP

S#	P#	K
S1	P1	300
S1	P2	200
S1	P3	400
S2	P1	300
S2	P2	400
S3	P3	200

2) Включить - (INSERT).

**Задание:** рабочее пространство W содержит информацию о новом поставщике S4.

Включить эту информацию в базу данных.

**Решение:** включить кортеж из W в отношение «Поставщики» (S=).

3) Удалить - (DELETE).

Удалить поставку, связывающую деталь P2 и поставщика S3.

Удалить кортеж поставки, в котором P# = P2 и S# = S3.

4) Обновить - (UPDATE).

Поставщик S1 переместился из Ленинграда в Сочи. Обновить кортеж поставщика, в котором S# = S1, изменив значение ГОРОД на "СОЧИ".

Все эти примеры мы проиллюстрировали на примере одной записи. Однако многие задачи формулируются более естественным способом не в терминах отдельных записей, а в терминах множеств.

При решении таких задач (которые работают с множествами) важными операциями являются **операции выборки**.

**Задачи.**

1. Найти ГОРОД для поставщика S2

S

S#	СИМЯ	Область	ГОРОД
S1	Иванов	Ленинградская	Ленинград
S2	Петров	Московская	Москва
-	-	-	-

Результат

→	ГОРОД
	МОСКВА

Ответом является таблица (отношение) с одной строкой и одним столбцом. Этот столбец основан на домене размещений (городов) и содержит единственное значение МОСКВА.

- 2) Найти S# (номер поставщика) и СИМЯ (ИМЯ), для поставщиков, находящихся в городе «Москве».

S				Результат	
S#	СИМЯ	Область	ГОРОД	S#	СИМЯ
S2	Петров	Московская	Москва	S2	Петров
S3	Сидоров	Московская	Москва	S3	Сидоров

Результатом опять является таблица на этот раз с 2-мя строками и 2-мя столбцами.

- 3) Найти РИМЯ (наименование деталей) для деталей, поставляемых поставщиком S1.

SP		P		Результат	
# S	P#	P#	РИМЯ	РИМЯ	
S1	P1	P1	Втулка	Втулка	
S1	P2	P2	Болт	Болт	
S1	P3	P3	Винт	Винт	

Опять результатом является таблица. Т.е. фактически результат любой операции выборки может рассматриваться как таблица. (Важно!).

В этом частном примере таблица результата является подмножеством одной таблицы, как и в предыдущем примере.

Однако при конструировании этого результата должны быть просмотрены две таблицы.

- 4) Для каждой поставляемой детали определить P# (номер детали) и названия всех городов, из которых она поставляется.

SP		S		ГОРОД		Результат	
S#	P#	S#		P#	ГОРОД	P#	ГОРОД
S1	P1	S1		P1	Ленинград	P1	Ленинград
S1	P2	S2		P2	Москва	P2	Ленинград
S1	P3	S3		P3	Москва	P3	Ленинград
S2	P1			P1		P1	Москва
S2	P2			P2		P2	Москва
S3	P2						

В этом примере не только необходимо опять просмотреть две таблицы, но значения результата действительно извлекаются из двух таблиц. Причем,

избыточные дублирующие строки исключаются из окончательного результата (следует из определения нормализованного отношения).

Итак, любое количество таблиц может быть использовано при формировании результата, как в выборе по условию, так и в фактическом получении значений результата. Другими словами процесс выборки есть процесс построения таблиц.

**Определим набор операндов, необходимых, для построения таблиц при выборке:**

- **SELECT** - выбрать;
- **PROJECT** - спроектировать;
- **JOIN** - соединить.

Оператор **SELECT** - конструирует новую таблицу посредством взятия **горизонтального подмножества существующей таблицы**, т.е. всех строк существующей таблицы, которые удовлетворяют какому-то условию.

Оператор **PROJECT**, наоборот, формирует **вертикальное подмножество существующей таблицы** посредством извлечения выбранных столбцов и удаления любых избыточных дублируемых строк в извлеченном множестве столбцов.

Используя эти два оператора, можно сразу написать программы для первых двух предыдущих примеров.

1) Найти ГОРОД для поставщика S1.

**Шаг 1.** **SELECT S WHERE S#='S1' GIVING TEMP** - этот шаг дает нам следующую таблицу:

TEMP			
S#	СИМЯ	Область	ГОРОД
S1	Иванов	Ленинградская	Ленинград

**Шаг 2.** **PROJECT OVER ГОРОД GIVING RESULT** - этот шаг выделяет столбец ГОРОД из таблицы TEMP, обеспечивая желаемый результат.

2) Найти S# и СИМЯ для поставщика в Москве.

**SELECT S WHERE ГОРОД='МОСКВА' GIVING TEMP.**

**PROJECT TEMP OVER S#, СИМЯ GIVING RESULT.**

Оставшиеся два примера (они используют две таблицы) требуют применения оператора **JOIN** (соединить).

Если две таблицы имеют общий домен, то они могут быть соединены по этому домену.

Результатом соединения является новая более широкая таблица.



S#	СИМЯ	ОБЛАСТЬ	СГОРОД	P#	РИМЯ	ВЕС	РГОРОД
S1	Иванов	Ленинградская	Ленинград	P1	Втулка	12	Ленинград
S1	Иванов	Ленинградская	Ленинград	P4	Винт	14	Ленинград
S2	Петров	Московская	Москва	P2	Болт	17	Москва
S3	Сидоров	Московская	Москва	P2	Болт	17	Москва

Из исходных таблиц берутся те строки, которые содержат одно и тоже значение из общего домена.

Таблицы S и P могут быть соединены по их совпадающим значениям домена «размещение» СГОРОД и РГОРОД. Мы переименовали два столбца СГОРОД и РГОРОД - чтобы избежать неоднозначности.

Заметим, если какая либо строка (ТОМСК) не имеет копии соответствующего значения в другой таблице, она просто не участвует в результате. В примерах продемонстрирована работа с подязыком данных высокого уровня, оперирующего множествами.

## 6.4. Язык запросов SQL

Язык запросов SQL (Structure Query Language) принят в качестве **языка манипулирования данными** (Data Management Language — DML). Основные операторы DML — это:

**SELECT** - реализует операции отображения(выборки) ;

**INSERT, UPDATE и DELETE**- реализуют операции запоминания: (обновления, удаления, включения).

Язык SQL имеет блочную структуру, которую образует выражение SELECT FROM WHERE, причем два последних слова необязательны.

Назовем правила формирования запросов:

1. Запрос может представляться как одноблочной программой, так и несколькими вложенными блоками.
2. Условия внутри блока соответствуют связям внутри отношения и между отношениями.
3. В зависимости от своего типа вложенность может подразумевать операцию соединения или деления.

*Конструкция операции выборки имеет следующий вид:*

- **SELECT** *список атрибутов и/или агрегатных функций от атрибутов(sum, max, min и др.)*
- **FROM** *имя/имена отношений*
- **WHERE** *условия (условия могут быть булевыми предикатами).*

**Введем в рассмотрение примерную схему отношений:**

**S**

<i>S#</i>	<i>SNAME</i>	<i>STATU</i>	<i>CITY</i>
<i>S1</i>	<i>Smith</i>	20	<i>London</i>
<i>S2</i>	Jones	10	Paris
<i>S3</i>	Black	30	Paris
<i>S4</i>	Clark	20	Lon-
<i>S5</i>	Adams	30	don Athens

**J**

<i>J#</i>	<i>JNAM</i>	<i>CITY</i>
<i>J1</i>	<i>Sorter</i>	<i>Paris</i>
<i>J2</i>	Dis-	Rome
<i>J3</i>	play	Athens
<i>J4</i>	OCR	Athens
<i>J5</i>	Con-	London
<i>J6</i>	sole	Oslo
<i>J7</i>	RAID	London
	EDS	
	Tape	

**P**

<i>P#</i>	<i>PNAM</i>	<i>COLOR</i>	<i>WEIGH</i>	<i>CITY</i>
<i>P1</i>	<i>Nut</i>	<i>Red</i>	12	London
<i>P2</i>	Bolt	Green	17	Paris
<i>P3</i>	Screw	Blue	17	Rome
<i>P4</i>	Screw	Red	14	London
<i>P5</i>	Cam	Blue	12	Paris
<i>P6</i>	Cog	Red	19	London

**SPJ**

<i>S#</i>	<i>P#</i>	<i>J#</i>	<i>QTY</i>
<i>S1</i>	<i>P1</i>	<i>J1</i>	200
S1	P1	J4	700
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
S3	P4	J2	500
S4	P6	J3	300
S4	P6	J7	300
S5	P2	J2	200
S5	P2	J4	100
S5	P5	J5	500
S5	P5	J7	100
S5	P6	J2	200
S5	P1	J4	100
S5	P3	J4	200
S5	P4	J4	800
S5	P5	J4	400
S5	P6	J4	500

Рассмотрим вначале операции выборки ( конструкция SELECT), а далее — операции обновления (INSERT, UPDATE, DELETE.).

Для простоты будем предполагать, что все операторы вводятся интерактивно (с экрана).

### Операции выборки в SQL

Операции выборки в SQL - это по существу **табличное выражение**, которое может быть сколь угодно сложным. Более полная и более формальная трактовка табличных выражений приведена ниже в этом разделе.

6.4.1. Получить значение цвета и название города для деталей “не из Парижа” с весом, большим десяти

```

SELECT  P.COLOR, P.CITY
FROM    P
WHERE   P.CITY <> 'Paris' AND  P.WEIGHT > 10 ;

```

Прежде всего, обратите внимание на использование в этом примере символа  $\neq$  (не равно). Обычные скалярные операторы сравнения в SQL записываются как  $\neq$ ,  $<$ ,  $>$ ,  $\leq$  и  $\geq$ .

Затем (что еще важнее) обратите внимание, что для используемых нами примерных данных этот запрос будет возвращать *четыре* строки, а не две, несмотря на то, что три из этих четырех строк *идентичны* и имеют вид (Red, London). **Язык SQL не удаляет лишних дублирующих строк из результата оператора SELECT, пока пользователь явно не потребует этого с помощью ключевого слова DISTINCT, как показано ниже:**

Это вызвано тем, что:

- 1) операция исключения повторяющихся значений может требовать значительных затрат;
- 2) пользователю часто не мешают полученные в результате повторяющиеся значения;
- 3) если исключение дубликатов логически необходимо в промежуточных результатах SQL выполнит это автоматически.

```
SELECT  DISTINCT P.COLOR, P.CITY
FROM    P
WHERE   P.CITY  $\neq$  'Paris'
AND     P.WEIGHT > 10 ;
```

Этот запрос будет возвращать только две строки.

Между прочим, в этом примере можно было прекрасно обойтись и без уточнителей (указатель на имя таблицы) “P”. Согласно общему правилу относительно **уточнения имени** в SQL неуточненные имена допускаются, если они не вызывают неоднозначности. Однако в наших примерах будут использоваться все уточнители, даже если они формально излишни (в определенных контекстах явно требуется, чтобы имена столбцов были не *уточнены*! (Например, это требуется в инструкции ORDER BY — см. ниже.)

И, наконец, заметьте, что последовательность строк в данной результирующей таблице в общем непредсказуема, *пока* пользователь не задаст некоторую определенную последовательность, как в этом примере:

```
SELECT  DISTINCT P.COLOR, P.CITY
FROM    P
WHERE   P.CITY  $\neq$  'Paris'
AND     P.WEIGHT > 10
ORDER   BY CITY DESC ;
```

В общем виде инструкция ORDER BY (упорядочить) записывается так:

```
ORDER BY order-item-commalist
```

Здесь список commalist не должен быть пустым, и каждый элемент order-item должен содержать не уточненное имя столбца, за которым (не обязательно) следуют ключевые слова ASC или DESC (которые означают возрастающий или убывающий порядок соответственно); по умолчанию принимается ASC (descent в переводе с английского «спускаться», ascent-«подниматься»).

#### 6.4.2. Для всех деталей получить номер детали и ее вес в граммах

```
SELECT P.P#, P.WEIGHT * 454 AS GMWT
FROM P ;
```

Спецификация AS GMWT вводит соответствующее имя результирующего столбца. Таким образом, два столбца результирующей таблицы будут называться P# и GMWT соответственно. Если бы спецификация AS GMWT была опущена, то соответствующий столбец был бы фактически безымянным. Обратите внимание, что хотя SQL на самом деле не требует от пользователя задавать имя результирующего столбца в таких случаях, но мы будем так делать в наших примерах всегда.

#### 6.4.3. Получить полную информацию обо всех поставщиках

```
SELECT *          -- или "SELECT S.*" (т.е. "*" можно уточнить)
FROM S ;
```

Результатом будет копия всей таблицы S; звездочка — это сокращение для списка всех имен столбцов в таблице (или таблицах), на которую делается ссылка в инструкции FROM, в порядке слева направо, как эти столбцы определены в таблице (таблицах). Обратите внимание на **комментарий** в этом примере (комментарии в SQL начинаются с *двойного дефиса* и заканчиваются символом новой строки).

Отметим, что звездочку очень удобно использовать в интерактивных запросах, так как вводится меньше символов. Однако использование звездочки во встроенном SQL (т.е. SQL в прикладных программах) потенциально опасно, так как смысл звездочки (\*) может измениться (например, в случае, если из таблицы был убран или в таблицу был добавлен столбец с помощью оператора ALTER TABLE).

*Замечание.* Согласно стандарту SQL/92 выражение SELECT \* FROM T (где T — название таблицы) можно упростить до TABLE T.

#### 6.4.4. Получить информацию обо всех парах поставщиков и деталей, совмещенных в одном городе

```
SELECT S.S#, S.SNAME, S.STATUS, S.CITY,
       P.P#, P.PNAME, P.COLOR, P.WEIGHT
FROM   S, P
WHERE  S.CITY = P.CITY ;
```

Концептуально можно рассматривать реализацию этой версии запроса следующим образом:

- Во-первых, после выполнения инструкции FROM мы получаем **декартово произведение** S TIMES P.
- Далее, после выполнения WHERE мы получаем **выборку** из этого произведения, в которой два значения CITY в каждой строке равны (иначе говоря, построено *соединение* поставщиков и деталей *по эквивалентности* городов).
- И, наконец, после выполнения оператора SELECT мы получаем **проекцию** выборки по столбцам, указанным в инструкции SELECT. Конечным результатом будет естественное соединение.

Следовательно, нестрого говоря, инструкция FROM в SQL соответствует декартову произведению, инструкция WHERE – выборке, а совместная инструкция SELECT – FROM – WHERE представляет проекцию выборки произведения.

6.4.5. Получить все пары имен городов, таких, что поставщик, находящийся в первом городе поставляет деталь, хранящуюся во втором городе

```
SELECT DISTINCT S.CITY AS SCITY, P.CITY AS PCITY
FROM   S JOIN SP USING S# JOIN P USING P# ;
```

Обратите внимание, что приведенная ниже инструкция уже будет некорректной, поскольку она включает столбец CITY как присоединяемый столбец во втором соединении:

```
SELECT DISTINCT S.CITY AS SCITY, P.CITY AS PCITY
FROM   S NATURAL JOIN SP NATURAL JOIN P ;
```

6.4.6. Получить все пары номеров поставщиков, таких, что оба поставщика в каждой паре размещаются в одном и том же городе

```
SELECT   FIRST.S# AS SA, SECOND.S# AS SB
FROM     S AS FIRST, S AS SECOND
WHERE    FIRST.CITY = SECOND.CITY
AND      FIRST.S# < SECOND.S# ;
```

Обратите внимание в этом примере на явные *переменные области значений* FIRST и SECOND. В наших предыдущих примерах все переменные об-

ласти значений были неявными. Также следует отметить, что вводимые имена столбцов SA и SB относятся к столбцам *результатирующей таблицы* и поэтому не могут быть использованы в инструкции WHERE.

#### 6.4.7. Получить общее число поставщиков

```
SELECT COUNT (*) AS N
FROM S ;
```

Результатом будет таблица с одним столбцом, названным N, и одной строкой, содержащей значение 5. Язык SQL поддерживает обычный набор **итоговых функций** (COUNT, SUM, AVG, MAX и MIN), но есть несколько специфических для SQL моментов, которые пользователю необходимо знать, а именно:

- Вообще аргументу функции не обязательно должно предшествовать ключевое слово DISTINCT, указывающее, что перед применением функции дублирующиеся строки должны удаляться. А для функций MAX и MIN ключевое слово DISTINCT излишне и не вызывает никакого действия.
- Специальная функция COUNT(\*), не допускающая использования слова DISTINCT, предназначена для подсчета *всех* строк в таблице без единого удаления дублирующихся строк.
- Любые null-значения в столбце аргументе всегда удаляются перед применением функции, в зависимости от того, указано ли слово DISTINCT, кроме случая COUNT(\*), когда null-значения обрабатываются точно так же, как обычные значения.
- Если аргумент будет пустым множеством, функция COUNT возвращает значение нуль; все другие функции возвращают null-значение (неопределенное значение). (В [8.19] показано, что такое поведение функций некорректно, но именно таким образом определяет их SQL.)

#### 6.4.8. Получить максимальное и минимальное количество для детали P2

```
SELECT      MAX ( SP.QTY ) AS MAXQ, MIN ( SP.QTY ) AS MINQ
FROM        SP
WHERE       SP.P = 'P2' ;
```

Здесь обе инструкции FROM и WHERE фактически предоставляют часть аргументов для двух итоговых функций. Следовательно, они должны были по логике вещей записываться в скобках, заключающих аргументы. Тем не менее, запрос действительно записывается так, как показано. Этот неорто-

доксальный подход к синтаксису оказывает значительное отрицательное влияние на структуру, удобство использования и ортогональность<sup>1</sup>

6.4.9. Для каждой поставляемой детали получить номер детали и общее количество поставки

```
SELECT  SP.P#, SUM ( SP.QTY ) AS TOTQTY
FROM    SP
GROUP   BY SP.P#
```

Это выражение в SQL является аналогом такого выражения реляционной алгебры:

```
SUMMARIZE SP BY ( P# ) AND SUM ( QTY ) AS TOTQTY
```

В частности, стоит отметить, что если указана инструкция GROUP BY, то выражения в инструкции SELECT должны быть *однозначными в группе*.

Вот альтернативная формулировка того же запроса:

```
SELECT  P.P#, ( SELECT SUM ( SP.QTY )
                FROM    SP
                WHERE SP.P# = P.P# ) AS TOTQTY
FROM    P;
```

Возможность использования вложенных операций выборки для представления скалярных элементов (например, в инструкции SELECT, как здесь) была добавлена в SQL/92 и является основным усовершенствованием по сравнению с первоначальным вариантом SQL. В рассматриваемом примере это дает возможность генерировать результат, который включает строки для деталей, не поставляемых совсем, а предыдущая формулировка (использующая инструкцию GROUP BY) этого не позволяла. (Однако значение TOTQTY для таких деталей будет, к сожалению, представлено как null-значение, а не нуль.)

6.4.10. Получить номера для всех деталей, поставляемых более чем одним поставщиком

```
SELECT  SP.P#
FROM    SP
GROUP   BY SP.P#
HAVING  COUNT ( SP.P# ) > 1 ;
```

---

<sup>1</sup> **Ортогональность** означает *независимость*. Язык является ортогональным, если независимые понятия сохраняют свою независимость и не смешиваются друг с другом непонятным образом. Ортогональность весьма желательна, поскольку чем менее ортогонален язык, тем он более сложен и, как это не парадоксально, менее мощен.



Инструкция HAVING для групп — это то же самое, что и инструкция WHERE для строк; другими словами, инструкция HAVING используется для исключения групп точно так же как инструкция WHERE используется для исключения строк. Выражение в инструкции HAVING должно быть однозначным для группы.

#### 6.4.11. Получить имена поставщиков, поставляющих деталь P2

См. аналогичные примеры в главах 6 и 7.

```
SELECT      DISTINCT  S.NAME
FROM        S
WHERE       S.S# IN
            ( SELECT  SP.S#
              FROM      SP
              WHERE     SP.P# = 'P2' );
```

*Пояснения.* В этом примере используется так называемый **подзапрос**. Проще говоря, подзапрос — это выражение SELECT-FROM-WHERE-GROUP BY-HAVING, которое вложено в другое такое же выражение. Подзапрос обычно используется для представления множества значений, поиск которых осуществляется с помощью инструкции **IN condition** (где condition — это условие), как проиллюстрировано в примере. Система вычисляет полностью запрос, вычислив сначала подзапрос (по крайней мере, концептуально). Этот подзапрос возвращает множество *номеров* поставщиков, поставляющих деталь P2: {S1, S2, S3, S4}. Таким образом, первоначальное выражение эквивалентно следующему более простому:

```
SELECT      DISTINCT S.SNAME
FROM        S
WHERE       S.S# IN ( 'S1', 'S2', 'S3', 'S4' );
```

Следует отметить, что первоначальную задачу — “Получить имена поставщиков, поставляющих деталь P2” — можно равносильно выразить с помощью операции *соединения*, например, так:

```
SELECT      DISTINCT S.NAME
FROM        S, SP
WHERE       S.S# = SP.S#
AND         SP.P# = 'P2' ;
```

#### 6.4.12. Получить имена поставщиков, поставляющих, по крайней мере, одну красную деталь

```
SELECT      DISTINCT S.NAME
FROM        S
```

```

WHERE          S.S# IN
              ( SELECT SP.S
                FROM    SP
                WHERE   SP.P# = IN
                      ( SELECT P.P#
                        FROM    P
                        WHERE   P.COLOR = 'Red' ) ) ;

```

Подзапросы могут иметь произвольную глубину вложения.

*Упражнение.* Приведите эквивалентную формулировку этого запроса с использованием операции соединения.

6.4.13. Получить номера поставщиков, статус которых меньше текущего максимального статуса в таблице S

```

SELECT      S.S#
FROM        S
WHERE       S.STATUS <
           ( SELECT MAX ( S.STATUS )
             FROM    S ) ;

```

В этом примере используются *две отдельные явные переменные области значений*, обозначенные одним и тем же символом “S” и изменяющиеся на таблице S.

6.4.14. Получить имена поставщиков, поставляющих деталь P2  
Такой же пример, как 6.4.11.

```

SELECT      DISTINCT S.SNAME
FROM        S
WHERE       EXISTS
           ( SELECT *
             FROM    SP
             WHERE   SP.S# = S.S#
             AND     SP.P# = 'P2' ) ;

```

*Пояснения.* Выражение SQL “EXISTS (SELECT ... FROM ...)” будет истинным тогда и только тогда, когда результат вычисления выражения “SELECT ... FROM ...” будет непустым. Другими словами, в SQL функция EXISTS соответствует *квантору существования* реляционного исчисления (более подробно это обсуждается в последующих главах книги).

6.4.15. Получить имена поставщиков, которые не поставляют деталь P2

```
SELECT  DISTINCT S.SNAME
FROM    S
WHERE   NOT EXISTS
        ( SELECT *
          FROM    SP
          WHERE   SP.S# = S.S#
          AND     SP.P# = 'P2' );
```

Или альтернативная формулировка:

```
SELECT  DISTINCT S.SNAME
FROM    S
WHERE   S.S# NOT IN
        ( SELECT SP.S#
          FROM    SP
          WHERE   SP.P# = 'P2' );
```

6.4.16. Получить имена поставщиков, поставляющих все детали

```
SELECT  DISTINCT S.SNAME
FROM    S
WHERE   NOT EXISTS
        ( SELECT *
          FROM    P
          WHERE   NOT EXISTS
                  ( SELECT *
                    FROM    SP
                    WHERE   SP.S# = S.S#
                    AND     SP.P# = P.P# ) );
```

Язык SQL не включает какой-либо непосредственной поддержки универсального квантора FORALL; следовательно, запросы типа FORALL обычно выражаются через отрицание кванторов существования, как в этом примере. Стоит отметить, что такие выражения, как только что показанное, хотя на первый взгляд и выглядят несколько устрашающе, легко составляются пользователями, знакомыми с реляционным исчислением.

Или с другой стороны, если эти примеры еще кажутся слишком сложными, то существует несколько “обходных” приемов, которые помогут избежать использования негативных кванторов. В нашем случае можно, например, записать:

```

SELECT  DISTINCT S.SNAME
FROM    S
WHERE   ( SELECT COUNT ( SP.P# )
          FROM    SP
          WHERE   SP.S# = S.S# ) = ( SELECT  COUNT ( P.P# )
                                     FROM    P );

```

(“имена поставщиков, для которых количество поставляемых деталей равно количеству всех деталей”). Однако обратите внимание на следующие обстоятельства.

- Во-первых, в последней формулировке предполагается, что нет номеров деталей в отношении SP, которые не содержатся в отношении P. Другими словами, эти две формулировки эквивалентны (а вторая — верная) только благодаря действию определенного ограничения целостности.
- Во-вторых, метод, примененный во второй формулировке (сравнение двух количеств), изначально не поддерживался в SQL и был добавлен в стандарт SQL/92. Он все еще не поддерживается на многих продуктах.
- Отметим также, что *на самом деле* предпочтительнее было бы сравнивать две *таблицы* (см. обсуждение реляционных сравнений в главе 6), и тогда запрос записывался бы так:

```

SELECT  DISTINCT S.SNAME
FROM    S
WHERE   ( SELECT SP.P# )
          FROM    SP
          WHERE   SP.S# = S.S# ) = ( SELECT  P.P#
                                     FROM    P );

```

Поскольку язык SQL непосредственно не поддерживает сравнения между таблицами, необходимо прибегнуть к приему, использующему сравнение кардинальных чисел (количество строк) таблиц (опираясь на практический опыт, мы убеждаемся, что если кардинальные числа таблиц одинаковы, то и таблицы одинаковы, по крайней мере, в обсуждаемом случае).

6.4.17. Получить номера деталей, которые или весят более 16 фунтов, или поставляются поставщиком S2, или и то и другое.

```

SELECT  P.P#
FROM    P
WHERE   P.WEIGHT > 16
UNION
SELECT  SP.P#
FROM    SP
WHERE   SP.S# = 'S2' ;

```

Лишние повторяющиеся строки всегда исключаются из результата безусловных операторов UNION, INTERSECT или EXCEPT (оператор EXCEPT в языке SQL служит аналогом операции MINUS реляционной алгебры). Но язык SQL также представляет уточненные варианты операторов: UNION ALL, INTERSECT ALL и EXCEPT ALL, при которых повторяющиеся строки (если они есть) сохраняются. Примеры с этими вариантами операторов умышленно опускаются.

В заключение отметим, что хотя список примеров выборок был довольно большим, тем не менее, многие возможности SQL даже не упоминались. Язык SQL действительно чрезвычайно *избыточный* в том смысле, что почти всегда представляется множество разных способов формулировки одного и того же запроса, и нам не хватит места, чтобы описать все возможные формулировки и все возможные опции даже для сравнительно небольшого числа примеров, которые рассматривались в этой главе.

### Операции обновления (запоминание\*) в sql

Как уже упоминалось, язык обработки данных DML включает три операции обновления: INSERT (вставка), UPDATE (изменение), DELETE (удаление). Приведем несколько простых примеров, которые, как мы полагаем, понятны без комментариев.

#### 6.4.18. Вставка одной строки

```
INSERT
INTO P ( P#, PNAME, COLOR, WEIGHT, CITY )
VALUES ( 'P8', 'Sprocket', 'Pink', 14, 'Nice' ) ;
```

#### 6.4.19. Вставка нескольких строк

```
INSERT
INTO TEMP (S#, CITY)
SELECT S.S#, S.SITY
FROM S
WHERE S.STATUS > 15 ;
```

#### 6.4.20. Удаление по заданному условию

```
DELETE
FROM SP
WHERE 'London' =
( SELECT S.CITY
FROM S
WHERE S.S# = SP.S# ) ;
```

Язык SQL включает в себя открытый список библиотечных функций: sum, min, avg, max, count и др, которые могут быть использованы как в предикате, так и во фразе с SELECT.

*Язык SQL обладает реляционной полнотой.*

*Для обеспечения реляционной полноты при реализации языка необходимы две операции:*

1. *Операция присваивания*, т.е. возможность создавать новые отношения для хранения результатов операций реляционной алгебры, являющихся также отношениями.
2. *Операция транзитивного замыкания*, когда допускаются вложенность операций реляционной алгебры для построения выражений любой произвольной сложности.

## 6.5. Язык запросов QBE

QBE – QUERY BY EXAMPLE (запрос с примером) – реляционный язык, разработанный М.М.Цлуфтом в научно-исследовательской лаборатории IBM, представляет языковое средство манипулирования данными.

Отличительной особенностью языка запросов QBE является его использование главным образом для работы с терминалом:

1. Всякая операция в QBE специфицируется с помощью одной или нескольких таблиц. Причем, каждая такая таблица строится на экране дисплея *частично системой, частично пользователем.*
2. QBE имеет *двумерный синтаксис*, т.е. операции в QBE задаются в табличной форме.

### Операции выборки языка QBE

Значительной особенностью языка запросов QBE является использование примеров для спецификации запросов. Технология запроса состоит в том, что пользователь заносит пример *возможного ответа* в соответствующее место пустой таблицы. Рассмотрим примеры на примере схемы отношений, приведенной выше:

#### 6.5.1. Получить номера поставщиков, находящихся в «Томске».

1. Первоначально система выдает на экран пользователя пустую двумерную таблицу:

<b>S</b>	

2. Пользователь, зная, что ответ на запрос находится в таблице S, запишет S в качестве имени таблицы в левом верхнем углу, после чего система ответит заполнением соответствующих имен столбцов:

Табл.1.

S#	Симя	Область	Город

3. Пользователь формирует запрос, заполняя две позиции в табл.2.

Табл.2.

S#	Симя	Область	Город
	<b>P. S7</b>		<i>ТОМСК</i>

«P.»(print) - означает печать; Эта литера указывает цель запроса, т.е. обозначает поле, значение которого необходимо определить в результате выполнения запроса.

«S7» - является *элементом-примером*, т.е. возможным примером ответа на запрос. Элемент-пример выделяется подчеркиванием (  ).

*ТОМСК* – заданная известная константа.

Этот пример может быть интерпретирован следующим образом:

*Напечатать все значения S7, такие как, может быть S7, для которых значение атрибута ГОРОД="ТОМСК". Причем, в результирующем множестве (или даже в исходном) не обязательно присутствует значение «S7», т.к. элемент-пример является полностью произвольным (мы могли использовать для обозначения экземпляра-примера, например, литеру «X»).*

Элемент-пример может опускаться в простых запросах, как в нашем примере, так что **P. S7** сократилось бы до **P.**

**Т.о. элемент-пример используется для установления связей между строками в сложных запросах.**

6.5.2. Простая выборка «Получить номера всех поставляемых деталей».

SP	S#	P#	КОЛ-ВО
		<b>P. PX</b>	

По желанию пользователя можно задать возрастающее или убывающее упорядочивание результата. В запросах на языке QBE повторяющиеся значения из результата *всегда* исключаются.

6.5.3. «Получить данные о всех поставщиках».

S	S#	Симя	Обл.	Город
	<b>P. SX</b>	<b>P. SI</b>	<b>P. SO</b>	<b>P. SG</b>

Этот запрос можно реализовать следующим образом:

S	S#	Симя	Обл.	Город
<b>P.</b>				

Здесь операция печати (**P.**) относится ко всей строке (записи).

6.5.4. «Выборка по условию». Получить номера поставляемых деталей «винт», имеющих вес  $\geq 14$ .

P	P#	РИмя	Цвет	Вес	Город
	<b>P. PX</b>	винт		$\geq 14$	

Для задания условия может использоваться любой из операторов сравнения ( $<$ ;  $>$ ;  $\leq$ ;  $\geq$ ;  $=$ ;  $<>$ , знак « $=$ » из условного выражения обычно опускается).

6.5.5. «Перечислить номера поставок, имеющих имя "винт" или вес  $\geq 14$  (или) удовлетворяющих одному и другому условию.»

P	P#	РИмя	Цвет	Вес	Город
	<b>P. PX</b>	винт			
	<b>P. PY</b>			$\geq 14$	

Условия, заданные в одной строке, рассматриваются как операнды операции «И». Для «ИЛИ» - условия необходимо задавать в разных строках, кроме того, использованы два разных экземпляра-примера (**PX**, **PY**), т.к. использование одного примера означало условие наличия у поставки и имя "винт" и вес  $\geq 14$ .

Когда запрос содержит более чем одну строку, их можно расположить в любом порядке.

6.5.6. «Получить номера тех поставщиков, которые поставляют деталь P1 и деталь P2».

SP	S#	P#	Кол-во
	<b>P. SX</b>	P1	
	<b>SX</b>	P2	

В запросе использован один и тот же экземпляр-пример (**SX**) и две строки, т.к. необходимо соединить операцией «И» два условия для одного столбца.



6.5.7. Выборка с использованием связи. «Получить имена поставщиков, поставляющих деталь P2».

S	S#	Симя	Обл.	Город
	<b>SX</b>	<b>P. SI</b>		

SP	S#	P#	Кол-во
	<b>SX</b>	P2	

Элемент-пример “SX” используется для связи между таблицами S и SP. Таким образом, реализуется вложенность запроса SQL.

Операции обновления и удаления реализуется с помощью использования служебных слов DELETE и UPDATE.

6.5.8. Удалить все сведения о поставщиках.

S	S#	Симя	Обл.	Город
<b>DELETE.</b>				

## ГЛАВА 7. СИСТЕМА FOXPRO

FoxPro является системой управления данными, обеспечивающей организацию хранения данных линейной структуры, формирование и корректировку хранимых данных, многоаспектную выборку данных и широкие возможности по их обработке. Система имеет мощные инструментальные средства по программированию и отладке, что позволяет отнести ее к средствам программирования, ориентированным на информационные задачи обработки данных.

### 7.1. Основные ограничения

Основной файл (тип .DBF) может содержать до миллиона однотипных записей, каждая из которых может содержать до 255 полей. Повторяющиеся групповые поля недопустимы. Объем файла в байтах не более двух миллиардов. Максимальный размер записи - 6400 байт.

1. Для описания данных в системе FoxPro приняты 6 типов данных полей:

*Тип N* - числовое поле. Целое или с десятичной точкой. Максимальный размер 20 знаков, включая знак числа и точку, причем целая часть не более 16 цифр.

*Тип F* - с плавающей точкой, при внешнем отображении имеет также ограничения, что и тип N.

*Тип C* - символьное поле, размером не более 254 байт, может содержать любые символы.

*Тип D* - дата и может иметь одну из нескольких форм представления, по умолчанию при вводе и выводе ММ/ДД/ГГ. Возможны и другие: ДД/ММ/ГГ; ГГ/ММ/ДД, а также (.) вместо (/) и ГГГГ вместо ГГ. Размер поля всегда 8 байт, хранение в формате ГГ/ММ/ДД.

*Тип M (memo)* - текстовое, размером > 254 (любой длины), допускает любые символы, хранится в специальном файле FPT, а в записях файла .DBF хранится адрес (10 байт) - номер блока в файле .FPT, где содержится значение поля.

*Тип L* - логическое с возможными значениями True (T), False(F) - "истина", "ложь", или Y,N (да, нет) - размер поля всегда 1 байт.

2. *Временные переменные* - это переменные, хранимые и используемые вне структуры БД. Максимальное число активных временных переменных - 256, могут сохраняться на диске в специальном файле типа .MEM, общим объемом до 64000 байт.

Временным переменным могут присваиваться значения выражений и сами они могут быть элементами выражений.

Тип временных переменных и размеры памяти, отводимые для них, точно такие же, как и для соответствующих типов полей.

### 3. Константы. Числовые, символьные и логические.

*Числовая* - 20 цифр, включая знак и (.), если они необходимы.

*Символьная* - последовательность символов, заключенных в апострофы или кавычки, или в квадратные скобки - 'AAA', "AAA", [AAA].

*Даты* - в [ ].

*Логическая* - T и F, Y и N при наборе на клавиатуре выделяется точками .T., .F., .Y., .N.

4. *Функции*. Тип функций - числовые, символьные, логические и даты. В системе реализовано 70 стандартных функций.

5. *Выражения*. Конечная последовательность данных, констант, переменных и функций, соединенных операторами, определяющими операции, которые нужно выполнять над элементами для получения нового значения выражений. Элементы выражения должны быть одного типа. Допустимые типы выражений - арифметические (математические), логические, строковые и дат.

#### 6. 4 класса операций:

- вычислений - (,), \*\* и ^ (возведение в степень), \*, /, +, - ;
- строковые - сцепление строк (+) и сцепление строк с удалением пробелов (-). Выполняются над символьными типами данных;
- сравнения >, <, =, <> и # (не равно), >=, <= ;
- == сравнения символьных строк;
- логические (,) , .NOT. , .AND. , .OR.;
- сравнение подстрок - \$.

Приоритет классов и собственно операций в классе соответствуют приведенному порядку.

7. Имена файлов - до 8 символов, полей и временных переменных до 10 символов. Могут содержать латинские буквы, цифры и символ подчеркивания (\_). Должны начинаться с буквы, не иметь внутри пробелов. Избегать употребления в качестве имен ключевых слов и букв от A до J.

8. Файлы - 28 видов, из них три уже рассмотрели - .DBF ,.FPT , .MEM .

.TXT - могут содержать только печатные символы и используются для обмена с другими системами. Файл TXT может быть создан любым редактором.

.IDX - для образования логического упорядочивания в .DBF отличного от физического, суть-значение некоторого поля (ключа) в записи связывается с номером записи, и эти пары упорядочиваются по возрастанию значения ключа. За счет этого обеспечивается возможность прямого доступа к записям файла .DBF в порядке возрастания значений полей ключа.

.PRG -командный файл (программа на языке FoxPro).

.FXP - откомпилированная программа FoxPro.

.CAT - традиционное использование каталога для сокращения поиска "своих" файлов.

.FMT - содержит команды создания экранных форм, используемых для вывода или редактирования содержимого файлов .DBF .

.LBX - описание метки, предназначенное для выдачи информации файла .DBF в виде ярлыков, этикеток, меток, почтовых открыток и т.п.

.LBV - содержит параметры установки среды при сохранении описания метки.

.FV - содержит изображения экранных форм и служит для генерации и модификации файлов формата (.FMT).

.VUE -хранит полную информацию о состоянии среды FoxPro, ( в том числе параметры установки среды, файлы, псевдонимы, связи файлов).

.FRX - содержит описание отчета.

.FRV - содержит параметры установки среды при сохранении описания отчета.

.BAK - предыдущая версия файлов TXT, PRG, FXP или DBF.

.TBK - предыдущая версия файла FPT (с мемо-полями).

Кроме того FoxPro имеет специальные файлы.

FOXUSER.DBF - файл ресурсов.

FOXHELP.DBF и FOXHELP.FPT - файлы помощи, содержат справочную информацию о FoxPro.

Во время выполнения программы система FoxPro создает на жестком диске несколько временных файлов (начинаются с цифр 0, 1 или 2 и не имеют расширений), а при нормальном завершении работы FoxPro - они уничтожаются автоматически).

Следует отметить, что пользователь работает с ограниченным числом файлов. Прежде всего, это файлы DBF, IDX, FPT, FMT, FRX, BAK, TBK.

Одновременно может быть открыто до 25 файлов (если используются текстовые поля, то здесь учитывается, кроме .DBF и файл .FPT) до 25 открытых индексных файлов, 1 файл .FMT на 1 .DBF.

## 7.2. Типовые технологии создания и обработки баз данных

**СОЗДАНИЕ НОВОГО ФАЙЛА.** Выбирается опция *New* в вертикальном меню опции *File* полосы меню экранного интерфейса. В результате на экране появляется окно диалога рис. 7.1 со списком типов файлов (в виде списка селективных переключателей), которые можно создать в текущий момент времени. По умолчанию считается выбранным тип *Database*, т.е. dbf-файл. Активизируем кнопку ОК, после чего система переходит в окно диалога (рис. 7.2), поддерживающего процесс создания структуры dbf-файла, пока со стандартным именем *Untitled* (безымянный). Фактически запущена команда *Create*, в чем можно убедиться по трансляции данной команды в командном окне.

Пользователю предлагается последовательно для каждого значения поля, начиная с первого, ввести: имя поля (до 10 символов), тип, размер и точность (для числовых) значений данных.

Начиная с этого момента, приводимые окна диалога будут отображать действие по работе с конкретными файлами. В частности, далее иллюстрируется процесс создания файла **ONE.DBF** с полями: CODE, FULL\_NAME, ORGANIZ, ADDRESS, NAME\_RUK, NUMBER\_TLG, CH\_MEST, SEASON, CH\_PAC\_Y.

```

| New:
| ( ) Database
| ( ) Program
| ( ) File
| ( ) Index      < OK >
| ( ) Report
| ( ) Label      < Cancel >
| ( ) Screen
| ( ) Menu
| ( ) Query
| ( ) Project

```

Рис. 7.1. Выбор типа файла

```

| Structure: Untitled
|   Name      Type      Width  Dec
|   Field
| ☐ CODE      Numeric  1      0
| ☐ FULL_NAME Character 40      <Insert>
| ☐ ORGANIZE Character 30
| ☐ ADDRESS Character 15      <Delete>
| ☐ NAME_RUK Character 15
| ☐ NUMBER_TLG Character 10
| ☐ CH_MEST Numeric   5      0
| ☐ SEASON  Float    0      < OK >
| ☐ CH_PAC_Y Date     8
|                                     <Cancel>
| Fields: 9   | Picture | Available: 3846 |

```

Рис. 7.2. Создание структуры .dbf-файла

Меню типов данных вызывается клавишей «ПРОБЕЛ» после ввода имени поля (когда становится активным поле TYPE).

Поле **DEC** (точность) имеет смысл только для числовых данных, а поле размер (WIDTH) автоматически назначается: для логических – «=1», дат – «=8» и Мемо-«=10» позиций.

По умолчанию для символьных и числовых данных размер установлен равным 10, пользователь, при необходимости, может его скорректировать.

Размер для Мемо полей не указывается (остается 10).

Если в процессе ввода описаний полей возникает необходимость вставить пропущенное описание поля, необходимо подвести курсор на описание поля, непосредственно за которым следует вставить пропущенное, а затем по TAB перейти к кнопке INSERT и активизировать ее. В описании появится

строка с именем данного NEW FIELD. Пользователь задает необходимое имя (исправляя New field), тип и размер данных.

Аналогично, для удаления описания необходимо вначале установиться на него курсором, а затем перейти с помощью клавиши TAB на кнопку DELETE и активизировать ее. Если пользователь намерен отказаться от создаваемого описания полностью, он должен активизировать кнопку CANCEL, после чего система запрашивает

**Discard structure Change?**  
**отказаться от изменений**  
 <Yes>            <No>

Если активизирована кнопка **No** – происходит возврат в окно ввода структурных характеристик.

Если активизирована кнопка **Yes** - происходит отмена всех созданных описаний и выход в начальное окно интерфейса FoxPro. В командном окне остается команда **CREATE UNTITLED**, ее можно вновь запустить, установив на нее курсор и нажав клавишу ENTER.

При выборе ОК (нормальное завершение формирования описания структуры файла) система переходит в окно определения имени файла (переопределение untitled). Для размещения создаваемого файла пользователь может выбрать любой диск (активизировав переключатель Drive), директорию (активизировав переключатель Directory) и указать (ввести с клавиатуры) имя файла в текстовом окне ввода имени вместо untitled.

Пользователь может отказаться от всех установок, выбрав кнопку Cancel, либо используя режим Save записать созданный файл (пока пустой) в заданную директорию. В последнем случае (выбор Save) система спросит, не намерен ли пользователь вводить значения данных (формировать записи файла).

**Input data record now?**  
 <<Yes>>        <No>

Если пользователь отвечает <No> - происходит возврат в окно описания структуры файла, <<Yes>>- система переходит в окно создания и редактирования записи, что соответствует запуску команды Append.

Окно создания-редактирования записей в режиме Append (рис. 7.3.) содержит специальную, анкетную форму представления записи в виде вертикально расположенных пар "имя данного-окно под занесение данного", причем курсор установлен в окне значение первого данного.

**System File Edit Database Record Program Window Brow  
ONE**

**Code  
Full\_name  
Organiz  
Address  
Name\_ruk  
Number\_tlg  
Ch\_mest  
Season  
Ch\_pac\_y**

Рис. 7.3. Ввод данных в файл данных .dbf

### **7.3. Диалоговая среда системы FoxPro**

Диалоговая среда (ДС) системы FoxPro построена по принципу много-оконного отображения информации (число создаваемых в сеансе окон ограничивается только объемом ОП), основывается на меню, командах и блоках диалога.

Всплывающие меню содержат команды. Команды могут завершаться многоточием (. . .), при выборе таковых появляются блоки диалога (диалоги) для ввода дополнительной информации.

Выбор команды в системе меню сопровождается генерацией (отображением) команд входного языка СУБД в окне команд (Command).

Другой способ взаимодействия - непосредственный ввод команд в окно Command.

ДС поддерживает несколько типов системных окон:

- окно команд (Command);
- окно помощи (Help);
- окно просмотра (View);
- окно редактирования и др.

( С помощью главного меню и его пункта (опции) Windows Вы можете изменить конфигурацию системного окна.)

#### **7.3.1. Блоки диалога**

Блок диалога содержит следующие средства управления:

**Текстовая кнопка < действие >** содержит действие, выполняемое сразу после выбора. Если текстовая кнопка сопровождается многоточием (...) появится другой блок диалога (рис. 7.4).

**Триггерная кнопка** [ x ] < атрибут > - включает/выключает атрибут с помощью клавиши пробела ( [x] - атрибут включен ).

Несколько триггерных кнопок могут быть заданы одновременно.

**Радиоклавиша** ( · ) < атрибут > . Радиоклавиши объединены в группу. Может быть выбрана только одна клавиша путем перемещения точки.

**Кнопка управления всплывающим меню** | | .

При выборе этой кнопки на экран выводится соответствующее меню.

Выбор осуществляется с помощью клавиш TAB и Space.

**Прокручивающийся** список содержит перечень элементов для выбора. Для фиксации выбора необходимо нажать клавишу Space (пробел).

**Текстовый блок** - предназначен для ввода и редактирования текста.

*Перемещение в блоке диалога с помощью клавиатуры:*

- Tab - выбирает следующее управляющее средство;
- Shift-Tab - выбирает предыдущее управляющее средство;
- PgDn, PgUp - клавиши управления прокручиваемым списком
- Esc - выход

Пример блока диалога. Открытие файла

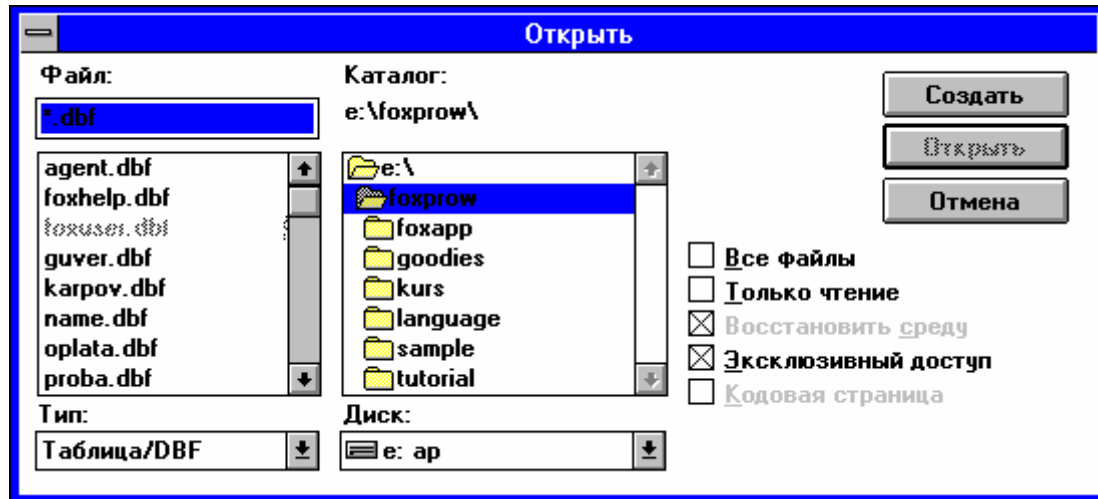


Рис. 7.4

### *Системные окна*

К системным окнам относятся:

**Command** - открывается автоматически; - повторное открытие - Window/Command;

**Debug** (отладка), **Trace** (трассировка), **View** (вид) - могут быть открыты в меню Window;



**Browse** (просмотр) - открытие в меню Database;  
**Program** (программа), **Text** (текст), **Report Layovt** (макет отчета), **Label Layovt** (макет метки), **Memo** (комментарий) открываются в меню File/Open.

Имена открытых системных окон отображаются в меню Window.

Закрыть системное окно можно двумя способами: при помощи клавиши **Esc** (кроме Command) или выбора пункта меню **File/Close**.

#### *Операции с окнами (меню Window):*

- **Hide** (скрытие) - делает окно невидимым, не закрывая его. Чтобы отобразить открытое окно, необходимо выбрать его имя в меню Window;
- **Zoom** (распахивание окна) - размеры окна увеличиваются до полного экрана. Распахнуты могут быть все системные окна, кроме View;
- **Cycle** (циклическое переключение) - команда выполняется до тех пор пока требуемое окно не станет текущим;
- **Scroll** (прокрутка);
- **Size** (изменение размера);
- **Move** (перемещение).

#### *Текстовый редактор (меню Text)*

Основная работа с данными осуществляется посредством программных файлов. В среде СУБД FoxPro командный файл представляется в виде обычного текстового файла, каждая строка которого содержит определенную команду в формате входного языка системы. В связи с этим СУБД FoxPro имеет встроенный редактор и пользователям СУБД необходимо знать приемы работы с ним.

Рассмотрим основные возможности встроенного редактора текстов FoxPro. Редактор активизируется с помощью меню Text DC, либо при вводе команды **Modi Comm** в окне Command.

!!! Обратите внимание - операции с файлами производятся в меню File.

**New** - создание нового файла.

( По умолчанию новый файл имеет имя *Untitled*. Для переименования выполните блок диалога Save as.)

**Save** - периодическое сохранение файла.

**Open** - открытие существующего файла.

**Close** - закрыть отредактированный файл.

#### *Выделение фрагментов текста*

Текст можно выделить одновременным нажатием Shift и клавиши управления курсором. Для выделения текста используйте также комбинации клавиш:

Shift - Ctrl - End - от позиции курсора до конца текста;  
 Shift - Ctrl - Home - от позиции курсора до начала текста;  
 Ctrl - A - весь текст.

Для отмены выделения нажмите любую клавишу перемещения.

#### *Основные команды редактора (меню **Edit**):*

- **Cut** (вырезать) - фрагмент выделенного текста удаляется с экрана и помещается в буфер для дальнейшего использования;
- **Copy** (копировать) - выделенный текст копируется с сохранением в буфере;
- **Paste** (вклеить) - перед вклеиванием установить курсор в точку вставки и выполнить **Paste**. Возможно многократное вклеивание.

Операции по вырезанию и вклеиванию можно выполнить между окнами (число открытых окон нерегламентированно):

- вырезать текст;
  - выбрать окно (меню Window);
  - установить курсор в место вставки текста;
  - выполнить Paste.
- **Undo** - режим отката (т.е отмена последней введенной команды);
  - **Redo** - отмена отката;
  - **Find** - поиск и замена;
  - **Preferences** (приоритеты) - установка параметров приоритетов в работе редактора.

## 7.4. Опции главного меню системы FoxPRO

### 7.4.1. Меню FILE

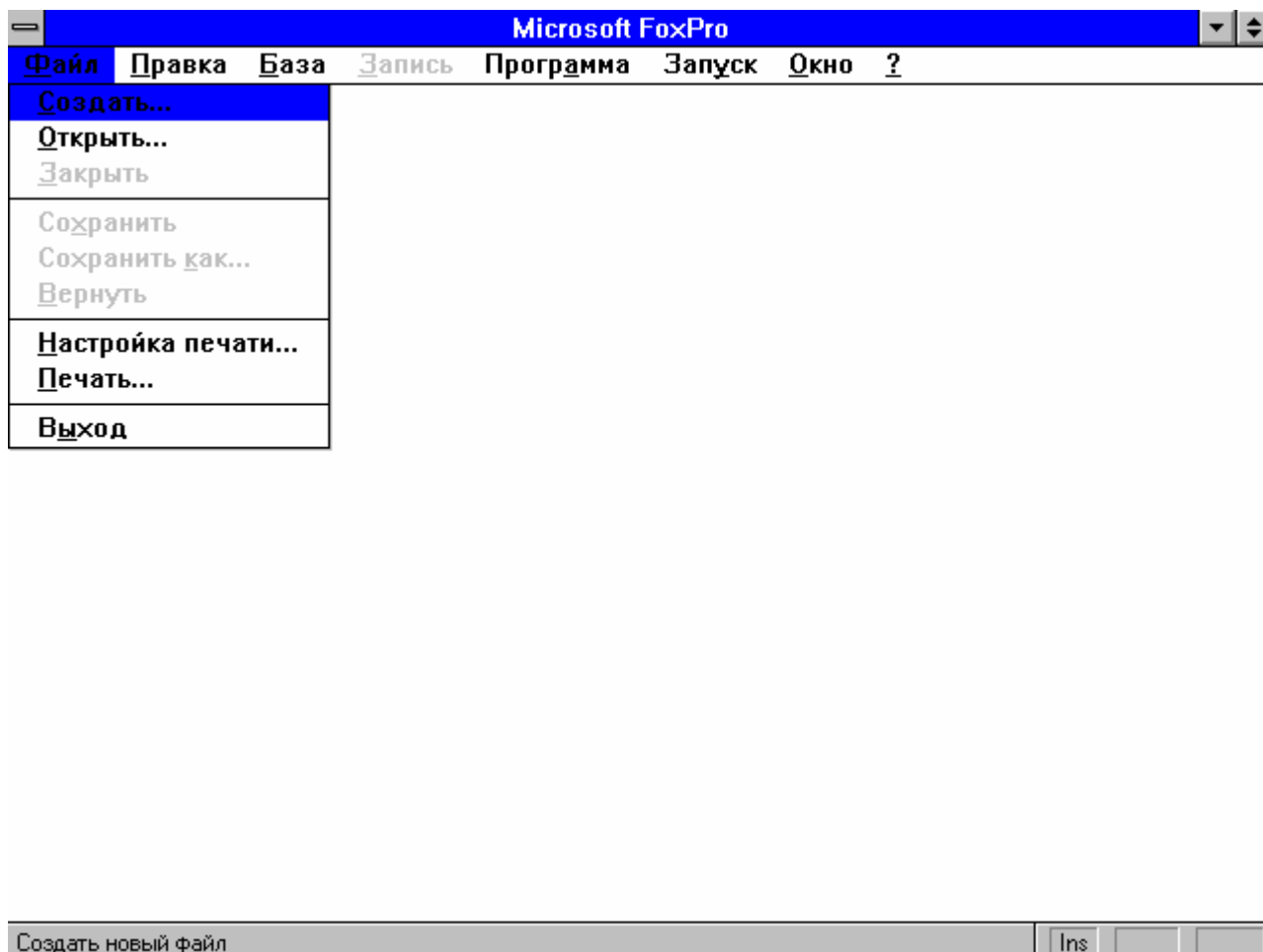


Рис. 7.5

- **New...** - позволяет создать и открыть новый файл.

Существует 6 типов файлов для создания:

Database - файл данных .dbf;  
 Program - файл программ .prg;  
 Text - текстовый файл .txt;  
 Index - индексный файл .idx;  
 Report - файл отчета .frx;  
 Label - файл метки .lbx.

При создании файла данных выполняется диалог **Structure (Setup-Настройка)** (рис. 7.6).

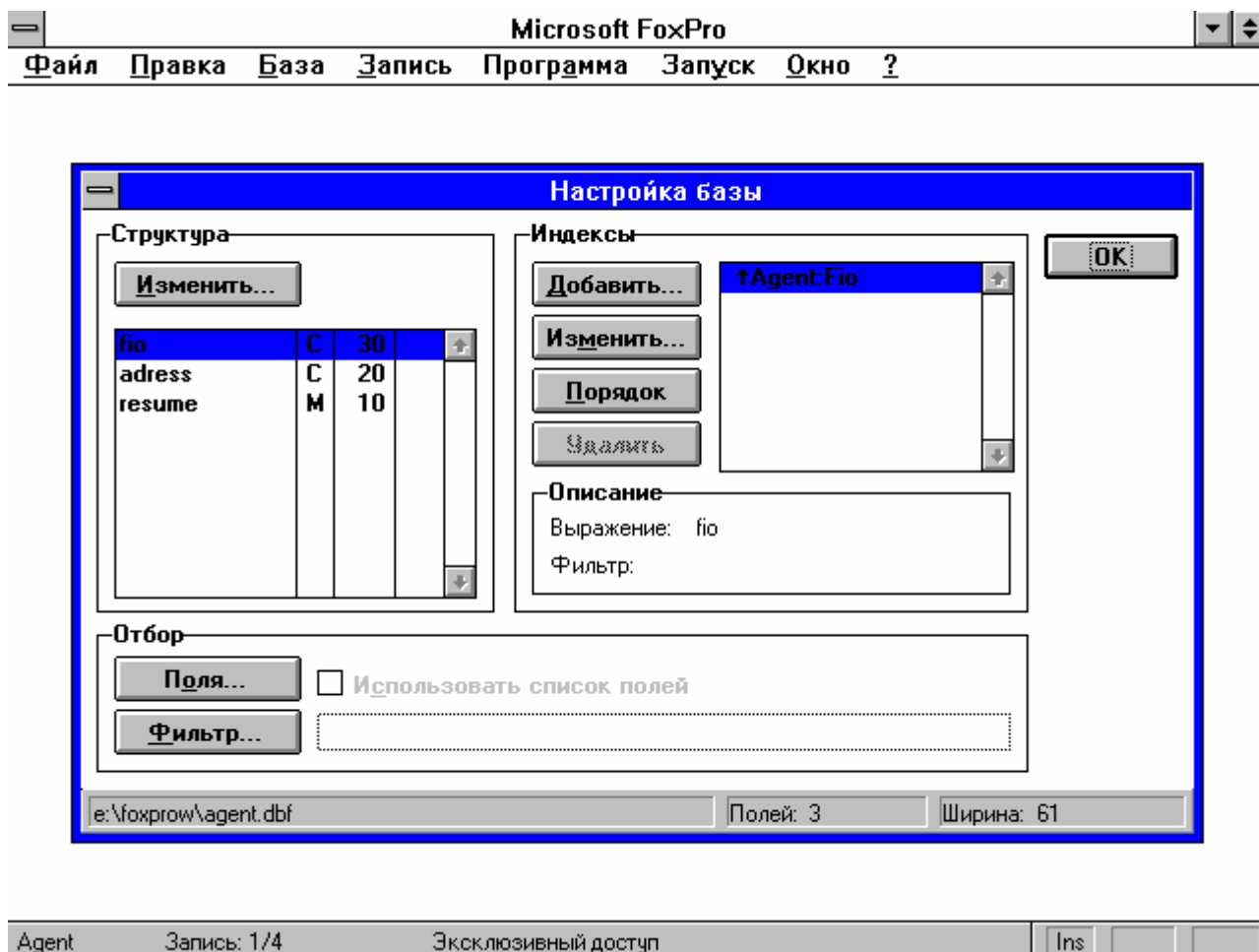


Рис. 7.6

Если база данных не открыта, переключатель Index (Индексы) блокируется, в противном случае активизируется диалог Индекса для определения индексного выражения.

*Способы создания индексного выражения:*

- 1) ввод выражения в текстовый блок справа от переключателя Expr...;
- 2) выбор ключа индексирования из прокручиваемого списка, если индекс строится по одному полю;
- 3) выбор и использование Построителя выражения (клавиша Expr...).

Построитель выражения - это диалог, используемый FoxPro всякий раз, когда для окончательного формирования команды необходимо выражение. Как правило выражения используются для создания индексов, установления зависимостей или задания операторов For и While.

Построитель имеет несколько вспомогательных меню.

Меню Expression содержит три типа опций:

- 1) опции Math Functions, String Functions, Logical Functions и Date Functions и отображают меню для выбора функций, подставляемых в выражение;

2) опции Field list, Variable list и Database высвечивают списки полей активной базы данных, доступных переменных памяти, имен файлов активных баз данных;

3) опции Verify выводит сообщение о корректности построенного Вами выражения.

- **Close** - закрывает активное окно.
- **Save** - сохраняет все изменения, которые Вы сделали в активном текстовом или программном файле. Файл при этом не закрывается. Если активны несколько окон, на диске сохраняются изменения только для текущего окна редактирования.
- **Save as...** - сохраняет новый файл под именем, заданным в диалоге Save as, либо делает копию существующего файла под новым именем.

Вы можете сохранить текущую среду под новым именем с расширением .VUE. Для этого активизируется окно Представления (View) и выполняется диалог Save as.

- **Revert** - замена текущей версии программного или текстового файла на предыдущую.
- **Printer Setup...** - отображает диалог Установки параметров принтера.
- **Print...** - высвечивает диалог Печати, в котором устанавливаются такие параметры, как число печатных копий, ...
- **Quit** - выход из FoxPro, завершение сеанса.

#### 7.4.2. Меню DATABASE

Setup...	<b>Setup...</b> - установка параметров рабочей области
Browse	

В диалоге установки Вы можете:

Append from...	- определить какие индексные файлы использовать с базой данных;
Copy to...	- изменить структуру БД;
Sort...	- выбрать список полей для доступа;
Totul...	- определить фильтр;
	- выбрать форманный файл для ввода данных.
Average...	
Count...	
Sum...	
Calculate...	
Report...	
Label...	
Pack	
Reindex	

### *Работа с индексами*

#### **Открыть индекс:**

выбрать переключатель Add...(Добавить) и выполнить диалог Открытия файла;

для открытия нескольких idx-файлов повторить предыдущее действие.

Открытые индексные файлы появляются в списке индексов. Помеченный переключателем индекс является **главным индексом**, определяющим порядок в базе.

#### **Закрыть индекс:**

выбрать индекс из списка;

выполнить Remove;

ОК.

#### **Создать индекс:**

выбрать Add... (Добавить. . .);

выбрать New в диалоге Открытия;

выполнить диалог Индекса.

#### **Модифицировать индекс:**

выбрать индекс из списка;

выбрать Modify (Изменить);

изменить выражение индексного ключа в текстовом блоке;

изменить оператор For; ОК.

### *Модификация структуры БД*

Под модификацией структуры понимается добавление и удаление полей, изменение имен полей, их размеров и типов.

Последовательность действий:

установиться на список Структуры;

выбрать Modify (<Изменить>);

выполнить диалог Структуры;

ОК.

### *Установка полей*

Двоичный переключатель Set Fields...( <Поля . . .> ) позволит задать активные поля в выделенном файле БД.

В диалоге Указателя полей размещены шесть текстовых переключателей:

Move - копирует **выделенное** поле из Списка полей БД в список Выделенных полей. Выделенные поля в списке полей отмечаются указателем.

All - копирует все поля.

Remove - удаляет поле в списке выделенных полей.

Clear - очищает список выделенных полей.

Cancel - выход из диалога Указателя полей с возвратом в диалог Установки.

ОК - выполнить все установки.

Переключатели on/off регулируют доступ к полям открытых БД.

ON - доступ разрешен в соответствии Set Fields. OFF - доступ разрешен по всем полям.

### *Установка фильтра*

Фильтр устанавливается переключателем Filter, с последующим выполнением Построителя выражения.

### *Выбор форматного файла (переключатель Format)*

После выбора Format в диалоге Установки появляется диалог Открытия файла. Установите нужный формат отображения записей файла БД.

В диалогах Change и Append стандартное окно заменяется установленным.

**Browse** - просмотр и/или редактирование активной базы данных в окне просмотра.

Окно просмотра является уникальным, т.к. его можно разделить на два участка и в одно и то же время просматривать различные части базы данных.

**Меню Browse** (добавляется к строке меню при выборе опции Browse):

**Change** - режим внесения изменений; если окно разделено, change влияет только на активный участок.

**Grid off** - удаление вертикальных линий в окне просмотра.

**Unlink Partitions** - просмотр участков независимо друг от друга.

**Change Partition** - изменение состояния участков активный/неактивный.

**Size Field** - выделенное поле (Tab) изменяет размеры. Если окно разделено, опция действует на обоих участках.

**Move Field** - перемещение поля для изменения порядка следования полей.

**Resize Partitions** - разделитель участков. Для изменения размеров участков используйте клавиши управления курсором.

**Enter** - завершение разделения.

**Toggle Delete** - установка точечного указателя для маркированных на удаление записей.

**ВНИМАНИЕ!** Все корректировки меню Browse касаются только окна просмотра и не влияют на файл БД.

Активный участок отмечен мерцающим курсором. Все выполняемые действия над записями влияют только на активный участок и отображаются как в окне просмотра, так и в реальном файле БД.

### *Функции редактирования данных*

#### 1. Изменение данных в поле.

Возможны любые операции редактирования, как описано в разделе "Текстовый редактор".

#### 2. Добавление записей.

Записи могут добавляться либо между существующими записями, либо в конец файла БД. Для добавления в конец файла выполните Browse/Append Record либо Record/Append. Возврат в режим просмотра через меню Browse. Для добавления между записями новых данных введите команду Insert в окно команд.

#### 3. Удаление записей.

Выделите запись и выполните опцию Toggle Delete меню Browse.

Для физического удаления записей из БД выполните опцию Pack меню Database.

### **Append From...** - добавление записей из файла.

Если Вы знаете имя базы данных, записи из которой хотите скопировать, введите ее имя в текстовый блок. В противном случае, выберите текстовый переключатель From... для выбора требуемого файла. From работает с файлами типа .dbf. Для указания другого типа файла используйте меню Type:

Database - стандартный файл данных .dbf;

Delimited with Tabs - текстовый файл, в котором каждое поле отделяется метками табуляции;

Delimited with Commas - текстовый файл, поля отделяются запятыми;

Delimited with Space - текстовый файл, поля отделяются пробелом;

SDF - текстовый файл, в котором записи имеют фиксированную длину и завершаются возвратом каретки.

Интервал и/или условия добавления задаются с помощью переключателей Scope, While, For.

**Copy To...** - копирование содержимого БД из открытого файла данных в новый файл.

**Sort..** - сортировка БД с созданием нового файла

**Total...** - вычисление итоговых сумм для числовых полей

**Average...** - вычисление среднего значения

**Count** - подсчет записей БД

**Sum** - сумма цифровых полей

**Calculate** - финансовые и статистические операции



- Report** - генератор отчетов. Файлы отчетов создаются с использованием программы FoxReport
- Label** - генератор метки
- Pack** - физическое удаление
- Reindex** - модификация индексного файла

### 7.4.3. Меню RECORD

Меню Record предназначено для выполнения операций с записями:

Append	
Change	
<hr/>	
Goto...	<b>Append</b> - добавляет пустые записи в конец активной БД. Экран ввода может быть определен пользователем;
Locate...	
Continue..	<b>Change</b> - корректировка записей БД;
Seek...	<b>Goto</b> - размещение указателя записи в указанную запись;
<hr/>	
Replace...	<b>Locate</b> - последовательный поиск;
Delete...	<b>Continue</b> - продолжение поиска;
Recall...	<b>Seek</b> - индексный поиск;
	<b>Replace</b> - модификация полей в записях БД (модификация одной или нескольких записей в открытых базах данных);
	<b>Delete</b> - маркировка на удаление;
	<b>Recall . . .</b> - отмена маркировки.

### 7.4.4. Меню PROGRAM

Опции меню Program используются для выполнения программных файлов системы FoxPro.

- Do** - активизация программного файла
- Cancel** - прекращение выполнения программы и закрытие командного файла
- Resume** - повторный запуск приостановленной программы
- Echo** - вывод исходного кода в процессе
- Step** - приостановка выполнения программы после каждой команды
- Talk** - вывод информации о состоянии процесса обработки
- Compile** - компиляция командного файла

Для компиляции группы файлов (программных или форматных) нажмите shift/Spacebar. Назначьте директорию, где будет храниться выходной файл и файл с сообщениями об ошибках. Если Вы хотите, чтобы откомпили-

рованные файлы были защищены, выберите двоичный переключатель Encrypt.

Вы можете сохранить сообщения об ошибках либо в регистрационном файле, либо в файле с расширением .ERR. Для сохранения сообщений об ошибках для каждого выделенного файла в отдельном файле с тем же самым базовым именем выберите переключатель Create .ERRs. Для сохранения сообщений об ошибках для всех выделенных .PRG и .Fmt файлов введите имя регистрационного файла в текстовый блок под переключателем Log To... . Для добавления к существующему регистрационному файлу сообщений об ошибках:

введите имя регистрационного файла;  
установите двоичный переключатель Append.

#### 7.4.5. Меню WINDOW

Опции меню Window используются для управления окнами.

**Hide** - скрытие окна. Скрытые окна остаются активными и выводятся в нижнюю часть меню Window

**Move** - перемещение окна в новое положение

**Size** - изменение размера выделенного окна

**Zoom** - увеличение размера текущего окна до размеров всего окна

---

**Cycle** - изменение порядка следования открытых окон.

**Color** - установщик цветов

**Command** - вывод на экран окна Команд

**Debug** - вывод окна Отладки. В окне Отладки

Вы можете вводить переменные и выражения (до 16 элементов отладки), значения которых хотите увидеть в окне и которые могут относиться к нескольким модулям и программам; устанавливать контрольные точки.

**Trace** - вывод окна Трассировки

**View** - вывод окна Представления. **Окно Представления** является основным диалоговым средством управления функционированием БД. Оно позволяет устанавливать переключатели FoxPro, открывать базы данных и устанавливать отношения.

**Табло Состояния** (активизируется переключателем ON/OFF в окне Представления) содержит 26 управляющих переключателей для системы FoxPro.

## 8. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

1. К.Дж. Дейт. Введение в системы баз данных: Пер. с англ. – 6-е издание.- Киев-Москва: Диалектика, 1998.-784с.:ил.
2. Г.Хансен, Дж. Хансен. Базы данных: разработка и управление : Пер. с англ. – М.:ЗАО «Издательство БИНОМ», 1999.- 704 с.:ил.
3. А.А. Попов. Программирование в среде СУБД FoxPro 2.0.Построение систем обработки данных. – М.: Радио и связь,. 1993. – 352 с.:ил.
4. С.Бемер. FoxPro 2.5. для Windows.: Пер. с нем. – К.: Торгово-издательское бюро BHV, 1994. – 416 с.: с ил.