## Лекция №1

**Название лекции: Система баз данных.**

План:

1. **Основные понятия: система баз данных, СУБД, основное назначение СУБД.**
2. **Основные компоненты системы баз данных:**
   1. **Данные;**
   2. **Аппаратное обеспечение;**
   3. **Программное обеспечение;**
   4. **Пользователи.**
3. **Уровни абстракции в СУБД.**

### Система баз данных, СУБД, основное назначение СУБД, функции СУБД.

*Система баз данных* - это компьютеризированная система хранения данных, основная цель, которой содержать информацию и предоставлять её по требованию.

*Система управления базами данных (СУБД)* - программное обеспечение, предназначенное для использования и (или) модификации этих данных одним или несколькими лицами.

*Назначение СУБД:*

* + *обеспечить пользователя инструментом, позволяющим оперировать данными в терминах, не связанных с особенностями их хранения в ЭВМ.*
  + *обеспечить секретность и разграничение прав доступа к информации;*
  + *защита целостности и непротиворечивость данных.*

(Например, контроль, что число проданных билетов не превышало числа мест в самолете;)

* + *синхронизация доступа к информации при одновременном обращении нескольких пользователей (проблема многопользовательского доступа)*.

(Например, исключение возможности продажи двух билетов на одно и тоже место в транспорте;)

* + *защита от отказов и восстановления состояния базы данных после отказа.*

### Основные компоненты системы баз данных.

*В системе БД выделяют* ***четыре*** *основных компонента:*

* *данные;*
* *аппаратное обеспечение;*
* *программное обеспечение;*
* *пользователи.*

**Данные.**

*Различают 2 типа СУБД*: **однопользовательские** и **многопользовательские**.

Основная задача **многопользовательской** системы обеспечить работу пользователю как в однопользовательской системе.

Данные в *системе БД* являются ***интегрированными*** и ***общими***.

***Интегрированные данные***подразумевают возможность представлять БД как объединение нескольких файлов данных, полностью или частично не перекрывающихся

***Общие данные***подразумевают возможность использования отдельных областей данных в БД несколькими отдельными пользователями отдельно.

Обычно данные, хранящиеся в БД, называются *постоянными* (хотя они недолго могут оставаться такими). «Постоянные» - по отношению к другим данным: *промежуточным, входным, выходным.*

*Входные данные* – это информация, передаваемая системе (обычно с терминала или рабочей станции). Такая информация может стать причиной изменения постоянных данных.

*Выходные данные* – это сообщения и результаты, выдаваемые системой (обычно на печать или отображается на экране, возможно, записывается на диски).

Ясно, что различие между видами данных нельзя назвать четкими, они определяются на интуитивном уровне.

На больших предприятиях в настоящее время все чаще используются два вида БД:

* *оперативное БД* - для поддержания повседневной работы предприятия;
* *БД, содержащая отчетную информацию* - данные для поддержания принятия решений по управлению предприятием. Эти данные периодически обновляются (раз в день, раз в неделю и т.д.), получая информацию из *оперативной БД.*

### Аппаратное обеспечение:

* накопители;
* сетевое оборудование;
* оперативная память
* процессор.

### Программное обеспечение:

* СУБД;
* утилиты;
* средства разработки приложений (программы конечного пользователя);

### Пользователи:

* + *Прикладные программисты* – пользователи, которые отвечают за написания прикладных программ (приложений), использующих БД.
  + *Конечные пользователи* – пользователи, которые работают с БД через рабочую станцию (терминал), они часто использует интерфейс, основанный на меню и различных формах, что облегчает работу.
  + *Администраторы базы данных* организуют и отвечают за работу с БД.

### 3.Уровни абстракции в СУБД

*Абстрагирование* – отбрасывание лишних элементов в моделях объектов реального мира с выделением основных частей и элементов.

*В широком смысле слова БД* – это совокупность сведений о конкретных объектах реального мира в какой–либо *предметной области*.

## Внешний уровень

Представле ние 1

Представле ние 2

Представле ние n

Концептуальный уровень

Обобщенное представление пользователей

Внутренний

(физический) уровень

Представление в памяти

### Трехуровневая система организации базы данных

Таким образом, существуют **три** уровня абстракции в архитектуре базы данных:

* + *представление (внешняя модель базы данных);*
  + *концептуальная база данных (КБД);*
  + *физическая база данных (ФБД).*

При проектировании БД процесс перехода от *реальности* к *информационной модели* происходит в несколько этапов:

1. ***Внешняя модель или уровень представления:***

На этом уровне предметная область описывается будущим пользователем БД*.*

1. ***Логический или концептуальный уровень****:*

На этом уровне *прикладными программистами* разрабатывается обобщенное описание предметной области, которое опирается на представления **пользователей**.

1. ***Внутренний либо физический уровень****:*

Описание модели (*концептуальной*) на языке некоторой СУБД.

***СУБД должна поддерживать все три уровня.***

# Лекция №2

**Название лекции: Схемы и экземпляры БД.**

План:

1. **Основные понятия: план, экземпляр БД, хранимое поле, хранимая запись.**
2. **Схемы уровня представлений.**
3. **Модели типа объект/отношение. (Понятие ER-модель):**
   1. **Особенности модели и ее недостатки;**
   2. **Общий подход и проблемы;**
   3. **Виды связей;**
   4. **Диаграммы объектов / связей.**
   5. **Пример диаграммы**

### Основные понятия: план, экземпляр БД, хранимое поле, хранимая запись.

*Схема (план)* - перечень типов объектов, относящихся к БД и связей между ними.

На стадии проектирования БД разработчики имеют дело с *схемой* БД.

На стадии эксплуатации мы имеем дело с содержащимися в базе данных *актуальными данными*.

*Данные в БД при эксплуатации часто изменяются. Планы (схемы) меняются значительно реже.*

*Экземпляр* БД – это совокупность информации, содержащейся в БД в каждый момент времени.

*Для описания схем и экземпляров используют следующие понятия:*

*Хранимое поле* – это наименьшая единица хранимых данных.

*(*БД может содержать много экземпляров одного из нескольких типов полей (ФИО, №Детали)).

*Хранимая запись* – это набор связанных хранимых полей, рассматриваемых как одно целое.

*Экземпляр* *записи* состоит из группы связанных экземпляров хранимых полей.

*Пример:*

запись о детали (номер, название, вес, цвет и т.д.).

*Хранимый файл* – это набор записей всех экземпляров хранимых записей одного типа.

### Схемы уровня представлений.

*Схемы уровня представлений* могут быть частями *концептуальной схемы*. В некоторых случаях схемы представления могут быть более “абстрактными” чем концептуальные.

*Пример:* для отдела кадров поддерживается представление, которое включает возраст каждого сотрудника. Поддерживать возраст на концептуальном уровне не целесообразно, т.к. изменять соответствующее хранимое поле необходимо ежедневно для всех сотрудников. Гораздо более целесообразно хранить дату рождения, а возраст вычислять как разность между текущей датой и датой рождения.

Но *представление и концептуальная схема совсем не обязательно должны определяться с помощью одной и той же модели данных. Но чаще всего происходит именно так.*

### Модели типа объект/отношение. (Понятие ER-модель).

Модели БД типа *объект/отношение* могут быть использованы на *всех уровнях абстракции.*

### Особенности модели. Ее преимущества и недостатки.

Особенностью модели является тесная логическая связь с предметной областью.

Данная модель относится к так называемым *семантическим моделям.*

*Преимущество*: проще описывать условия целостности.

*Недостаток*: сложно проводить формальную оптимизацию информационной модели, т. к. тесная логическая связь с предметной областью не дает разработчику отвлечься от ее особенностей.

### Общий подход и проблемы.

Для того чтобы описать представление *предметной области* необходимо задать множество объектов реального мира.

*Объект* – это сущность реального мира, он не обязательно должен быть материальным. Важным является только существенность и различимость каждого объекта от других.

*Пример:* объектами являются: сотрудник, пассажир, самолет, деталь и т.д.

***Различают сильные и слабые объекты:***

*Слабый* объект подчиняется другому объекту и не может без него существовать.

*Пример*: подчиненный сотрудник  основной сотрудник

Объект, не являющийся слабым, называется *сильным*.

**Различают *тип* объекта и *экземпляр* объекта**

*Пример*: тип – сотрудник; экземпляр – Иванов.

Объект обладает рядом свойств, которые иногда называют *атрибутами* объекта.

*Пример*: объект — сотрудник, его атрибуты: ФИО, дата рождения...

**Различают *тип* атрибута и *значение* атрибута.**

Совокупность значений атрибутов объекта является его *информационной моделью.*

Для определения *нового типа объекта* иногда используют понятие *подтипа.*

*Например*, пилот – есть подтип объекта сотрудник. Очевидно, что механизм *подтипов* является прообразом наследования из ООП.

*Атрибуты* или *множество атрибутов значения*, которых уникальным образом идентифицируют экземпляр объекта, называют к*лючом.*

Иначе говоря: *не существует двух различных объектов, у которых совпадают значения ключей*

*Пример*: объект студент, ключ – номер зачетной книжки.

### Виды связей Между объектами могут возникать *связи*.

Связь является отношением, т.е. подмножеством прямого произведения, но не обязательно это бинарное отношение (*поставщик – детали*).

Рассмотрим виды связей на примере базы данных «Хирургическое отделение больницы». Выделим четыре типа объектов: *Место\_в\_палате, Палата, Пациент, Хирург.*

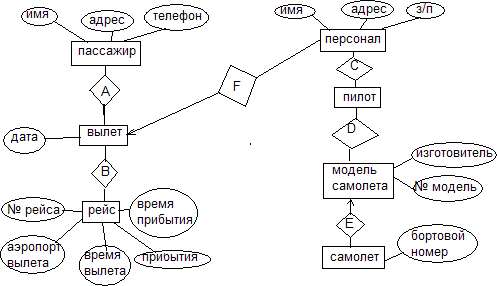
Между объектами могут возникать следующие связи:

* + один к одному 1:1 (П*ациент - Место\_в\_палате)*;
  + один к многим 1:n, или, как вариант - многие к одному n:1 (П*алата - Место\_в\_палате*);
  + многие к многим n:n (П*ациент - Хирург*).

### Диаграммы объектов / связей.

#### Диаграмма – графическое изображение структуры БД с использованием ER- модели.

Объект - Связь- Атрибут -



### Связи:

А – пассажир купил билет В – выполняется

С – есть

D – допущен к Е – типа

F – назначен на

# Лекция №3

**Название лекции: Концептуальная схема и её модели данных.**

**План:**

1. **Значение концептуальной схемы. Язык описания концептуальных схем.**
2. **Модели, которые используются для описания концептуальных схем:**
   1. **Иерархическая модель;**
   2. **Сетевая модель;**
   3. **Реляционная модель;**
   4. **СУБД, использующих другие модели.**

### Значение концептуальной схемы. Язык описания концептуальных схем.

*Выбор концептуальной схемы является центральным пунктом в разработке баз данных.*

*Концептуальные схемы* оказывают существенное влияние на способы описания схем уровня *представлений* и *физических* схем.

Для определения базы данных на концептуальном уровне СУБД предоставляет так называемый язык определения данных (DDL - data definition language). Этот язык позволяет записать концептуальную схему в терминах некоторой модели данных.

### Модели, которые используются для описания концептуальных схем:

В настоящее время хорошо разработаны и описаны три основных модели, которые используются в системах БД:

### Иерархическая модель

Моделью данных здесь является древовидный граф, т.е. связанный, ориентированный граф без циклов, у которого имеется только одна вершина, из которой дуги только выходят и ни одна дуга не входит. Эта вершина называется корнем дерева. Вершина, в которую дуги только входят, называется листом. Все остальные вершины - корни. Вершина, для некоторой вершины А, называется родительской, если из родительской вершины начинается дуга и заканчивается в А. Братья - это множество вершин имеющих одного родителя.

*Пример,*  СУБД IMS фирмы IBM .

### Сетевая модель.

Это модель ориентированных графов, вершинами которых является множество однородных объектов, а дугами – ассоциации (связи). Ясно, что модель объект/отношение является примером сетевой модели.

### Реляционная модель.

Базируется на теоретико - множественном понятии отношения. Модель разработана американским математиком Коддом (Codd), сотрудником фирмы IBM и опубликована в 1970 г. В настоящее время является наиболее полно разработанной основой теории БД. В настоящем курсе будет рассмотрена теория реляционных БД.

*Примеры реляционных* СУБД: MySQL и PostgreSQL.

**Из СУБД, использующих другие модели,**

дедуктивные СУБД;

* + экспертные СУБД;
  + расширяемые СУБД;
  + объектно-ориентированные СУБД;
  + семантические СУБД;

универсальные реляционные СУБД.

# Лекция №4

**Название лекции: Внутренняя (физическая) схема БД.**

План:

1. **Внутренняя (физическая) схема БД.**
2. **Соотношение внутреннего и внешнего языка определения данных. Язык SQL. Сильно связанные и слабо связанные языки.**

### Внутренняя (физическая) схема БД.

*Третьим уровнем архитектуры ANSI/SPARC является внутренний уровень.*

*Внутренний (физический) уровень* описывает размещение БД на устройствах внешней памяти.

Сама по себе *физическая БД* может быть представлена на нескольких уровнях абстракции от *записей* и *файлов* в языках программирования до уровня *цилиндров, дорожек, блоков, байтов и бит*, хранимых на внешних устройствах.

*Будем рассматривать внутреннее представление на уровне, эквивалентном записям и файлам языка программирования.*

*Внутреннее представление*, в таком случае, описывается *внутренней схемой*, которая определяет различные *типы хранимых записей*, *способ представления хранимых полей*, *физическую последовательность хранимых записей.*

### Соотношение внутреннего и внешнего языка определения данных.

***Внешний уровень*** – это индивидуальный уровень пользователя.

К пользователям относятся *конечные пользователи* и *прикладные программисты*. Особую роль играет *администратор БД*, который участвует в разработке всех уровней представлений.

У каждого пользователя есть свой язык общения с БД:

* + *для конечного пользователя* это, или специальный язык запросов, или язык специального назначения, возможно основанный на формах и меню.
  + для *прикладного программиста* это либо один из распространенных языков программирования, такой как С, либо специальный язык рассматриваемой СУБД.

Как для *конечного пользователя*, так и для *прикладного программиста* используемые языки включают подъязыки данных, т.е. подмножество операторов всего языка, связанное

только с объектами и операциями с БД. Т.о. подъязык встроен в базовый язык, который имеет ещё операторы, не связанные с БД (if, while и т.д.).

*СУБД может поддерживать любое количество базовых языков и любое количество подъязыков данных.*

В настоящее время существует один язык, который поддерживается практически всеми СУБД это язык *SQL (Structured Query Language).*

*Базовый язык* и включенный в него *подъязык* данных могут быть практически неразличимыми (например, SQL – как самостоятельный язык).

Если они *неразличимы*, или трудно различимы, их называют ***сильно связанными***.

Если они *легко различаются*, то говорят, что они ***слабо связанны***.

*Подъязык* данных на самом деле является комбинацией двух языков: *языка определения данных* (data definition language - DDL), который поддерживает объявление или определение объектов БД, и *языка обработки данных* (data manipulation lаnguage - DML), который поддерживает операции с такими объектами или их обработку.

#### Внешние и внутренние языки могут совпадать или не совпадать.

*Пример*: для *прикладного программиста* – это язык С + SQL. Внутренняя схема описана на С, а запросы на SQL. В этом случае внутренний и внешний языки не совпадают. Если внутренняя схема описана на SQL и запросы на SQL, то языки совпадают.

# Лекция №5

**Название лекции: Независимость данных. Администратор баз данных.**

План:

1. **Понятие независимости данных.**
2. **Уровни абстракции архитектуры СУБД: физическая и логическая независимость данных.**
3. **Администратор баз данных. Логическое проектирование БД. Список функций администратора БД при совмещении им функций системного аналитика.**

### Понятие независимости данных.

Основное, качественное отличие СУБД от предшествующих систем обработки информации является независимость данных.

Для предшествующих систем сведения об *организации данных* и *способах доступа* к ним были встроены в *логику и код приложения*, если, по каким-то причинам, необходимо было изменить структуру записей или способ доступа к файлу с информацией, то для приложений, не использующих технологию СУБД, необходимо было переписать приложения (программы).

СУБД построены таким образом, чтобы обеспечивать развитие БД, не оказывая существенного влияния на приложения.

### Уровни абстракции архитектуры СУБД.

**Три** уровня абстракции в структуре СУБД, *внешний – концептуальный – внутренний,* обеспечивают **два** *уровня независимости данных*.

**Первый уровень** *независимости данных* обеспечивается отображением концептуального уровня на внутренний (физический) уровень и называется ***физической независимостью данных.***

*Пример*: изменение размеров полей не приводит к нарушению работы ранее написанных приложений.

**Второй вид** *независимости* обеспечивается отображением *внешнего – концептуальный* и называется ***логической независимостью данных*.**

*Пример*: требуется информация об уровнях шума и загрязнении атмосферы на авиарейсах компании. Если таких сведений в базе данных нет, то возможно включение в

БД сведения о шуме и загрязнении каждого самолета, без перестройки уже разработанных приложений.

### Администратор баз данных.

Для реализации *большой БД* важную роль играют два специалиста: *администратор данных (системный аналитик)* и *администратор БД*.

Иногда функции этих специалистов совмещаются в одном лице.

***Администратор данных* (системный аналитик)** – это специалист, который определяет, какие именно данные необходимо сохранять в БД, определяет объекты, в которых заинтересовано предприятие, а также информацию, которую необходимо записывать об этих объектах.Этот процесс иногда называют *логическим проектированием БД*.

***Администратор БД***– это специалист, обеспечивающий необходимую техническую поддержку реализации *логического проектирования* БД.

Приведём список функций *администратора БД* при совмещении им функций *системного аналитика*:

1. *Создание схем представлений*.
2. *Определение концептуальной схемы с помощью концептуального языка определения данных.*
3. *Определение внутренней схемы (физическое представление) с использованием внутреннего языка определения данных* (Определение соответствия между внутренней и концептуальной схемами).
4. *Взаимодействие с пользователями*:
5. *Определение правил безопасности и целостности.*
6. *Определение процедур резервного копирования и восстановления информации после аппаратных или программных сбоев.*

*7)Управление производительностью и реагирование на изменяющиеся требования*

# Лекция №6

**Название лекции: Функции СУБД.**

План:

1. **Этапы работы СУБД.**
2. **Функции СУБД.**

### Этапы работы СУБД.

*СУБД* – программное обеспечение, которое управляет доступом к БД.

*Это происходит следующим образом:*

* 1. Пользователь водит запрос на доступ, применяя определенный подъязык данных (например SQL.)
  2. СУБД перехватывает и анализирует запрос.
  3. СУБД строит преобразование *внутренний – концептуальный* и *внешний – концептуальный*.
  4. СУБД выполняет необходимые операции над хранимой БД.

### Функции СУБД

* 1. **Определение данных.**

СУБД должна допускать определение данных. Описания должны быть произведены на некотором исходном языке и СУБД должна преобразовать эти описания в форматы соответствующих компонент БД.

### Обработка данных.

СУБД должна обрабатывать запросы пользователей на *выборку*, *изменение*, *удаление* существующих данных в БД или на добавление новых данных.

*Замечание:* Запросы языка обработки данных бывают *планируемые* и *непланируемые*.

***Планируемый запрос*** – это запрос, необходимость которого предусмотрена заранее.

***Непланируемый запрос*** – это специальный запрос, необходимость которого не была предусмотрена заранее.

Получение наибольшей производительности для *непланируемых запросов* представляет собой одну из проблем.

### Безопасность и целостность данных.

СУБД должна контролировать пользовательские запросы и пресекать попытки нарушения правил безопасности и целостности, определенных Администратором БД.

### Восстановление данных и дублирование.

СУБД, или другой связанный с ней *программный компонент*, должен осуществлять необходимый контроль над восстановлением данных и дублированием.

### СУБД должна обеспечить функцию словаря данных.

*Словарь данных* является БД, но не *пользовательской*, а ***системной***, он содержит данные о данных (так *называемые метаданные*).

### Обеспечение производительности.

Очевидно, что СУБД должна выполнять все указанные функции с максимально возможной производительностью.

# Лекция №7

## **Название лекции: Введение в теорию реляционных СУБД.**

**План:**

1. **Основные термины и понятия реляционных БД.**
2. **Основные реляционные операции**
3. **Правила Кодда**

### Основные термины и понятия реляционных БД.

*Объект* – сущность предметной области.

*Атрибут* – параметр объекта предметной области. (Свойство некоторой сущности)*.*

*Пример***:** Фамилия, Возраст – свойства объекта сотрудник.

*Домен* (атрибута) – множество допустимых значений, которые может принимать *атрибут*.

*Пример***:** значение атрибута Возраст должно принадлежат интервалу [18…80]

*Схема отношения* – конечное множество *имен атрибутов*, определяющих объект.

(Мощность схемы отношения = арности кортежей.)

(картеж=строка из таблице)

*Отношение* – конечное множество кортежей, составленных из допустимых значений атрибутов отношений.

|  |  |  |  |
| --- | --- | --- | --- |
| Фамилия | Должность | Возраст | Схема отношения  Кортеж Значение атрибута |
| Иванов  Петренко | Директор  Бухгалтер | 40  35 |
| Петров | Менеджер | 36 |
| Сидоров | Инженер | **27** |

*Ключ* (первичный ключ) – множество атрибутов, значение которых уникальным образом идентифицирует кортеж в отношении.

*Схема реляционной базы* – множество используемых в приложениях схем отношений.

*Реляционная база данных* (РБД) – множество отношений (предполагается, что отношения логически связаны между собой).

*Реляционные операции* – операции над отношениями.

*Результатом любой реляционной операции является также отношение.*

*Реляционное выражение* – выражение над отношениями, составленное из реляционных операций.

*Реляционное выражение* – тоже отношение.

*Реляционный запрос* – описание свойств (условий), которые должны удовлетворять интересующие пользователя данные.

*Эквивалентной формой описания запроса является реляционное выражение.*

Приложение взаимодействует с СУБД на некотором языке:

*Язык описания данных (****DDL****)* – язык, позволяющий описать структуру БД и создать БД с требуемой структурой.

*Язык манипулирования данными (****DM****L)* – язык, позволяющий описать действия по чтению, добавлению, обновлению и удалению данных в БД.

*Язык запросов* – часть языка **DML**, предназначенный для удобного определения сложных реляционных запросов.

*Целостность базы данных* – свойство БД, при наличии, которой БД содержит полную и непротиворечивую информацию, необходимую и достаточную для корректного функционирования приложений.

*Ограничения целостности* – набор условий, определяющие целостность базы данных.

Различают ограничения диапазонов возможных значений атрибутов и структурные ограничения (т.е. ограничения на кортежи, имеющиеся в отношениях).

Примером ограничения диапазонов является определение доменов атрибутов. Примером структурного ограничения является определение ключей.

*Транзакция* – неделимая операция по изменению содержания БД.

*Защита баз данных* – это:

* защита БД от физических и логических разрушений;
* обеспечение санкционированного доступа к данным.

### Основные реляционные операции

*Операция объединения* (**соединения**) – объединение множества кортежей двух отношений в одно общее отношение.

*Операция определена для двух отношений одинаковой арности.* Пусть R и S – отношения арности n.

**Объединение RS** = { (V1,V2 … Vn) │(V1…Vn)R (V1 … Vn)S }

*Замечание:*

повторяющиеся строки (кортежи) склеиваются в одну сроку (картеж)

*Операция разности отношений* – разностью двух отношений арности n называется отношение (арности n), в которую включены кортежи первого отношения, не принадлежащие одновременно второму отношению.

Пусть R и S – отношения арности n.

**Разность R – S** = { (V1 … Vn) | (V1 … Vn)R  (V1 … Vn)S}

*Замечание:*

Удаляем совпадения из мн. R которые есть в мн. S

*Операция декартового произведения* – двух отношений R (арности n), и S (арности m) называется отношение арности m+n, кортежи которого составлены из кортежей R и S.

Пусть R – отношение арности n, S – отношение арности m.

**Декартовое произведение** для R= {(V1 … Vn)} и S= {(W1 … Wm)} это:

RхS = {(V1…Vn, W1…Wm) │ (V1…Vn)R(W1…Wm)S}

*Операция проекции* – это унарная операция, заключающаяся в проецировании отношения на заданную схему.

Схема проецирования получается из схемы исходного отношения, удалением ряда атрибутов и (или) перестановкой атрибутов исходной схемы.

Пусть R – отношение арности n, обозначим πi1,i2…im (R) – проекцию R на атрибуты i1, i2,

…im, где 1 ≤ ij ≤ n, 1 ≤ j ≤ m

**Проекция:** πi1,i2…im (R) = {(a1 … am) |  (b1 … bn)  R | aj = bij  j = 1 … m }

Пример: проекция π2,1(R) – составлена из значений второго и первого элемента схемы отношения R.

*Операция селекции* – это унарная операция над отношением, заключающаяся в выборе из этого отношения множества кортежей, удовлетворяющих заданному условию.

Селекцией отношенияR по формуле F – это отношение: δF(R )={(V1…Vn)R| F≡1}

*Естественное соединение.*

Пусть отношение А имеет атрибуты {X1, X2…Xm, Y1, Y2…Yn}, а отношение В {Y1, Y2…Yn, Z1, Z2, …Zk}.

Атрибуты Y1, Y2 … Yn, и только они являются общими для этих отношений. Пусть атрибуты с одинаковыми именами определены на одних и тех же доменах. Для простоты множества атрибутов обозначим буквами: X, Y, Z

*Естественным соединением* А и В (A Join B) называется отношение с атрибутами X, Y, Z, состоящими из кортежей (x, y, z), таких, для которых в отношении А атрибуты

X=x Y=y, при этом в отношении В атрибуты Y=y  Z=z. (A Join B) JoinС = A Join (B Join C)

*Пример*: А = {код, имя, статус, город} – поставщики, В = {номер, вес, город} – детали Отношение A:

|  |  |  |  |
| --- | --- | --- | --- |
| **Код** | **Имя** | **Статус** | **Город** |
| 1 | Иванов | 20 | Москва |
| 2 | Петров | 10 | Казань |
| 3 | Сидоров | 30 | Казань |
| 4 | Семенов | 20 | Москва |
| 5 | Конкин | 30 | Новгород |

Отношение В:

|  |  |  |
| --- | --- | --- |
| **Номер** | **Вес** | **Город** |
| 1 | 12 | Москва |
| 2 | 17 | Казань |
| 3 | 17 | Ростов |
| 4 | 14 | Москва |
| 5 | 12 | Казань |
| 6 | 19 | Москва |

Отношение A Join B:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Код** | **Имя** | **Статус** | **Город** | **Номер** | **Вес** |
| 1 | Иванов | 20 | Москва | 1 | 12 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1 | Иванов | 20 | Москва | 4 | 14 |
| 1 | Иванов | 20 | Москва | 6 | 19 |
| 2 | Петров | 10 | Казань | 2 | 17 |
| 2 | Петров | 10 | Казань | 5 | 12 |
| 3 | Сидоров | 30 | Казань | 2 | 17 |
| 3 | Сидоров | 30 | Казань | 5 | 12 |
| 4 | Семёнов | 20 | Москва | 1 | 12 |
| 4 | Семёнов | 20 | Москва | 4 | 14 |
| 4 | Семёнов | 20 | Москва | 6 | 19 |

Другие примеры операций над отношениями: R:

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| a | b | c |
| d | a | f |
| c | b | d |

S:

|  |  |  |
| --- | --- | --- |
| **D** | **E** | **F** |
| b | g | a |
| d | a | f |

R  S:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| a | b | c |
| d | a | f |
| c | b | d |
| b | g | a |

**

Схему необходимо выбрать дополнительно R – S:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| a | b | c |
| c | b | d |

R  S

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **E** | **F** |
| a | b | c | b | g | a |
| d | a | f | b | g | a |
| c | b | d | b | g | a |
| a | b | c | d | a | f |
| d | a | f | d | a | f |
| c | b | d | d | a | f |

A, C (R):

|  |  |
| --- | --- |
| **A** | **C** |
| a | с |

|  |  |
| --- | --- |
| d | f |
| c | d |

δB=b (R )

|  |  |  |
| --- | --- | --- |
| **A** | **B** | **C** |
| a | b | c |
| c | b | d |

### Правила Кодда (требования к реляционным БД)

Большинство СУБД, распространённых на ПК, принято считать реляционными, хотя они таковыми не являются в полной мере. Кроме представления данных в виде двумерных таблиц, принадлежность к разряду реляционных систем определяется рядом признаков, сформулированных Коддом, получивших название *правил Кодда.*

*Перечислим эти правила:*

### Явное представление данных.

Все данные должны быть представлены явно и их значения должны рассчитываться косвенными алгоритмами.

*Пример*: если, явно не указан пол, то его нельзя (ошибочно) получать из фамилии.

### Гарантированный доступ к данным.

Вся информация в БД должна быть доступной для приложения.

### Полная обработка неопределенных значений.

Неопределенные значения, отличные от любого определенного значения, должны поддерживаться для всех типов данных при выполнении всех операций.

### Доступ к базе данных в терминах реляционной модели.

Описание БД должно быть выполнено на реляционном языке. Пользователь должен иметь доступ к этой информации с помощью реляционного языка. Т.е. должна быть возможность администрирования баз данных независимо от приложений.

### Полнота подмножества реляционного языка.

Реляционная схема может поддерживать несколько языков, по крайней мере, язык DDL и DML.

### Обновляемость представлений.

Все представления (виртуальные отношения) должны автоматически обновляться при модификации данных в базовых отношениях.

(Если, например, A= R  S, и А – это представление, то А должно обновляться как только меняется R или S.)

### Наличие высокоуровнего языка манипулирования данными.

Операции вставки, обновления и удаления должны применяться к отношению в целом: обеспечивая контроль над целостностью базы данных при модификации отношений. При выполнении вставки над отношением в целом легко проверить уникальность первичного ключа, ограничения на значения и пр.

### Физическая независимость данных.

Прикладные программы не должны зависеть от используемых способов хранения данных на носителях информации и методов доступа к ним.

### Логическая независимость данных.

Прикладные программы не должны зависеть от реализации любого из используемых представлений (виртуальных отношений).

### Независимость контроля целостности.

Все ограничения целостности и внешнее представления (виртуальные отношения, отчеты) должны определяться не в приложениях, а должны быть определены с помощью языка определения данных и сохранения в каталоге (словаре) базы данных.

### Дистрибутивная независимость.

Реляционная система должна быть распространяема и переносима. Создание разнородных компьютерных систем требует обеспечения доступа к базам данных в различных OS и на различных платформах.

### Согласования языковых уровней.

Если реляционная система имеет низкоуровневый язык доступа (элемент доступа – запись) и высокоуровневый язык доступ (элемент доступа – отношения), то выполнение низкоуровневых команд должно производиться с контролем целостности, так же, как и при высокоуровневых командах.

# Лекция №8

Название лекции: **Анализ реляционности распространенных СУБД клона xBase. Отношения и реляционная алгебра.**

План:

**1. Отношения и реляционная алгебра. Переменная отношения. Свойства отношений. Алгоритм нормализации отношения до 1 НФ**.

### Отношения и реляционная алгебра. Переменная отношения. Свойства отношений. Алгоритм нормализации отношения до 1 НФ.

**Отношения и реляционная алгебра.**

*Определив БД* как *множество отношений* и задав на этом множестве *операции*, мы получили алгебру, т.е. систему, позволяющую производить вычисления на множестве отношений.

Т.к. *операндами* и *результатом* каждой операции является *отношение*, то полученная система является замкнутой, поэтому мы приходим к понятиям **«переменная»** и

**«значение переменной»**

**Переменной отношение** – это абстрактное понятие, под которым мы будем понимать произвольное отношение

*Для того чтобы задать значение отношения необходимо задать:*

* 1. Схему отношения (заголовок отношения), как множество атрибутов, точнее множество пар вида ( < имя\_ атрибута >, < имя\_ домена >).
  2. Множество кортежей (тело отношения).

*Каждый кортеж* – упорядоченное множество.

С каждым значением в кортеже связано имя атрибута, т.е. кортеж можно понимать как множество пар (< имя \_ атрибута>,< значение \_атрибута>).

В нашем курсе мы рассматриваем **отношения, обладающие** следующими

### свойствами:

1. *В отношениях нет одинаковых кортежей.*
2. *Кортежи неупорядочены.*
3. *Атрибуты неупорядочены (т.е. атрибуты именованы).*
4. *Все значения атрибутов – атомарные (скалярные).*

*Пояснения:*

*Aтрибут является атомарным,* если его невозможно разбить на более простые части, которые соответствуют каким-то параметром объекта из предметной области.

Отношения, у которых все атрибуты *атомарные,* называются *нормализованными* (находящимися в 1НФ).

***Реляционная теория БД рассматривает только нормализованные отношения.***

### Алгоритм нормализации отношения до 1 НФ.

*Пример нормализации отношения* Отношение Сотрудники.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Фамилия | Должность | Оклад | Дети |
| 1. | Иванов | Директор | 100 | Оля 1990  Маша 1992 |
| 2. | Петров | Гл. инженер | 100 | Сережа 1989  Катя 1991  Коля 1995 |

Нормализованная база.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № | Фамилия | Должность | Оклад | Имя | Год рождения |
| 1 | Иванов | Директор | 100 | Оля | 1990 |
| 1. | Иванов | Директор | 100 | Маша | 1992 |
| 2. | Петров | Гл. инженер | 100 | Сережа | 1989 |
| 2. | Петров | Гл. инженер | 100 | Катя | 1991 |

# Лекция №9

Название лекции: **Виды отношений, используемых в реляционных системах. Целостность БД и используемые ключи.**

**План:**

1. **Виды отношений, используемых в реляционных системах.**
2. **Целостность БД и используемые ключи:**
   1. **Понятие целостности;**
   2. **Потенциальные ключи;**
   3. **Внешние ключи;**
   4. **Ссылочная целостность.**
3. **Ключи и Null - значения**

**3.1.Правило целостности объекта. Предпосылки введения правила. 3.1.Внешние ключи и Null – значения.**

### Виды отношений, используемых в реляционных системах.

*Именованное отношение* – это переменная типа отношение, у которой есть имя.

*Базовое отношение* – это именованное отношение, которое не является производным от других отношений.

*Производное отношение* – это отношение, определённое через другие именованные отношения (посредством реляционного выражения), и в конечном итоге через базовые отношения.

*Выражаемое отношение* – это отношение, которое можно получить из набора именованных отношений посредством некоторого реляционного выражения.

Каждое *именованное отношение*является выражаемым отношением, но необязательно, что выражаемое отношение имеет имя.

*Множество всех выражаемых отношений* – это объединение множества всех базовых отношений и множества всех производных отношений.

*Представление* - это именованное производное отношение.

*Снимки* – это именованные производные отношения.

Создание снимка похоже на выполнение запроса, за исключением того, что результат такого запроса сохраняется в базе данных под некоторым именем как отношение, доступное только для чтения.

*Результаты запроса* – неименованное производное отношение, является результатом вычисления некоторого реляционного выражения.

БД не обеспечивает постоянное хранение результатов запроса. Для этого результат запроса необходимо присвоить некоторому именованному отношению.

*Промежуточным результатом* называется неименованное производное отношение, являющееся результатом некоторого реляционного выражения.

*Хранимое отношение* – это отношение, которое поддерживается во внешней памяти.

### Реляционная модель и исчисление предикатов

Результатом реляционного выражения является отношение, т.е. множество, которое может быть задано с помощью предиката **R = {{a, b, c,} | P}**

Можно показать, что *реляционному выражению* можно поставить в соответствие некоторый *предикат*, который определит некоторое отношение, совпадающее с результатом вычисления данного реляционного выражения.

Ясно, что *исчисление предикатов* необходимо производить на *множестве* некоторых *переменных.*

*Первоначально реляционное исчисление было основано на переменных кортежа.*

*Переменная кортежа* – это переменная, которая изменяется на некотором отношении, т.е. это переменная, допустимые значения которой – это кортежи данного отношения.

Если переменная кортежа Т изменяется в пределах отношения R, то в любое данное время значения переменой Т это некоторый кортеж t из отношения R.

*Реляционное исчисление*, основанное на переменных кортежа, называется *исчислением кортежей*.

Позже было предложено альтернативное исчисление, определённое на переменных доменов, т.е. переменных, которые изменяются не на отношениях, а на доменах.

Это исчисление называется *исчислением доменов*.

Реляци*онная алгебра, исчисление кортежей и исчисление доменов являются эквивалентными вариантами языков реляционных СУБД.*

### Целостность БД и используемые ключи.

**Целостность БД** – свойство БД, при наличии которого БД содержит полную и непротиворечивую информацию, необходимую и достаточную для корректного функционирования приложений.

Некоторые *условия целостности* проверяются с помощью типов (например, даты).

В сложных случаях *целостность* проверяется вычислением *логических выражений*. Разработка таких выражений существенно опирается на *предметную область*.

#### Любое правило целостности является специфическим для данной БД.

Имеется **два правила целостности**, применимые к любой БД.

Эти **два** правила относятся к так называемым ***потенциальным и внешним ключам*.**

### Потенциальные ключи.

*Потенциальный ключ* – это обобщение понятия *первичного ключа*.

*Первичный ключ*- уникальным образом идентифицирует кортеж в отношении.

*Потенциальные ключи* также обладают таким **свойством**, но *первичный клю*ч в отношении должен быть выбран только *один*, (из потенциальных, которых может быть несколько).

### Определение потенциального ключа:

Пусть **R** – некоторая *переменная отношения*, тогда *потенциальный ключ* **K** для **R** это подмножество атрибутов R, всегда обладающее следующими свойствами:

* 1. *Свойство уникальности*: нет двух различных кортежей в текущем значении переменной R с одинаковыми значениями K
  2. *Свойство не избыточности*: никакое из подмножеств K не обладает свойством уникальности.

*На практике чаще всего встречается 1 потенциальный ключ, который выбирается первичным.*

*Пример*:

Дана БД элементов таблицы Менделеева: порядковый номер элемента, атомная масса, название элемента и т.д. – все это потенциальные ключи.

*Потенциальный ключ* – множество атрибутов-составной ключ.

*Потенциальный ключ из 1-го атрибута* – это простой ключ.

*Потенциальный ключ из нескольких атрибутов* – это составной ключ.

Если потенциальных ключей несколько, то один должен быть выбран в качестве

*первичного ключа*, а остальные будем называть *альтернативными ключами*.

### Внешние ключи.

*Определение:*

Пусть **R2** – *базовое отношение*, тогда *внешний ключ FK* в отношении **R2** – это подмножество множества атрибутов **R2**, такое что:

1. существует базовое отношение **R1** (R1 и R2 не обязательно различны) с потенциальным ключом **СК**;
2. каждое значение **FK** в текущем значении **R2** всегда совпадает со значением **СК** некоторого кортежа в текущем значении **R1**.

*Замечания:*

1. *внешний ключ* – это множество (вводят через {}, но если внешний ключ простой, то скобки могут опускаться);
2. каждому значению *внешнего ключа* соответствует значение *потенциального ключа* **(обратно не справедливо).**

*Пример*:

цвет автомобиля в таблице Auto это указатель на значение цвета, который храниться в таблице menu.

#### В теории реляционных баз данных требуется:

1. 1Если *внешний ключ* составной, то соответствующий *потенциальный ключ* тоже составной, а если *внешний ключ* простой, то *потенциальный ключ* тоже простой.
2. Каждый атрибут, входящий в данный *внешний ключ*, должен быть определен на том же *домене*, что и соответствующий атрибут, соответствующего *потенциального ключа*
3. Для *внешнего ключа* не требуется, чтобы он был компонентой *первичного ключа* данного отношения.
4. Значение внешнего ключа является ссылкой на кортеж, содержащий соответствующее значение потенциального ключа (так называемый *ссылочный или целевой кортеж*).
5. Ссылочные ограничения представляются ссылочными диаграммами. Иногда над стрелкой пишут внешний ключ.
   * + 1. C\_COLOR AUTO  MENU
6. Отношение (R2) может быть одновременно и ссылочным, и ссылающимся R3 R2R1
7. Отношение может ссылаться само на себя
   * + 1. R1 R1

*Пример:* Список сотрудников, у которых есть атрибут – код сотрудника, которому данный сотрудник подчиняется

1. Ссылки могут образовывать цикл (ссылочный цикл) Rn  Rn-1 … R1  Rn
2. Заметим, что атрибут(ы) становится *внешним ключом*, если имеется отношение с соответствующим ему *потенциальным ключом*.

### Ссылочная целостность.

*Определение:*

БД обладает *ссылочной целостностью*, если она не содержит *значений внешних ключей*, которым не соответствует значение *первичного ключа*.

*Удаление в ссылающемся отношении никогда не нарушает ссылочную целостность.*

Проверить *ссылочную целостность* в момент добавления или коррекции ссылающегося отношения не является сложным.

Вся сложность лишь в предоставлении возможности добавить кортеж в ссылочное отношение из среды работы с ссылающимся отношением.

*Добавление в ссылочное отношение никогда не нарушает ссылочную целостность.*

*Т.е. проблемы проверки ссылочной целостности возникают лишь при удалении и корректировки ссылочного отношения*.

Если при выполнении этих операций нарушается ссылочная целостность, то используют две основные стратегии:

1. операции, нарушающие целостность, отвергнуть;
2. изменить ссылающееся отношение, чтобы целостность не нарушили. При этом возможно потребуется откорректировать несколько отношений. Этот процесс называется *каскадированием.*

*Особая сложность возникает при наличии ссылочных циклов.*

*Возможно, проверку ссылочной целостности следует отложить до момента фиксации транзакции.*

**3.1. Правило целостности объекта. Первичные ключи и Null значения. Предпосылки введения правила.**

Вместе с понятием *первичного ключа* модель включает правило *целостности объекта,* которое заключается в следующем:

*Ни один элемент первичного ключа базового отношения не может быть Null – значением*.

Предпосылки введения такого правила следующие:

1. кортежи базового отношения соответствует объектам реального мира;
2. объекты реального мира различны (т.е. некоторым образом опознаваемы);
3. значит, должны быть различимы представления реального мира, т.е. кортежи базового отношения.

Если *первичный ключ* или его часть имеет **Null** – значение  *идентичность объекту теряется.*

***В реляционной базе данных мы никогда не записываем информацию о чём – то, чего мы не можем идентифицировать.***

*Проблема использования Null – значений до конца ещё не решена. Введение и поддержка этого правила имеет несколько противоречий.*

*Альтернативной стратегией* Null – значениям является использование значений *по – умолчанию.*

### 1.2.Внешние ключи и Null – значения.

Не вызывает особых сомнений, что *внешние ключи* могут принимать **Null** – значения.

*Пример*: если рассмотреть само ссылающееся отношения <сотрудники>, в котором есть атрибут <код\_начальника>, то у президента компании это атрибут содержит Null – значение.

Т.е. понятие *внешнего ключа* должно быть дополнено возможностью принимать значение **Null**.

Следовательно *определение внешнего ключа* должно быть расширено:

Пусть **R2** – *базовое отношение*, тогда *внешний ключ* **FK** в отношении **R2** – это подмножество *множества атрибутов* **R2**, такое что:

1. Существует *базовое отношение* **R1** (R1 и R2 не обязательно различны) с *потенциальным ключом* **СК**.
2. Всегда каждое значение **FK** в текущем значении **R2** или является **Null** – значением, или совпадает со значением **СК** некоторого кортежа в текущем значении **R1**

Возможность включения или не включения **Null** – значений в *первичные и внешние ключи* регулируются соответствующими опциями операторов языка определения БД.

*Замечание:*

Допустимость принятия **Null** – значения для *внешнего ключа* решает проблему *ссылочной целостности*.

*Пример*:

При удалении некоторого цвета из справочника цветов достаточно у автомобилей (база AUTO), у которых был удаляемый цвет, установить цвет в Null.

# Лекция №10

**Название лекции: Проектирование БД.**

План:

1. Функциональные зависимости. Основные зависимости:
   1. Функциональные зависимости первого типа;
   2. Функциональные зависимости второго типа.

## Функциональные зависимости.

*Определение функциональных зависимостей даётся в двух видах:*

* 1. *Для значения элементов отношений в некотором времени.*
  2. *Для всевозможных значений элементов отношения.*

Определение **ФЗ первого типа**:

Пусть **R — отношение**. **X** и **Y** некоторые *подмножество* множества его **атрибутов**. Тогда **Y** *функционально зависимо* от **Х**, тогда и только тогда, когда *каждому* элементу множества **Х** соответствует один и только один элемент множества **Y**.

*Функциональную зависимость записывают так*: **XY**.

*Пример*:

Рассмотрим отношение **R1** с множеством атрибутов A = {Поставщик, Город, Товар, Количество}

|  |  |  |  |
| --- | --- | --- | --- |
| Поставщик | Город | Товар | Кол-во |
| 1 | Москва | 1 | 100 |
| 1 | Москва | 2 | 100 |
| 2 | Саратов | 1 | 200 |
| 2 | Саратов | 2 | 200 |
| 4 | Ростов | 1 | 400 |
| 4 | Ростов | 2 | 400 |

*Выпишем ФЗ, которые существуют в R1 (при этих значениях):*

поставщик→город

(поставщик, товар) → количество (поставщик, товар) → город

(поставщик, товар) → (город, количество) (поставщик, товар) → поставщик

(поставщик, товар) → (поставщик, товар, город, количество) (поставщик) → количество

(количество) → поставщик

В **ФЗ** *левая часть* называется ***детерминантом***, а *правая* – ***зависимой частью***.

Интерес представляют **ФЗ второго типа** определение, которых справедливо не только для некоторого *значения переменной отношения*, но остается справедливым и для **любого** *значения переменной отношения*.

*Определения* **ФЗ второго типа**:

Пусть **R** – является *переменной отношения*, а **X** и **Y** являются *произвольными подмножествами* множества атрибутов отношения **R**. Говорят, что **Y** *функционально*

*зависит* от **X** тогда и только тогда, когда для **любого** *допустимого значения* отношения **R** каждому значению атрибутов, имена которых находятся во множестве **X**, соответствует одно и только одно значение атрибутов, имена которых находятся во множестве **Y**.

*Пример*:

**ФЗ «поставщик  город»** это означает, что в *предметной области* есть ***ограничение***

*«поставщики не имеют филиалов».*

.

# Лекция №11

**Название лекции: Зависимости и Правила Армстронга.**

План:

1. ФЗ и целостность БД.
2. Тривиальные и нетривиальные зависимости.
3. Замыкание множества зависимостей.
4. Правила АРМСТРОНГА.

## ФЗ и целостность БД.

**ФЗ** фактически является *условиями ограничения целостности* т.е. их необходимо

*проверять* при *обновлении данных* в СУБД.

Пусть множество таких **ФЗ** обозначается **S**, тогда если таких зависимостей *много*, то их сложно *проверить* т.е. возникает задача *получить* такое множество **Т**, которое было бы *гораздо меньшего размера*, чем исходное множество **S**, причём каждая **ФЗ** из **S** могла бы быть *заменена условиями* из **Т**.

*Поиск такого множества* ***ФЗ*** *представляет большой практический интерес.*

## Тривиальные и нетривиальные зависимости.

**ФЗ тривиальна** *тогда и только тогда*, когда *правая* часть **ФЗ** является

*подмножеством левой* части.

*Исключения тривиальных зависимостей сокращает множество* ***ФЗ***.

## Замыкание множества зависимостей.

Из некоторых **ФЗ** могут быть получены другие **ФЗ**.

## *Пример*: (A, B)(C, D)  (A, B)C , (A, B)D.

*Пусть дано некоторое множество* ***ФЗ S****. Определение*:

Множество всех **ФЗ**, которые могут быть получены из данного множества **S**, называют

**замыканием множества зависимостей** и обозначается: S+.

Чтобы получить множество **S+** используют правила вывода **ФЗ** (правила АМСТРОНГА).

Пусть **А, В, С, D** – *произвольные* подмножества некоторого множества отношения **R**. Запись **АВ** означает **АВ** (для краткости).

## Правила Армстронга.

* + 1. **Рефлексивность** ВА => А→В.
    2. **Дополнение** (А→В) => АС→ВС.
    3. **Транзитивность** (А→В)(В→С) => (А→С).
    4. **Самоопределение** (А→А).
    5. **Декомпозиция** (А→ВС) => (А→В)(А→С).
    6. **Объединение** (А→В)(А→С) => А→ВС.
    7. **Композиция** (А→В)(С→D) => (АС→ВD)

*Доказательство правил выполняется на основании определения* ***ФЗ****.*

Правила 1...3 называются *основными правилами*: Они являются необходимыми и достаточными для получения из **множества S замыкание множества S+.**

Для удобства используют правила 4...7, которые называются *вспомогательными*.

*Замечание*: **Правила Армстронга** позволяют получить **замыкание** множества **ФЗ**, но не дают *алгоритм* как это сделать.

# Лекция №12

**Название лекции: Замыкание множества атрибутов.**

**Неприводимое множество зависимостей.**

План:

1. Определение суперключа.

2. Алгоритм построения замыкания множества атрибутов Кi.

1. Алгоритм проверки, следует ли Ф.З. X  Y из множества S, т.о. проверки принадлежности X  Y множеству S+.
2. Неприводимое множество зависимостей.
3. Потенциальные ключи и функциональные зависимости.

## Определение суперключа.

***Суперключом*** отношения **R** называется *множество атрибутов* отношения **R**, которое содержит как подмножество хотя бы один *потенциальный ключ* т.е. ключ, который обладает свойствами:

* 1. – *уникальности*
  2. – *не избыточности*.

*Суперключ обладает свойством уникальности, но является избыточным.*

***Суперключи*** – это такие *подмножества* **Кi** (KiA) множества атрибутов отношения **R**, обладающее свойством *уникальности и избыточности*.

Пусть известно некоторое множество **S ФЗ** отношения **R**. Требуется найти *потенциальные ключи* этого отношения.

Если построить множество всех ключей и исключить из этого множества **суперключи**, то мы получим множество **потенциальных ключей**.

Множество всех элементов из **A**, **ФЗ** от **Ki  A,** называется *замыканием множества* **Ki**

для отношения **R** и множества **ФЗ S** и обозначается **Ki  A**. Если для **Ki** построили **Ki+**, то выполняется **ФЗ Ki→Ki**+.

**+**

## 2. Алгоритм построения замыкания множества атрибутов Кi.

*Дано*:

Отношение **R** с множеством атрибутов **A** и множеством **ФЗ**

S={X1→Y1,X2→Y2,...,Xm→Ym}.

Кроме того, пусть дано подмножество атрибутов Ki  A.

Построим **K** +, **замыкание множества ФЗ Ki**.

**i**

атрибутов из **A**, которые **ФЗ** от **Ki** (т.е. Ki→K +).

i

Т.е. требуется построить множество всех

Алгоритм построения такого множества является *рекуррентным*. *Итак*:

**K + = K**

**i i**

Цикл\_пока истина

**j** = 0, **f** = ложь Цикл\_пока **j < m**

## j + +

Если **(X j  K + )  (  yy  Yj  y  K + )**

**i i**

K + = K +  Y

i i j

**f** = истинна Всё\_если

Всё\_цикл Если не (f) то

Выход из цикла Всё\_если

Всё\_цикл

## Алгоритм проверки, следует ли Ф.З. XY из множества S, т.е. проверки принадлежности X  Y множеству S+.

1) Для **R, S** построить **X+**

2) Если **Y**  **X+** то

иначе

все - если.

## {X  Y}  S+

**{X  Y}  S**+

1. **Неприводимое множество зависимостей**.

Пусть **S1** и **S2** два множества зависимостей отношения **R**. Если **S1+**  **S2+,** то говорят, что **S2** является *покрытием* **S1**.

*Это означает, что, если выполняются накладываемые на отношение ограничения S2, то будут выполняться и ограничения S1.*

Если **S1** покрытие **S2**, а **S2** является покрытием **S1**, т.е. **S1+ = S2+,** то **S1** и **S2** называют

*эквивалентными множествами ФЗ.*

В теории реляционных **БД** принято выбирать так называемые *неприводимые множества ФЗ.*

Множество **ФЗ** называется ***неприводимым*** тогда и только тогда, когда:

 *Правая* часть каждой **ФЗ** множества **S** содержит только *один атрибут*.

 В левой части каждой **ФЗ** множества **S** не может быть опущен ни один атрибут без изменения замыкания **S**+.

 Ни одна **ФЗ** в **S** не может быть опущена из **S** без изменения **S+.**

## 6 Потенциальные ключи и ФЗ.

Если **Х** является *потенциальным ключом* отношения **R**, то все атрибуты этого отношения **ФЗ** от **Х**.

Если отношение **R** удовлетворяет **ФЗ АB** и **А** не является *потенциальным ключом*, то **R** обладает некоторой *избыточностью*.

*Пример*:

**A={<Поставщик>, <Город>, <Товар>, <Количество>}** множество атрибутов с

*потенциальным ключом* ={<**Поставщик**>, <**Товар**>}

Есть **ФЗ** <**Поставщик**><**Город**>, то отношение обладает *избыточностью.*

# Лекция №13

**Название лекции: Нормализация отношений.**

План:

* Нормализация отношений.*

## Нормализация отношений.

Понятие *нормализованной формы* ввел **Кодд** и определил 3 нормальных формы.

*Определение:*

***Нормализация*** *–* преобразование исходного *отношения* по *определенным правилам* и получение других отношений, которые *эквивалентны* исходному отношению.

## Замечание:

1. Предполагается, что необходимо использовать отношения в 5НФ, однако это не абсолютно, т. к. возможны ситуации, когда принципами нормализации необходимо пренебречь. Важно, чтобы база находилась в 1НФ.
2. Изучаемые схемы нормализации не являются естественными. Используются более эффективные способы проектирования.

# Лекция №14

**Название лекции: Декомпозиция без потерь и ФЗ. Теорема Хеза.**

План:

 Декомпозиция без потерь и функциональные зависимости.

 Теорема Хеза.

 Предварительные (перед нормализацией) замечания о ФЗ. ФЗ как семантическое понятие.

## 1Декомпозиция без потерь и ФЗ.

*Основной механизм нормализации* – декомпозиция исходного отношения.

Ясно, что получение новых отношений не должно приводить к потере информации, т. е. соединение полученных проекций должно дать исходное отношение:

Пусть **R1** и **R2** – проекции некоторого отношения **R**.

*Поставим задачу*:

Какие условия должны выполняться, чтобы при соединении отношений **R1** и **R2**

получить исходное отношение **R**.

*Решение поставленной задачи дает теорема Хеза (Heath).*

### *2.Теорема Хеза*.

Пусть **R** отношение с атрибутами {**A, B, C**}.

Если **R** удовлетворяет зависимости **AR**, т.е. **(А  В и А  С), R = ABAC**

*эквивалентно*, т. к. **А  А** по *4 правилу Амстронга*, то **R** равно соединению его проекций

## {A, B} и {A, C}.

**Лекция №15**

**Название лекции: 1,2,3НФ.**

План:

1. Определение неключевого атрибута и взаимно независимых атрибутов.
2. Неформальное определение 3НФ. Первая, вторая и третья нормальные формы.

## 1. Определение неключевого атрибута и взаимно независимых атрибутов.

*Допущение*: Для простоты изложения предполагаем, что каждое отношение имеет только один *потенциальный ключ*, который является *первичным ключом*.

*Неключевой атрибут* – это атрибут, который не входит в *первичный ключ*

рассматриваемого *отношения*.

Два или несколько атрибутов образующих множество **А** *называются* ***взаимно независимыми***, если ни один из них не зависит *функционально* от какого-либо подмножества остальных атрибутов множества А.

*Физический смысл* ***взаимно независимости***:

Каждый атрибут из множества ***взаимно независимых*** атрибутов **А** может быть

*обновлен независимо* от остальных атрибутов множества **А**.

## 2. 1, 2 и 3НФ.

Отношение находится в **1НФ** тогда и только тогда, когда все используемые *домены*

содержат только *скалярное* значение.

### *Любое нормализованное отношение находится в 1НФ.*

Недостатки **1НФ**: *избыточность*.

*Пример*:

Отношение **R1**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код | Статус | Город | Товар | Кол-во |
| 1 | 20 | Москва | 1 | 300 |
| 1 | 20 | Москва | 2 | 200 |
| 1 | 20 | Москва | 3 | 400 |
| 1 | 20 | Москва | 4 | 200 |
| 1 | 20 | Москва | 5 | 100 |
| 1 | 20 | Москва | 6 | 100 |
| 2 | 10 | Ростов | 1 | 300 |
| 2 | 10 | Ростов | 2 | 400 |
| 3 | 10 | Ростов | 2 | 200 |
| 4 | 20 | Москва | 2 | 200 |
| 4 | 20 | Москва | 4 | 300 |
| 4 | 20 | Москва | 5 | 400 |
|  |  |  |  |  |

S = {{код, товар} {количество}; {код}  {город}; {код} {статус}; {город} 

{статус}} множество **ФЗ** отношения **R1**.

**Первичный ключ** в отношении **{код, товар**} Статус поставщика определяется его месторасположением.

Данное отношение обладает избыточностью (для каждого поставщика указан город и статус).

*Избыточность приводит к различным аномалиям обновления:*

* *аномалия – вставки* **INSERT**. Нельзя добавить информацию о поставщике, который не поставил ни одного товара.
* *аномалия – удаления* **DELETE**. Возможна, что с удалением некоторой строки таблица (удаление поставки) исчезнет информация о поставщике.
* *аномалия* **UPDATE** (переписать, обновить) – эта проблема возникает в том случае, если необходимо переместить поставщика из одного города в другой необходимо откорректировать все записи о поставках от этого поставщика.

Для решения этих проблем заменим отношение **R1** несколькими проекциями.

В одно включим *первичный ключ* и все *неключевые атрибуты*, *неприводимо зависимые* от *первичного ключа*.

В остальные проекции включим *неключевые атрибуты*, *приводимо зависимые* от *первичного ключа* и ту часть *первичного ключа*, от которой данные атрибуты *неприводимо зависят*.

*Итак, получим 2 отношения:*

**R2**: **R3**:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Код | Товар | Кол-во | Код | Статус | Город |
| 1 | 1 | 300 | 1 | 20 | Москва |
| 1 | 2 | 200 | 2 | 10 | Ростов |
| 1 | 3 | 400 | 3 | 10 | Ростов |
| 1 | 4 | 200 | 4 | 20 | Москва |
| 1 | 5 | 100 | *5* | *30* | *Новгород* |
| 1 | 6 | 100 |  |  |  |
| 2 | 1 | 300 |  |  |  |
| 2 | 2 | 400 |  | | |
| 3 | 2 | 200 |
| 4 | 2 | 200 |
| 4 | 4 | 300 |
| 4 | 5 | 400 |
|  |  |  |

**ФЗ** для отношения **R2**: {код, товар} → {кол-во}

**ФЗ** для отношения **R3**: {код}→{город}, {код}→{статус}, {город}→{статус}

*Такие отношения позволяют преодолеть указанные противоречия:*

**INSERT**: Можно вставить поставщика из Новгорода, который не поставлял товар;

**DELETE**: Можно удалить товар 2 от поставщика 3, а сведения о поставщике останутся;

**UPDATE**: Для того, чтобы переместить поставщика 1 из Москвы в Новгород, достаточно поменять запись в отношении R3.

*Физический смысл* противоречия в отношении **R1** в том, что это отношение описывает не один объект (поставку товара) а два: *поставку и поставщика*.

*Определение* **2НФ** (при условии *единственности* потенциального ключа).

Отношение находится **2НФ** тогда и только тогда, когда оно находится в **1НФ** и каждый его *неключевой атрибут неприводимо* зависим от *первичного ключа*.

Проблемы, возникающие в **R3**: *транзитивная зависимость*.

**INSERT**: Нельзя включить город с некоторым статусом, из которого нет ни одного поставщика.

**DELETE**: удалив поставщика 5, удалим информацию о том, что Новгороду был установлен статус 30.

**UPDATE**: информация о статусе повторяется, т.о. изменив статус Москвы с 20 на 30 необходимо откорректировать несколько записей.

*Физический смысл* противоречия тот же: информация о двух объектах *предметной области* (город и поставщик) находится в одном отношении.

Формальным признаком проблем в R3 является наличие *транзитивной ФЗ*.

Для решения проблемы найдем от **R3** *проекции*, в которые включим *первичный ключ* и

*атрибут*, через который осуществляется *транзитивная зависимость*

.

А во второе отношении, этот же атрибут и атрибут, *транзитивно зависящий* от

*первичного ключа* исходного отношения.

*Получим отношение*:

**R5**: **R6**:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код | Город |  | Город | Статус |
| 1 | Москва | Москва | 20 |
| 2 | Ростов | Ростов | 10 |
| 3 | Ростов | Новгород | 30 |
| 4 | Москва | *Казань* | 40 |
| 5 | Новгород |  | | |

**ФЗ** для отношения **R5**: {код} →{город}

**ФЗ** для отношения **R6**: {город}→ {статус}

*Каждое отношение описывает только одну сущность*

*.*

Отношение находится **3НФ** тогда и только тогда, когда оно находится в **2НФ** и каждый его *неключевой* атрибут *не транзитивно* зависим от *первичного ключа*.

# Лекция №16

**Название лекции: Декомпозиции с независимыми проекциями.**

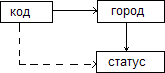
План:

1. Условия декомпозиции с независимыми проекциями.

##  Условия декомпозиции с независимыми проекциями.

Рассмотрим отношение **R** с **ФЗ** код→город и город→статус, и следовательно

*транзитивной зависимостью* код→статус.



|  |  |  |
| --- | --- | --- |
| Код | Статус | Город |
| 1 | 20 | Москва |
| 2 | 10 | Ростов |
| 3 | 10 | Ростов |
| 4 | 20 | Москва |
| 5 | 30 | Новгород |

При *нормализации* данное отношение может быть подвергнуто **декомпозиции**

различными способами.

А:

R1A R2A

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код | Город |  | Город | Статус |
| 1 | Москва | Москва | 20 |
| 2 | Ростов | Ростов | ~~10~~ |
| 3 | Ростов | Ростов | 10 |
| 4 | Москва | Москва | ~~20~~ |
| 5 | Новгород | Новгоро  д | 30 |

B:

R1B R2B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Код | Город |  | Код | Статус |
| 1 | Москва | 1 | 20 |
| 2 | Ростов | 2 | 10 |
| 3 | Ростов | 3 | 10 |
| 4 | Москва | 4 | 20 |
| 5 | Новгород | 5 | 30 |

*Определение***:**

Дано отношение **R**. Проекции **R1** и **R2** этого отношения являются *независимыми* тогда и только тогда, когда:

* 1. Каждая **ФЗ** в отношении **R** является *логическим следствием* **ФЗ** в проекциях **R1** и

**R2**.

* 1. *Общие* атрибуты проекций **R1** и **R2** образуют *потенциальный ключ*, по крайней мере, для одной из них.

В **декомпозиции А**: проекции **R1A** и **R2A** – являются *независимыми*, т.к.:

 **ФЗ** код→город и город→статус являются *естественными*, а **ФЗ** код→статус является их *логическим следствием*.

 *Общий атрибут* (город) является *первичным ключом* в одной из проекций (в R2A).

**Декомпозиция В**: проекции **R1B** и **R2B** не являются *независимыми*.

# Лекция №17

**Название лекции: Нормальная форма Бойса – Кодда (НФБК).**

План:

1. Определение НФБК.
2. Примеры НФБК.
3. Преимущества НФБК.

## Определение НФБК.

Рассмотрим более *общий* случай отношения.

*Пусть*:

 Отношение имеет два (или более) *потенциальных ключа*.

 Два *потенциальных ключа* является **сложными**.

 *Потенциальные ключи* **перекрываются**

*Замечание*: отношения, у которых имеются условия 1, 2, 3, *встречаются редко*, если у отношения нет условий 1, 2, 3, то **3НФ** *совпадает* с **НФБК**.

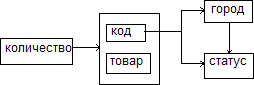
*Отношение находится в НФБК* тогда и только тогда, когда ***детерминанты*** являются

### *потенциальными ключами*.

На диаграммах **ФЗ** стрелки будут *начинаться* только с ***потенциальных ключей***.

*Никакие другие стрелки не допускаются.*

1. **Примеры НФБК.** Рассмотрим отношение **R1** Схема **ФЗ** имеет вид:



***Детерминанты***:{код}, {город}, {код, товар}, только {код, товар}→ *потенциальный ключ*.

Таким образом отношение **R1** не находится в **НФБК**.

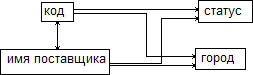
*Рассмотрим отношение*: {**код, имя-поставщика, статус, город**}

.

Допустим, что **код** и **имя-поставщика** являются *потенциальным ключом* (т.е. поставщик имеет уникальный код и уникальное имя).

Кроме того, пусть **ФЗ** город→статус не выполняется.

*Диаграмма* **ФЗ** имеет вид:



Все ***детерминанты*** являются ***потенциальными ключами***. Отношение находится в ***НФБК***.

## Преимущества НФБК.

 Позволяет избавится от некоторых проблем, присущих **ЗНФ**.

 Определение **НФБК** *концептуально* проще **ЗНФ** (нет: 1НФ, 2НФ, *первичного ключа*, *транзитивной зависимости*).

# Лекция №18

**Название лекции: Нормальные формы более высокого порядка. Многозначные зависимости.**

План:

1. Нормальные формы более высокого порядка. Многозначные зависимости.
2. Теорема Фейгина. Четверная нормальная форма.
3. Зависимости соединений и пятая нормальная форма.

##  Нормальные формы более высокого порядка. Многозначные зависимости и 4НФ.

Рассмотрим *ненормализованное* отношение:

|  |  |  |
| --- | --- | --- |
| Курс | Преподаватель | Учебники |
| Физика | Иванов Петров | Основы механики Оптика |
| Математика | Иванов | Основы механики  Векторный анализ Тригонометрия |

Из отношения видно, что никаких **ФЗ** в данном отношении нет. Проведем нормализацию до **1НФ**, получим:

R:

|  |  |  |
| --- | --- | --- |
| Курс | Преподаватель | Учебники |
| Физика Физика Физика Физика Математика Математика Математика | Иванов Иванов Петров Петров Иванов Иванов Иванов | Основы механики Оптика  Основы механики Оптика  Основы механики Векторный анализ Тригонометрия |

Очевидно, что отношение обладает *большой избыточностью*, т.е *аномалиями обновления*.

Проблема может быть решена, если отношение **R** разбить на два отношения: R1: R2:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Курс | Преподаватель |  | Курс | Учебники |
| Физика | Иванов | Физика | Основы механики |
| Физика | Петров | Физика | Оптика |
| Математика | Иванов | Математика | Основы механики |
|  | | Математика | Векторный анализ |
| Математика | Тригонометрия |

Каждый из которых является полностью *ключевым*, т.е. находится в **НФБК**.

Отношение **R** может быть получено соединением отношений **R1** и **R2**, т.е. *композиция*

из **R1** и **R2** *выполняется без потери информации*.

## Многозначные зависимости.

Пусть **А**, **В** и **С** являются произвольными подмножествами множества атрибутов отношения **R**.

Говорить, что **В *многозначно зависит*** от **А** (**А–››В**), тогда и только тогда, когда множество атрибутов **В** *зависит* от множества атрибутов **А** и *не зависит* от множества атрибута **С**.

Этот факт имеет место всегда, т.е. **многозначные зависимости** существуют парами, что регистрируется как:

**А–››В|С**, *для примера КУРС–››ПРЕПОДАВАТЕЛЬ|УЧЕБНИК.*

**Заметим**, что **ФЗ** является *частным случаем многозначной зависимости* (**МЗ**).

*Определение*:

*Отношение R находится в* ***4НФ***, если оно находится в **НФБК** и все ***МЗ*** являются фактически **ФЗ** от *потенциальных ключей*.

##  Теорема Фейгина.

Пусть **А, В** и **С** являются множествами атрибутов отношения **R {А, В, С}.**

Отношение **R** будет равно соединению его проекций **{A, B}** и **{A, C}** тогда и только тогда, когда для отношения **R** выполняется **МЗ А–››В|С**.

*Эта теорема является обобщенной теоремой ХЕЗА.*

*Фейгин показал, что* ***4НФ*** *всегда может быть получена без потери информации.*

##  Зависимости соединений и 5НФ.

Рассмотрим пример:

R

|  |  |  |
| --- | --- | --- |
| Поставщик | Детали | Проекты |
| Иванов | 1 | 2 |
| Иванов | 2 | 1 |
| Петров | 1 | 1 |
| Иванов | 1 | 1 |

Пусть данное отношение является полностью *ключевым*, не содержит **нетривиальных и МЗ**, т.е. находится в **4НФ**.

*Рассмотрим три проекции этого отношения:*

R1

|  |  |
| --- | --- |
| Поставщик | Детали |
| Иванов | 1 |
| Иванов | 2 |
| Петров | 1 |

R2

|  |  |
| --- | --- |
| Детали | Проекты |
| 1 | 2 |
| 2 | 1 |
| 1 | 1 |

R3

|  |  |
| --- | --- |
| Проекты | Поставщик |
| 2 | Иванов |
| 1 | Иванов |
| 1 | Петров |

Соединим R1 и R2 по атрибуту детали R4

|  |  |  |  |
| --- | --- | --- | --- |
| Поставщик | Детали | Проект | Линейный кортеж к отношению R |
| Иванов | 1 | 2 |
| Иванов |  |  |
| 1 | 1 |  |
| Иванов | 2 | 1 |
| Петров | 1 | 2 |

Петров11Соединим R4 и R3 по комбинации атрибутов (проект, поставщик) R5

|  |  |  |
| --- | --- | --- |
| Поставщик | Детали | Проект |
| Иванов | 1 | 2 |
| Иванов | 1 | 1 |
| Иванов | 2 | 1 |
| Петров | 1 | 1 |

Отношение R5=R

Итак, получили *свойства* отношения **R**:

Отношение **R** нельзя разбить ни на какие *два отношения*, которые после соединения будут опять отношения **R**, но если разбить **R** на *три отношения*, то после их соединения получим исходное отношение **R**.

Такое отношение *называется* ***3-х декомпозируемым отношением***.

*Аналогично можно определить n-декомпозируемые отношения.*

Для *n-декомпозируемого* отношения возможна **декомпозиция без потери** только на *n проекций*, а на меньшее число проекций **декомпозиция невозможна**.

Пусть **R** – отношение **A, B, … Z** – произвольные подмножества множества атрибутов отношения **R**.

*Отношение* ***R*** *удовлетворяет зависимости соединения* (записывают \*(A, B, … Z)) тогда и только тогда, когда оно **равносильно** соединению *своих проекций* с

*подмножества атрибутами* **A, B, …Z**.

Процесс определения находится отношение в **4НФ** или **5НФ** до конца **не определен**, но из практики такие отношения **очень редкие**.

**5НФ** является последней в том смысле, что оно *свободно от аномалий*, которые можно устранить с помощью **разбиения на проекции**.

**Нет вопросах просто оставил для понимания**

*Схема нормализации:*

 Отношение в **1НФ** следует разбить на проекции для исключения всех – **ФЗ**, которые не являются **неприводимыми**.

*Получим набор отношений в 2НФ.*

 Отношение в **2НФ** следует разбить на проекции для исключения любых

## транзитивных ФЗ.

*Получим набор отношений в 3НФ.*

 Отношение в **3НФ** следует разбить на проекции для исключения любых **оставшихся ФЗ**, в которых **детерминанты** не являются **потенциальными ключами**.

 Отношение в **НФБК** следует разбить на проекции для исключения любых

**многозначных зависимостей**, которые не являются **ФЗ**.

*Получим набор отношений в 4НФ.*

 Отношения **4НФ** следует разбить на проекции для исключения любых ***зависимостей соединения***, которые не подразумеваются **потенциальными ключами**

*Получим набор отношений в 5НФ.*

# Лекция №20

**Название лекции: Архитектура клиент/сервер. Распределенная обработка**

План:

1. Анализ структуры СУБД с точки зрения поддержки приложений БД:
   1. Понятия сервер и клиент;
   2. Приложения и типы приложений;
   3. Понятие распределенной обработки.
2. Распределенная обработка.

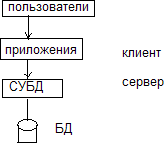
## Анализ структуры СУБД с точки зрения поддержки приложений БД. Архитектура клиент/сервер.

Ранее рассмотрена *трехуровневая* структура СУБД, в основу которой положены три уровня абстракции.

Рассмотрим структуру **СУБД** с точки зрения основной цели, для достижения которой созданы **СУБД**, а именно *поддержка* и *разработка* **приложений БД**, через которые осуществляется доступ к *информации*.

С этой точки зрения система БД состоит из **двух** частей:

***непосредственно СУБД*** (сервер или машина БД) и ***внешнего интерфейса*** (клиенты).

***Сервер*** (машина БД) – это собственно ***СУБД***.

*Таким образом, СУБД это, прежде всего,* ***программное обеспечение****.*

***Сервер*** поддерживает все основные ***функции СУБД***:

1. *определение данных*, 2)*обработку данных*, 3)*защиту и целостность* и т.д.

Таким образом в данном контексте **сервер**– это другое имя **СУБД**.

***Клиент*** – это различные *приложения*, которые выполняются **«над»** *СУБД*. *Рассмотрим подробнее* ***типы приложений****:*

* 1. ***Приложения, написанные пользователем*** *-* это *прикладные программы*, написанные на одном из *зыков программирования* типа **С** или *специализированном языке*.
  2. ***Приложения, поставляемые поставщиками***. В *целом назначение* таких средств –

*содействовать* процессу создания и выполнения других **приложений**.

## Понятие распределенной обработки.

Т.к. **СУБД** в целом может быть четко разделена на **две** части (**сервер** БД и **клиенты**), то появляется возможность работы этих двух частей на *разных машинах*, т.е. ***возможность распределенной обработки***.

**Распределенная обработка** предполагает, что *отдельные* машины можно соединить какой-нибудь *коммуникационной сетью*, чтобы *определенная задача*, обрабатывающая данные, могла быть выполнена на нескольких машинах **в сети**.

**СУБД**, которые не позволяют вести **распределенную обработку**, могут допускать *обработку* информации через *коммуникационную сеть* только в так называемом **файл - серверном режиме**.

В этом режиме *программное обеспечение* **СУБД** должно быть *загружено* в *компьютер пользователя* и при выборке информации из **файла** *удаленной* ***БД*** весь файл должен быть *перегружен* по сети с *файл-сервера* в компьютер **пользователя**.

*Замечание*: *Даже при небольшом числе пользователей такой дополнительный трафик может практически парализовать сеть.*

Термин **клиент/сервер** часто подразумевает только *распределённую обработку*, как дающую максимальную выгоду, хотя такую технологию можно реализовать и на одной **ЭВМ**.

## Распределенная обработка.

Один из простых случаев **распределенной обработки** – это *архитектура*, в которой

**сервер** *СУБД* запускается на одной машине, а **клиентские** приложения на другой.

**Термин клиент/сервер** – *фактически синоним такой схемы*. Преимущества такой схемы:

* параллельная обработка.
* **машина сервера** может быть изготовлена по специальному заказу и приспособлена для работы с **СУБД**, что обеспечит лучшую ***производительность***;
* несколько разных машин **клиентов** могут иметь доступ к одной и той же машине сервера.

*Пример*: *распределенной системой БД*

