# 1. Понятие файловой системы обработки данных (ФСОД). Организация обработки данных в ФСОД. Недостатки ФСОД.

## 1.1 Понятие файловой системы обработки данных

Такое название объясняется тем, что программы ФСОД обрабатывали файлы и получали доступ к файлам через посредство файловой системы ОС.

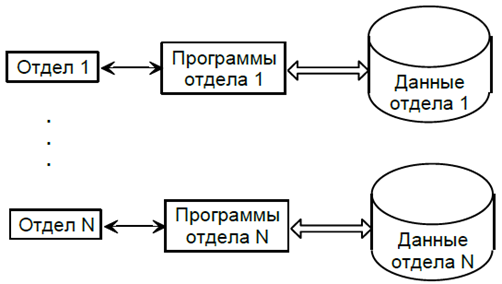
**ФСОД** состояла из отдельных автономных программ, работающих под управлением операционной системы. Каждая программа решала обычно одну задачу, т.е. поддерживала одну из функций какого-то работника.

## 1.2 Организация обработки данных в ФСОД

Разработка программ ФСОД требовала привлечения усилий многих профессиональных программистов. Поэтому на крупных предприятиях стали появляться отделы обработки данных.

ООД изучали потребности предприятия в автоматизированной обработке данных и создавали необходимые программы.

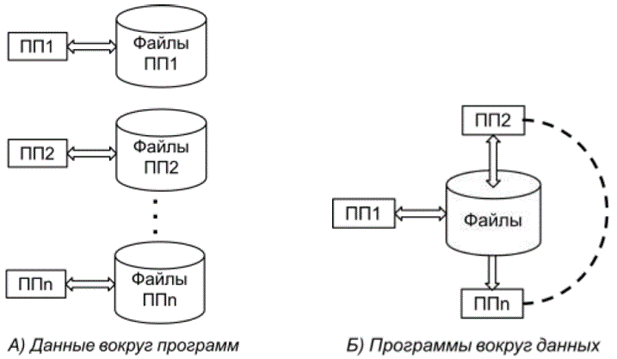
Разделение отдела обработки данных на рабочие группы естественно приводило к организации обработки данных предприятия по схеме, изображенной на Рис. 1.



**Рисунок 1 -** Организация обработки данных предприятия в ранних СОД

Здесь каждый отдел представлен как автономная (независимая) в информационном плане единица. Программы каждой рабочей группы ведут и обрабатывают свои файлы. Предполагается, что наборы данных различных отделов, не пересекаются.

Внутри отдела обычно использовался один из двух вариантов организации прямого доступа, изображенных на Рис. 2.



**Рисунок 2 -** Варианты организации обработки данных отдела предприятия

**В варианте А)** каждая программа самостоятельно вела все файлы, содержащие необходимые ей данные. Форматы записей файлов определялись автором программы. Это вполне приемлемо, если двум различным программам не приходится обрабатывать одни и те же данные. В противном случае одни и те же по смыслу данные, возможно, в различных формах, придётся сохранять в нескольких независимо обрабатываемых файлах. Идентичность (свойство данных соответствовать состоянию объектов) этих "копий" не гарантирована.   
 - файлы различных ПП обрабатываются независимо  
 - если файлы пересекаются, то происходит неуправляемое дублирование

**В варианте Б)** состав и форматы записей файлов согласовывались внутри рабочей группы. Создавалось общее для всех программ отдела поле данных. Каждая программа обрабатывала определенную часть общих данных. При такой организации удаётся избежать неуправляемого дублирования данных, но возникают другие проблемы. В частности, проблемы управления параллельным доступом программ к данным и разграничения полномочий доступа.

Заметим, что в обоих вариантах:

* каждая программа содержит в своем теле определения всех необходимых ей файлов
* каждая программа самостоятельно сканирует файлы, для того чтобы извлечь нужные записи;
* каждая программа использует методы доступа конкретной операционной системы.

## 1.3 Недостатки ФСОД

**1 – Неконтролируемая избыточность данных (превышение количества)**

Главный недостаток ФСОД с точки зрения пользователя — возможная противоречивость результатов работы различных программ. Она обусловлена тем, что каждая программа использует свой собственный набор файлов. Информационные интересы пользователей различных программ пересекаются. Поэтому различные, независимо обрабатываемые файлы должны содержать дубликаты общих данных.

ФСОД не имеет средств контроля идентичности дубликатов, а люди, вводящие в файлы значения данных, не могут работать безошибочно. Поэтому "дубликаты" нередко оказываются неидентичными. Это приводит к дезинформации пользователей и к неверным управленческим решениям.

Принципиальное решение этой проблемы — "обобществление" данных предприятия, создание единого хранилища данных для всех подразделений.

**2 – Зависимость программ от физических форматов данных**

Каждая программа ФСОД содержит определения физических форматов всех используемых файлов. Если потребуется изменить структуру записи какого-то файла, например, увеличить длину одного поля, то придется перекомпилировать и заново отладить все программы, обрабатывающие этот файл. Подчеркнем, придётся модифицировать даже те программы, которые не обрабатывают измененное поле.

Поэтому большая часть прикладных программ ФСОД постоянно находится в состоянии отладки.

**3 – Разделение и изоляция данных.**

Обычно каждый файл ФСОД содержит записи о каких-то объектах одного типа: ТОВАРах, ПРОДАЖАх, ПОСТАВЩИКах, ПОСТАВКАх и т.п. В реальной жизни объекты связаны какими-то отношениями. Эти отношения представлены в системе обработки данных какими-то связями записей.

О связях записей различных файлов знают только прикладные программы ФСОД, выполняющие их совместную обработку. Так, о связях записей файлов ТОВАР и ПРОДАЖА знает программа, которая готовит отчёт о продажах. Чтобы подготовить такой отчёт, она должна выполнить синхронную обработку записей этих файлов. О связях записей файлов ТОВАР, ПОСТАВКА и ПОСТАВЩИК знает программа подготовки отчетов о поставках, поскольку она должна выполнять синхронную обработку этих файлов.

В общем случае любая программа ФСОД, извлекающая информацию из нескольких разнотипных записей, должна выполнять синхронную обработку соответствующих файлов. Программист, автор программы, должен написать и отладить код синхронной обработки. Синхронная обработка файлов — типовое действие. Конкретные особенности реализации скрываются в параметрах синхронизации, т.е. в связях записей. В принципе можно реализовать алгоритм синхронной обработки как некую стандартную функцию.

Эта функция должна получать откуда-то сведения о структурах записей обрабатываемых файлов. ФСОД не имеет службы, которая могла бы предоставить эти сведения.

**4 – Невозможность обработки произвольных запросов**

Каждая программа ФСОД создается для поддержки какой-то одной функции или нескольких связанных функций. Многие программы ФСОД создаются для получения справочной информации определённого типа. Обобщая, можно сказать, что все программы ФСОД обрабатывают предопределенные запросы к данным.

Накопленные в файлах ФСОД данные используются в процессе оперативного управления предприятием. Информационные потребности управляющего зависят от текущей ситуации. Поэтому система должна уметь принимать и обрабатывать произвольные запросы на выборку данных. В рамках ФСОД для каждого запроса приходится писать специальную программу. К моменту её окончательной отладки управляющего уже не интересуют данные, которые она может извлечь. Изменилась ситуация.

## 1.4 Причины недостатков ФСОД

* Данные, с которыми работает подразделение, рассматриваются как его внутренний ресурс – Считается, что каждое подразделение самостоятельно накапливает, хранит и обрабатывает всё необходимое. Такой подход приводит к избыточности данных. ФСОД не имеет никаких средств контроля согласованности данных различных подразделений.
* Программы ФСОД получают доступ к данным, используя методы доступа конкретной операционной системы (ОС) - Определения файлов хранятся в телах программ и неизбежно дублируются в различных программах. Кроме того, программы содержат ссылки на конкретные устройства ввода/вывода. Поэтому любое изменение структур файлов, обновление ОС или внешних устройств приводит к необходимости изменения и перекомпиляции программ.
* Программы ФСОД реализуются и исполняются как автономные функциональные единицы, выполняющие только предопределённую обработку данных

# 2. Система баз данных: компоненты, категории пользователей, компоненты приложений.

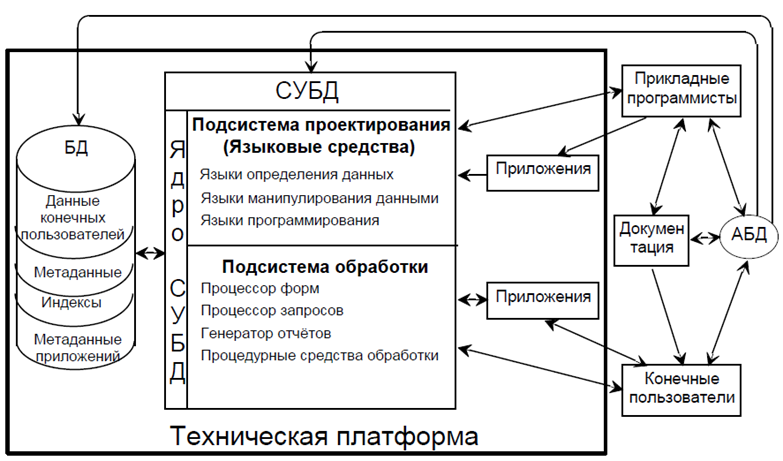
## 2.1 Понятие системы баз данных

База данных и СУБД образуют ядро системы баз данных (СБД). Помимо БД и СУБД в систему включаются: аппаратура, приложения, документация и пользователи.

**Система баз данных** — это человеко-машинная система, предназначенная для поддержания динамической модели ПО и коллективного многоцелевого использования данных.

СБД является центральным хранилищем информации предприятия и, в то же время, инструментальным средством поиска и анализа информации для различных пользователей, решающих свои задачи. Она предоставляет доступ к данным одновременно и независимо многим пользователям, создавая для каждого из них иллюзию индивидуальной работы.

Схематически компоненты СБД представлены на Рис. 3



**Рисунок 3 –** Компоненты СБД

*Документация* содержит различные инструкции и правила, которые должны учитываться при проектировании и эксплуатации базы данных. Содержание — в книжке. Документация создается вместе с системой и поддерживается в актуальном состоянии администратором базы данных.

## 2.2 Пользователи СБД

Пользователи системы по их отношению к процессам обработки данных разделяются на три категории.

* **Конечные пользователи (КП)** – это работники предприятия, использующие данные для выполнения служебных обязанностей. Как правило, КП не имеет прямого доступа к БД организации. Для каждого КП создаётся прикладная программа (приложение), поддерживающая специально для него разработанный интерфейс (интерфейс конечного пользователя). Приложение обеспечивает КП доступ только к той части БД организации, которая соответствует его обязанностям. Это и есть его база данных. Приложение поддерживает только те манипуляции данными в БД, которые обусловлены служебными функциями КП.
* **Прикладные программисты** – это пользователи системы, создающие и сопровождающие программы для конечных пользователей. Они не работают с данными непосредственно. Их задачи – реализация алгоритмов обработки данных и создание удобного интерфейса конечного пользователя
* **Администратор базы данных** — это группа специалистов, проектирующих, реализующих и сопровождающих систему. АБД несёт всю ответственность за функционирование и развитие системы.

## 2.3 Компоненты приложений

**Приложением называют** входящий в состав СБД комплекс программных средств, предназначенный для поддержки служебных функций определенной группы конечных пользователей.

Приложение состоит из запросов, форм, отчетов, меню и прикладных программ

**Запрос** — это требование на выборку из БД подмножества данных, удовлетворяющих определенным условиям. Запрос записывается на языке манипулирования данными, сохраняется в БД и может быть в дальнейшем активирован пользователем или прикладной программой по мере необходимости. Используя механизм запросов, пользователи могут получать из БД нужную в данный момент информацию в форме простой таблицы

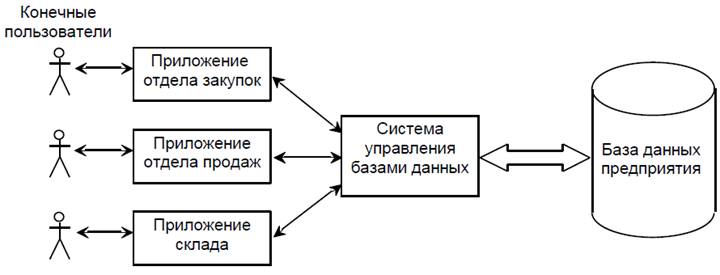
**Форма** — это способ отображения данных пользователя в виде, удобном для восприятия. Формы используются для просмотра, ввода и редактирования записей.

**Отчёт** — это результат обработки одного или нескольких запросов, отображённый в стандартной форме документа. Отображаемые данные сгруппированы в разделы и часто агрегированы (объединены в единое целое). Отчет содержит заголовки, колонтитулы и прочие атрибуты оформления документа.

**Меню** — это средства доступа к функциям приложения. С возможностями механизма меню вы знакомы достаточно хорошо.

**Прикладные программы** пишутся на входном языке СУБД. Прикладные программы дополняют стандартные возможности СУБД функциями, необходимыми конкретному конечному пользователю. Обычно они выполняют нестандартный контроль ввода данных, обрабатывают транзакции конечного пользователя и т.п.

Итак, приложение обеспечивает интерфейс конечного пользователя с СУБД. Пользователи через посредство приложений запрашивают у СУБД необходимые действия над БД. СУБД выполняет запрошенные действия и передаёт результаты пользователям через приложения. Схема взаимодействия конечных пользователей с СУБД приведена на рис. 4



**Рисунок 4 -** Схема взаимодействия конечных пользователей с БД

# 3. База данных и СУБД. Понятие БД. Структурные единицы БД. Ключи. Метаданные. Индексы. Назначение СУБД. Основные подсистемы СУБД.

## 3.1 Понятие БД

**Базой данных** называют совокупность записей, необходимых для осуществления какой-то организованной деятельности. В этих записях содержится представляющая интерес для владельца информация — сведения об объектах и отношениях объектов, попадающих в сферу деятельности.

**БД** (в широком смысле, независимо от способов накопления и сохранения данных)– это динамическая модель предметной области. Ее состояние в каждый момент времени отражает текущее состояние ПО.

***База данных (БД)*** *— это самодокументированная интегрированная совокупность записей.*

## 3.2 Структурные единицы БД



**Поле** – это элементарная именованная единица логической организации данных, которая соответствует неделимой единице информации — (атрибуту, реквизиту).

Для описания поля обязательны следующие характеристики:

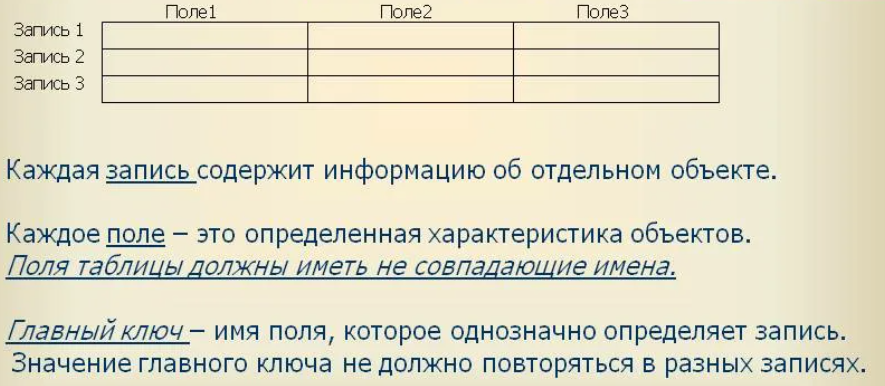
* имя
* тип (множество допустимых значений)
* длина (максимально возможное количество символов)
* точность (для числовых данных)

**Пример**: Фамилия CHAR (15), Дата\_рождения DATA (8)

**Запись —** именованная совокупность логически связанных полей. Запись понимается как структурный тип данных. Описание записи составляется из её имени и описаний полей.

**Экземпляр записи –** это отдельная реализация записи, содержащая конкретные значения ее полей (значение структурного типа данных).

**Таблица** — это совокупность экземпляров записей одного типа.



## 3.3 Ключи

В структуре записи выделяют поля (группы полей), значения которых используются для идентификации экземпляров записи.

**Первичный ключ (Primary Key, PK)** – это поле (или группа полей), значения которого принципиально не могут быть одинаковыми в различных одновременно существующих экземплярах записи.

**Первичный ключ** – это уникальный идентификатор экземпляров записи. Его значения используются для быстрого поиска экземпляров:

* Каждая запись имеет первичный ключ.
* Первичный ключ записи не может быть выбран произвольно.

В качестве **вторичного ключа** может быть выбрано любое поле, по значениям которого приходится часто искать или сортировать экземпляры записи.

Важной разновидностью вторичного ключа является внешний ключ.

**Внешний ключ (Foreign Key, FK)** – это поле (группа полей) записи R1, значения которого являются ссылками на существующие экземпляры записи R2.

## 3.4 Метаданные

**Метаданные** – это информация об объектах БД пользователя.

Метаданные, как и данные пользователей, хранятся в таблицах. Их называют системными таблицами. Они составляют часть системного каталога. В них содержатся, в частности, описания полей и типов записей БД пользователя, сведения об отношениях таблиц, индексах, хранимых процедурах, триггерах и других объектах.

## 3.5 Индексы



Индексы – это специальные служебные записи. Используются для повышения производительности приложений.

Кроме метаданных и индексов многие современные системы сохраняют в БД метаданные приложений. Это определения интерфейсных окон, отчётов и т.п.

**Индекс –** сопоставленная базовой таблице служебная структура, обеспечивающая быстрый поиск записей БД пользователя.

Индекс поддерживает прямой доступ к записям по ключу (PK, FK, SK) и последовательный просмотр записей в порядке возрастания или убывания значений ключа. Основной идеей индекса является хранение упорядоченного списка значений ключа с привязкой к каждому значению списка указателей — физических идентификаторов записей, содержащих это значение. Эти избыточные данные, используются для улучшения производительности системы и доступности БД.

Индексы бывают уникальными и неуникальными. Уникальные создаются для полей, значения которых не могут дублироваться в существующих записях таблицы. Неуникальные могут создаваться для любых других полей, если по их значениям нужно часто сортировать или фильтровать записи.

## 3.6 Понятие СУБД

**Система управления базами данных** — это комплекс программных и языковых средств, необходимых для создания баз данных, поддержания их в актуальном состоянии и организации поиска в них необходимой информации.

## 3.7 Основные подсистемы СУБД

**1 – Подсистема проектирования**

Подсистема проектирования (см. рис. 3) предназначена для создания баз данных, обеспечения интерфейсов всех категорий пользователей и проектирования приложений.

* **Язык определения данных** (ЯОД) предоставляет средства описания элементов и структур данных, экранных форм и других параметров приложений.
* **Язык манипулирования данными** (ЯМД) обеспечивает навигацию в БД и формулирование запросов к данным.
* **Язык программирования** предназначен для написания прикладных программ, обрабатывающих данные.

**2 – Подсистема обработки**

Подсистема обработки занимается обработкой компонентов приложения. Так, при открытии формы процессор форм запрашивает ее определение из раздела БД «Метаданные приложений», воспроизводит элементы формы на экране, связывает поля с элементами хранимых данных и отображает в них текущие значения данных.  
Например, при открытии формы процессор форм

– запрашивает её определение из раздела БД «Метаданные приложений»,

– воспроизводит элементы формы на экране,

–связывает поля с элементами хранимых данных и

– отображает в них текущие значения данных.

**Ядро СУБД**

**Ядро СУБД** — это постоянно загруженная часть системы. Оно обеспечивает интерфейс между БД и двумя другими компонентами СУБД.

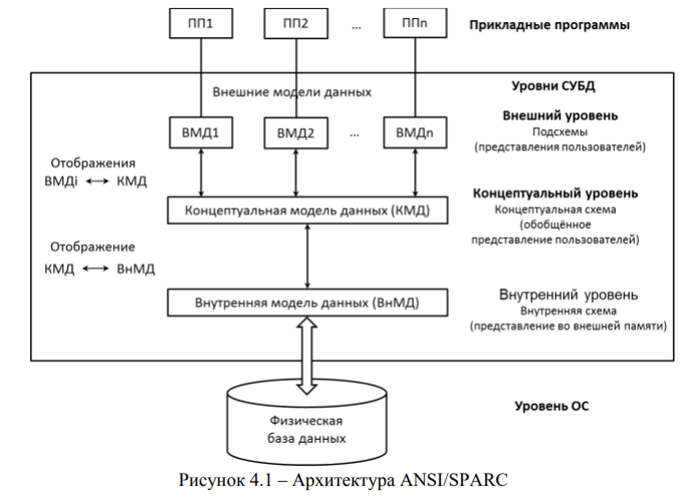
Кроме этого, ядро участвует в управлении транзакциями, санкционировании доступа к данным, резервном копировании и восстановлении БД.

## 

# 4. Архитектура ANSI/SPARC. Уровни представления данных. Отображения. Независимость от данных. Системный каталог СУБД (назначение, сохраняемые данные, владелец, пользователи).

## 4.1 Архитектура ANSI/SPARC

БД и программные средства их создания и ведения (СУБД) имеют трехуровневое строение (см. рис. 4.1). Различают концептуальный, внутренний и внешний уровни представления данных БД, которым соответствуют одноименные схемы (описания).

Ш

ANSI – American National Standard Institute (Национальный институт стандартизации США), SPARC – Standards Planning and Requirement Committee (Комитет планирования стандартов и норм).

## 4.2 Уровни представления данных

**1 – Концептуальный уровень**

Центром архитектуры является концептуальный уровень. Он соответствует логическому представлению данных ПО в обобщённом виде без каких-либо ссылок на реализацию.

Концептуальная схема состоит из определений типов концептуальных записей, их связей, ограничений целостности данных и других типов объектов.

В состав концептуальной записи входят только семантически значимые поля. Никаких служебных полей, видимых пользователю, в её структуре нет.

Концептуальный уровень представления данных можно понимать как множество экземпляров концептуальных записей. Они физически не существуют. СУБД воспроизводит их из экземпляров внутренних записей, обрабатывая запросы пользователей.

**2 – Внутренний уровень**

Внутренний уровень описывает организацию данных в среде хранения и соответствует физическому представлению данных.

Внутренняя схема состоит из определений типов внутренних (хранимых) записей, файлов внешней памяти, индексов и других компонентов уровня реализации. Эти определения выполняются на входном ЯВУ СУБД.

Кроме схемы на внутреннем уровне представлена информация:

* о распределении дискового пространства,
* о размещении файлов на физических носителях,
* о сжатии данных и методах шифрования.

Таким образом, множество экземпляров внутренних записей и есть база данных с «точки зрения» СУБД. Именно их она запрашивает у операционной системы.

**3 – Внешний уровень**

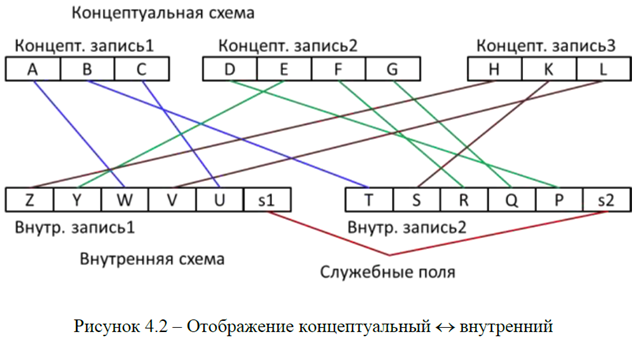
Внешний уровень поддерживает частные представления данных, необходимые конкретному пользователю. Каждая внешняя схема (подсхема) является частью (подмножеством) концептуальной схемы или выводится из неё посредством однозначных преобразований.

Внешние записи формируются из полей концептуальных записей, но они представляют только те сущности, атрибуты и связи ПО, которые представляют интерес для конкретного КП. Этот КП может даже не подозревать о том, что в БД есть и другие сведения.

Кроме внешних схем на этом уровне представлены определения форм, отчётов, запросов и другие компоненты приложений. С помощью внешних схем поддерживается интерфейс конечного пользователя и санкционированный доступ к данным и приложениям.

## 4.3 Отображения

Уровни представления данных связаны однозначными отображениями. Отображение определяет соответствие между представлениями верхнего и нижнего уровней. Отображение **концептуальный ↔ внутренний** ставит в соответствие концептуальным полям и записям хранимые. Это соответствие является взаимно однозначным. При изменении структур хранения отображение **концептуальный ↔ внутренний** изменяется так, чтобы концептуальная схема осталась неизменной. Аналогично отображение **внешний ↔ концептуальный** ставит в соответствие концептуальным полям и записям внешние.

****

## 4.4 Независимость от данных

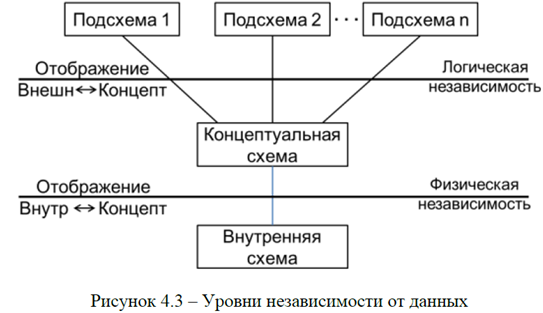
Основное назначение трехуровневой архитектуры – обеспечить независимость приложений от данных. Требование независимости от данных означает, что никакие изменения на нижних уровнях не должны влиять на верхние уровни представления данных.

Различают два типа независимости: логическую и физическую.

**Логическая независимость** есть полная защищённость существующих внешних схем от любых изменений, вносимых в концептуальную схему.

**Физическая независимость** есть полная защищённость концептуальной схемы от любых изменений, вносимых во внутреннюю схему.

Независимость достигается за счёт модификации отображений при изменении схем нижних уровней. Отображение реализуется с помощью таблиц системного каталога, хранящих сведения о соответствии полей и записей верхнего и нижнего уровней. Для модификации отображения достаточно обновить данные в этих таблицах. Реализация независимости от данных в архитектуре ANSI/SPARC изображена на рис. 4.3.



## 4.5 Системный каталог СУБД

Определения данных всех уровней сохраняются в системном каталоге СУБД. Он представляет собой внутреннюю базу данных СУБД, предметной областью которой является система баз данных. В системном каталоге сохраняются:

* определения типов записей всех уровней;
* описания отображений;
* сведения о пользователях системы и их привилегиях (правах на использование данных и приложений);
* сведения о программных и аппаратных ресурсах системы;
* и другие сведения, необходимые для управления системой.

**Системный каталóг СУБД** – это информационное ядро системы, обеспечивающее согласованную работу всех её компонентов.

У него нет владельца, владелец сама БД, создается при создании БД, извне в него вмешаться нельзя.

Состав и структуры таблиц СК определяются разработчиком СУБД. Владельцем СК является СУБД. Никакой пользователь не имеет права прямого обновления СК посредством операторов манипулирования данными

# 5. Организация доступа приложений к данным. Типовые операции обработки данных. Организация работы с внешней памятью. Обеспечение целостности данных.

## 5.1 Дисциплина доступа приложений к хранимым данным

В соответствии с архитектурной концепцией ANSI/SPARC приложения получают доступ к данным, хранящимся в ФБД только через посредство СУБД.

Приложение направляет СУБД запрос на обработку данных, сформулированный в терминах внешней модели. СУБД (средствами ОС) извлекает нужные физические записи, помещает их в свой рабочий буфер, СУБД выдаёт ОС запрос на считывание в свои буферы необходимых записей физической базы данных (ФБД).

**Шаг 5.** ОС считывает затребованные записи и помещает их в системные буферы СУБД.

**Шаг 6.** На основании схем и описаний отображений СУБД формирует в своём буфере затребованные внешние записи.

**Шаг 7.** СУБД пересылает сформированные внешние записи в рабочую область (РО) ПП.

**Шаг 8.** СУБД передаёт ПП сообщение об исполнении запроса.

## 5.2 Типовые операции обработки данных

В процессе исполнения запросов приложений СУБД выполняет четыре разновидности высокоуровневых (описанных средствами входного ЯМД) операций обработки данных.

**Извлечение записей (RETRIEVE)** – считывание в рабочий буфер совокупности записей ФБД и формирование в буфере совокупности внешних записей, затребованных приложением.

Извлеченное множество записей может обновляться приложением. СУБД выполняет следующие операции обновления.

**Добавление записей (INSERT)** – создание в рабочем буфере новых записей, содержащих введенные приложением новые значения данных.

**Изменение значений (UPDATE)** – замена существующих значений указанных полей в извлеченном множестве записей новыми значениями, введенными приложением.

**Удаление записей (DELETE)** – уничтожение в рабочем буфере подмножества извлеченных записей, указанного приложением. При этом записи могут либо физически уничтожаться, либо помечаться как удалённые, но физически сохраняться в буфере.

Все эти операции изменяют только состояние рабочего буфера СУБД. При этом никаких изменений в ФБД не происходит.

## 5.3 Организация работы с внешней памятью

Записи ФБД, считанные в рабочий буфер СУБД по запросу какого-либо приложения, не удаляются из буфера по исполнении запроса. Они могут потребоваться тому же или другому приложению. СУБД поддерживает определенную дисциплину обменов данными с внешней памятью.

СУБД, приняв запрос приложения, проверяет, есть ли в буфере данные, необходимые для его исполнения. Они могли содержаться в физических записях, извлеченных по предшествующим запросам. Если данные есть, то они не извлекаются повторно.

Если нужных данных нет в буфере, то система проверяет, достаточно ли в нём места для размещения требуемых физических записей.

Если места достаточно, то нужные записи извлекаются, и запрос приложения исполняется. В противном случае содержимое рабочего буфера выталкивается во внешнюю память и буфер частично очищается. Из него удаляются редко используемые данные. Для того чтобы иметь возможность сделать это, система накапливает статистику обращений к данным.

Такая дисциплина обменов позволяет существенно уменьшить среднее время реакции системы на запрос приложения.

## 5.4 Обеспечение целостности данных

При каждой попытке обновления состояния базы данных СУБД автоматически проверяет, удовлетворяет ли новое состояние этим ограничениям. Если ограничения нарушены, то СУБД выполняет одно из двух предопределенных действий:

* сообщает пользователю об ошибке и ждёт его реакции, или
* автоматически устраняет рассогласования данных.

**Например**, согласно правилам ведения записей в регистратуре больницы, поле Пол таблицы БОЛЬНОЙ может принимать значения ‘Муж’ или Жен’. Это правило должно быть описано предикатом:

Пол = ’Муж’ or Пол = ’Жен

При попытке ввода значения в это поле СУБД запустит процедуру вычисления значения этого предиката на введенном значении поля Пол. Если предикат принял значение ИСТИНА, то обновление принимается. В противном случае оно отвергается, а пользователь получает сообщение об ошибке

**Обновленное состояние БД фиксируется** во внешней памяти только после того, как СУБД убедится в его корректности

СУБД не в состоянии гарантировать истинность вводимых значений данных. Она может лишь гарантировать их соответствие заданным ограничениям целостности.

Для СУБД состояние БД является допустимым только тогда, когда оно удовлетворяет всем ограничениям целостности.

Обобщая, можно сказать, что вся совокупность ограничений целостности данных может быть проверена лишь по исполнении некоторой логически завершенной последовательности операций обновления.

# 6. Управление доступом к данным. Принципы ограничения доступа. Авторизация пользователей. Привилегии пользователей.

## 6.1 Принципы ограничения доступа

Любая система санкционирования доступа к данным базируется на двух основных принципах.

* Служащий имеет право доступа только к тем сведениям, которые необходимы для исполнения его служебных обязанностей.
* Служащий имеет право выполнять только те манипуляции доступными данными, которые обусловлены его служебными обязанностями.

## 6.2 Авторизация пользователей

В компьютерной системе эти принципы реализуются механизмом авторизации (подсистемой доступа). Подсистема доступа обеспечивает предоставление прав (привилегий) доступа к системе и ее объектам.

Пользователем считается зарегистрированный в системном каталоге идентификатор авторизации **(User’s ID),** который может быть присвоен лицу, группе лиц или прикладной программе.

**Привилегия** – это право пользователя выполнять определенное действие над определенным объектом БД: запускать приложение, просматривать таблицу, вставлять строки в таблицу и т.п

В учётной записи указывается:

* **идентификатор авторизации** – имя, под которым пользователь будет известен системе;
* **тип пользователя** – лицо, группа или прикладная программа;
* **описание** – ФИО, должность лица, наименование группы, имя программы, её назначение и т.п.

Учётные записи сохраняются в системном каталоге.

## 6.3 Аутентификация

С каждым идентификатором связывается пароль, выбираемый пользователем и известный только ему. Пароль сохраняется в зашифрованном виде в защищенной части системного каталога и недоступен для просмотра никакому пользователю, в том числе и владельцу. Пароль используется при подключении пользователя к системе для аутентификации, т.е., для проверки того, является ли пытающийся подключиться тем, за кого он себя выдаёт.

## 6.4 Привилегии доступа

С каждым идентификатором авторизации связывается набор привилегий. Выделяют системные и объектные привилегии.

**Системная привилегия** – это право создания и модификации объектов БД – схем, таблиц, приложений, правил и т.п. Пользователь, создавший объект, является его владельцем. Он может использовать его в любых операциях.

**Объектная привилегия** – это право использования объекта в операциях определённого типа. Например, право просмотра конкретной таблицы или её отдельных столбцов, или некоторого подмножества её строк, право вставки строк в конкретную таблицу, право запуска приложения и т.п.

Говоря обобщённо, объектная привилегия есть тройка (идентификатор авторизации, объект БД, действие).

Сведения о привилегиях сохраняются в системном каталоге в виде матрицы управления доступом.

Приняв запрос пользователя, СУБД извлекает соответствующую строку матрицы управления доступом, и проверяет, имеет ли пользователь все необходимые привилегии на вовлечённые в запрос объекты. Запрос обрабатывается, только если проверка дала положительный результат.

## 6.5 Подсхемы

**Подсхема (внешняя схема)** представляет собой набор виртуальных записей, реально не существующих в БД. Эти записи создаются в системном буфере тогда, когда пользователь запускает свое приложение. Подсхема является мощным и гибким средством ограничения доступа к данным. Она отображает для пользователя только ту часть БД, которая необходима ему для работы.

## 6.6 Сеанс

**Сеанс** – это промежуток времени между моментом подключения пользователя к системе и моментом отключения. Подключив пользователя, система готова предоставить ему все свои ресурсы в рамках его привилегий.

Момент начала, и момент окончания сеанса пользователя регистрируются в системном журнале.

# 7. Транзакции в БД: понятие транзакции, свойства транзакции.

**Транзакция** – это последовательность действий над базой данных, выполняемая по запросам одного пользователя или приложения и переводящая БД из согласованного начального состояния в согласованное конечное состояние.

Транзакция воспринимается системой как неделимая единица работы. Должны быть выполнены либо все операции транзакции, либо ни одна. Прикладной программист или пользователь, работающий в интерактивном режиме, должен указать первый и последний операторы транзакции. Транзакция должна начинаться и завершаться при согласованном (целостном) состоянии БД.

**Свойства**:

* Атомарность. Транзакция является неделимой единицей работы в БД.
* Согласованность. Транзакция начинается и завершается при согласованном состоянии БД.
* Изолированность. Система гарантирует эквивалентность результатов параллельного исполнения любого набора транзакций результатам независимого (последовательного) исполнения транзакций того же набора в любой последовательности.
* Долговечность. Если транзакция завершается фиксацией, то система гарантирует сохранение произведенных ею обновлений данных.

# 8. Виды конфликтов параллельного доступа транзакций к данным: потеря обновлений, «грязные» чтения, несогласованные изменения.

**Чтение «грязных» данных**. Эта ситуация может возникнуть, если транзакция имеет доступ к промежуточным результатам другой транзакции. Грязные чтения невозможны, если до завершения транзакции, изменяющей некоторый объект, никакая другая транзакция не сможет читать этот объект.

**Потеря обновлений**. Эта ситуация может возникнуть, если две транзакции пытаются параллельно обновлять один и тот же объект.

Избежать потери обновлений можно, если до завершения транзакции, изменяющей объект, никакая другая транзакция не сможет его изменять.

**Несогласованная обработка**. В двух предыдущих примерах речь шла о параллельном исполнении транзакций, изменяющих состояние БД. Однако проблемы возникают и в том случае, если одна из двух параллельно исполняемых транзакций обновляет состояние БД, а другая только извлекает данные, и выполняет какие-либо преобразования.

Несогласованная обработка невозможна, если до завершения транзакции, читающей некоторый объект, никакая другая транзакция не может изменять его

# 

# 9. Управление параллелизмом. Конфликты параллельного доступа. Принцип изолированности транзакций. Двухфазный протокол синхронизационных блокировок.

Из сказанного следует, что транзакции являются не только единицами работы в БД, но и единицами управления. СУБД должна обеспечить такой режим обработки транзакций, при котором результат их параллельного исполнения совпадал бы с результатом последовательного исполнения в каком-либо порядке. В этом случае транзакции будут изолированы, т.е., никакая из них не будет влиять на результаты других. Для обеспечения изолированности транзакций система должна поддерживать определённую дисциплину блокировок объектов.

**Двухфазный протокол блокировки**. Идея, лежащая в основе этого подхода, состоит в следующем. Прежде чем выполнять какую-либо операцию над объектом базы данных А, транзакция Т должна запросить блокировку (захват) А. В зависимости от вида предполагаемой операции объект может быть заблокирован в однонесогм из двух режимов: S (Shared lock) – разделяемый захват, необходимый для выполнения операции чтения; X (eXclusive lock) – монопольный захват, необходимый для выполнения операций добавления, удаления и модификации объекта. Если объект захвачен некоторой транзакцией в режиме S, то его может захватить в этом же режиме любая другая транзакция, однако захват в режиме X невозможен. Объект, захваченный в режиме X, не может быть захвачен другой транзакцией ни в каком режиме. Транзакция, запросившая блокировку объекта, заблокированного другой транзакцией в несовместимом режиме, ожидает до тех пор, пока блокировка с этого объекта не будет снята.

**Сформулируем теперь протокол доступа к данным.**

* Перед операцией извлечения объекта транзакция должна заблокировать его в S-режиме.
* Перед операцией обновления объекта транзакция должна заблокировать его в X-режиме. Если она уже заблокировала его в S-режиме, то эта блокировка должна быть заменена X-блокировкой.
* Если запрашиваемая транзакцией Т1 блокировка отвергается из-за несовместимости с блокировкой, наложенной на объект транзакцией Т2, то Т1 переходит в состояние ожидания. Она находится в этом состоянии до тех пор, пока наложенная транзакцией Т2 блокировка не будет снята.
* Все наложенные транзакцией блокировки сохраняются вплоть до завершения транзакции оператором COMMIT или ROLLBACK.

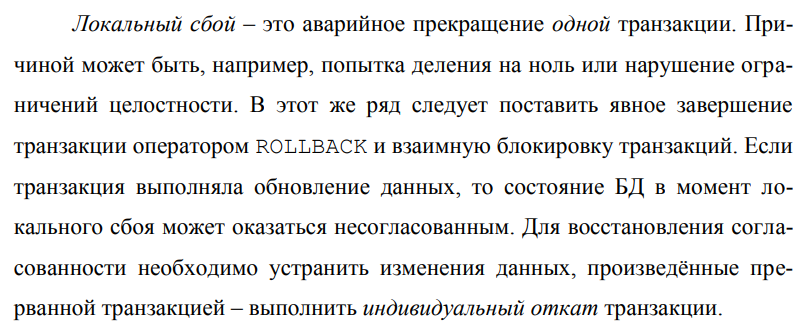
Если все транзакции соблюдают двухфазный протокол блокировки, то коллизии потерянных изменений, «грязных» чтений и несогласованной обработки невозможны.

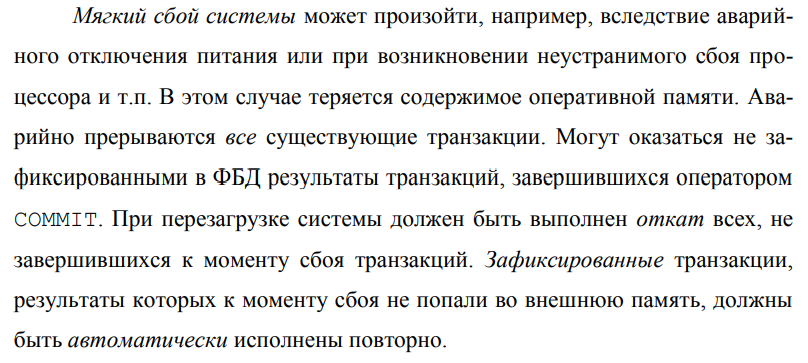
# 10. Восстановление базы данных. Виды аварийных ситуаций. Системный журнал СУБД (назначение, сохраняемые данные, протокол WAL). Индивидуальный откат транзакции. Восстановление после мягкого сбоя. Восстановление после жесткого сбоя.

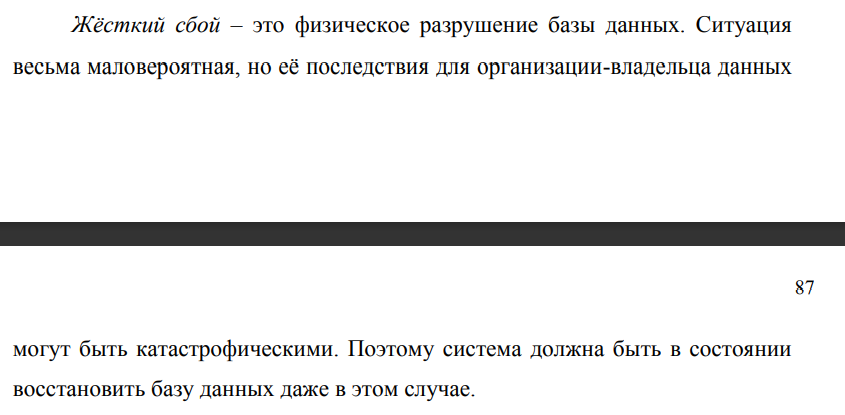
## 10.1 Восстановление базы данных

* Локальный сбой- аварийное или принудительное прекращение ОДНОЙ транзакции (Например:попытка деления на ноль или нарушение ограничений целостности.)
* Мягкий сбой- Аварийное прекращение ВСЕХ транзакций (следствие аварийного отключения питания или при возникновении неустранимого сбоя процессора и т.п.).
* Жесткий сбой-Физическое разрушение базы данных.

Дополнение:







## 10.2 Системный журнал СУБД (назначение, сохраняемые данные, протокол WAL).

Системный журнал СУБД- Служебный файл,содержащий информацию, необходимую для восстановления БД.

Записи журнала содержат детальные сведения обо всех операциях каждой транзакции. В них для каждой *транзакции* фиксируется:

* идентификатор транзакции;
* идентификатор пользователя, запустившего транзакцию;
* время начала транзакции;
* время и способ завершения транзакции.

Для каждой операции фиксируется:

* идентификатор транзакции, в которой выполнялась операция;
* время начала операции;
* идентификатор обрабатываемого элемента данных;
* тип операции;
* копия элемента данных до операции (для операций обновления и удаления);
* копия элемента данных после операции (для операций обновления значения и вставки).

Обновления, выполненные транзакцией, не могут попасть во внешнюю память раньше, чем соответствующие ей записи системного журнала -Это правило составляет суть протокола предварительной записи в журнал (протокол WAL – Write Ahead Log).

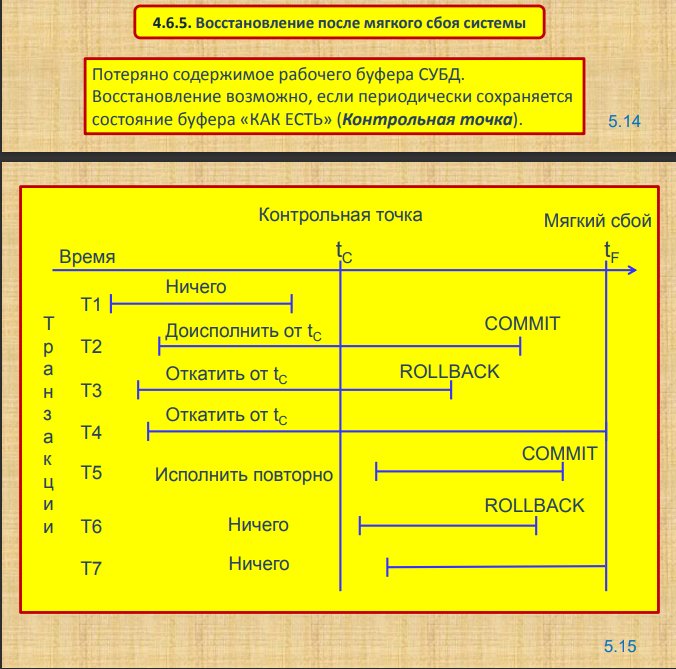
## 10.3 Индивидуальный откат транзакции.

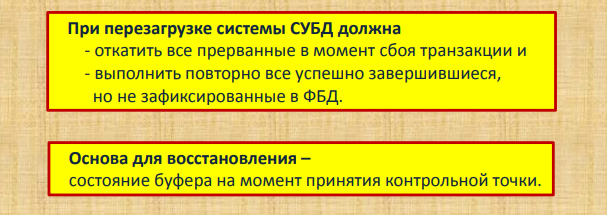
# 

Дополнение: Для осуществления отката создается список записей журнала от данной транзакции. Элементы списка размещены в порядке, обратном хронологическому. Список последовательно просматривается и для каждой записи выполняется противоположная по смыслу операция, восстанавливающая предыдущее состояние объекта базы данных. Начальным состоянием для процедуры отката является состояние буферов БД в момент прекращения транзакции.

С точки зрения системы процедура отката является транзакцией. Поэтому обратные операции также регистрируются в журнале. Это «перестраховка» на случай мягкого сбоя системы в процессе отката. Имея эти записи, система сможет «дооткатить» транзакцию, откат которой был прерван. Восстановление после мягкого сбоя. Восстановление после жёсткого сбоя.

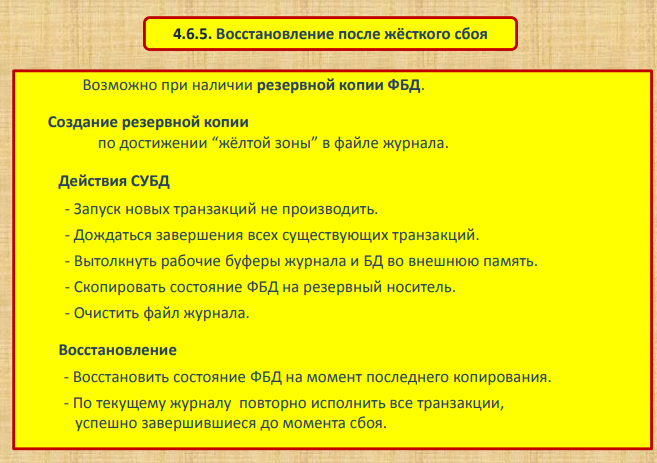
## 10.4 Восстановление после мягкого сбоя





Дополнение: стр 92.Основы технологии баз данных

## 10.4 Восстановление после жесткого сбоя



Дополнение: При жёстком сбое физическая база данных оказывается разрушенной. Поэтому для восстановления необходимо иметь её резервную копию. Обычно резервное копирование ФБД выполняется по факту переполнения системного журнала. Для этого в файле журнала устанавливается так называемая «жёлтая зона», по достижении которой запуск новых транзакций не производится. Система дожидается окончания всех существующих транзакций. После этого рабочие буферы журнала и базы данных выталкиваются во внешнюю память. Созданное таким образом состояние ФБД копируется на резервный носитель, а файл журнала очищается. Может быть также создана резервная копия журнала.

При восстановлении после жёсткого сбоя восстанавливается состояние ФБД на момент последнего копирования, а затем по текущему журналу регистрации повторно исполняются все транзакции, успешно завершившиеся до момента сбоя.

Поскольку жёсткий сбой не сопровождается потерей буферов, после восстановления можно даже продолжить исполнение не завершившихся до сбоя транзакций, но обычно все незавершённые транзакции откатываются.

# 

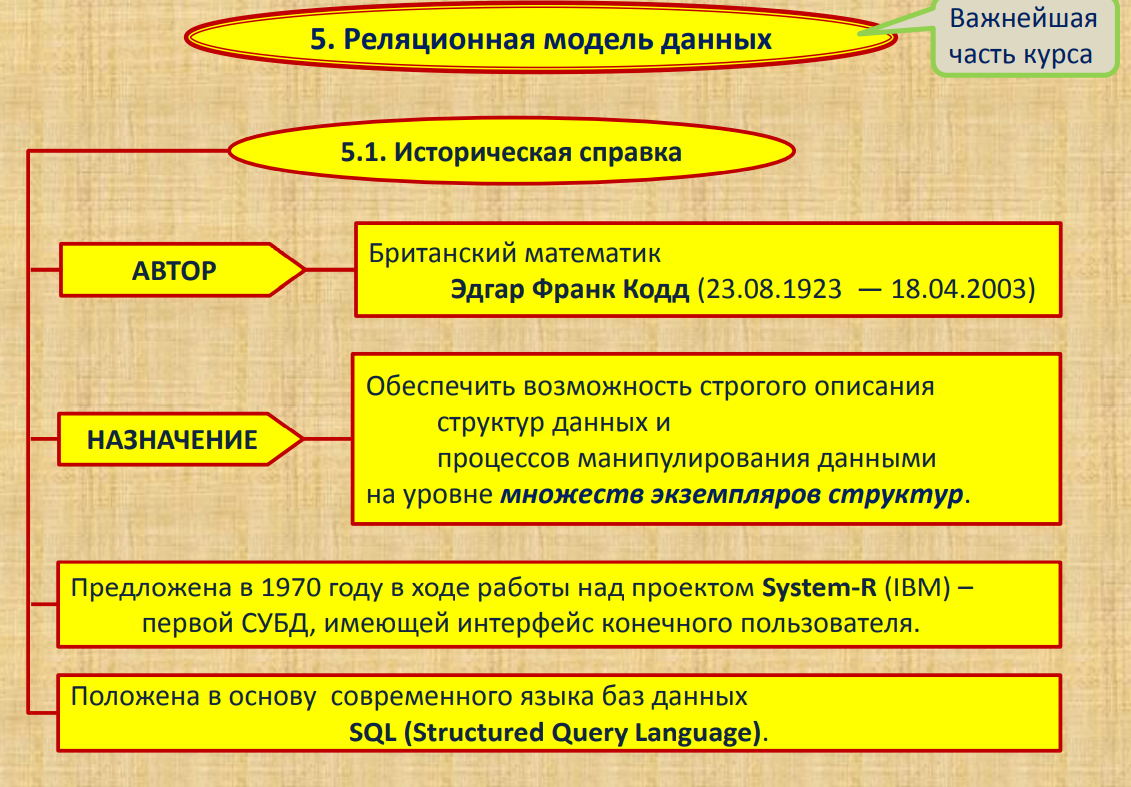
# 11. Функции СУБД (восемь сервисов Кодда).

Требования к функциям и службам, которые должна поддерживать полномасштабная СУБД, сформулированы в 1982 году Э. Коддом и известны как «восемь сервисов Кодда».

1. СУБД должна предоставлять пользователям возможность сохранять, извлекать и обновлять данные в базе данных.
2. СУБД должна поддерживать доступный пользователям каталог, в котором хранятся описания элементов и структур данных.
3. СУБД должна иметь механизм, гарантирующий выполнение либо всех операций транзакции, либо ни одной из них. Эта служба обеспечивает *атомарность* и (совместно со службами 5 и 8) *согласованность* транзакции.
4. СУБД должна иметь механизм, гарантирующий корректное обновление базы данных при параллельном выполнении операций обновления многими пользователями. Эта служба обеспечивает *изолированность* транзакций.
5. СУБД должна предоставлять средства восстановления базы данных в случае какого-либо её повреждения или разрушения. Эта служба обеспечивает *долговечность* транзакций.
6. СУБД должна иметь механизм, гарантирующий доступ к данным только санкционированных пользователей. Этот механизм обеспечивает *безопасность* данных, уменьшает возможности их некомпетентного или злонамеренного использования.
7. СУБД должна иметь механизмы взаимодействия с различными коммуникационными программными средствами. Современные системы баз данных реализуются в технологии «клиентсервер». Приложение, размещённое на клиентском узле сети, обращается к серверу БД и получает сообщения от него через сеть, управляемую менеджером обмена данными (МОД). МОД не является частью СУБД, но жизнеспособная СУБД должна быть способна к интеграции с различными МОД.
8. СУБД должна иметь средства контроля данных на соответствие заданным правилам целостности. Эта служба обеспечивает предотвращение ошибок пользователей при обновлении данных, а также поддержание делового регламента организации в рамках известных системе правил.

# 

# 12. Назначение и составные части реляционной модели данных (РМД).



Модель состоит из трёх частей — структурной, целостностной и манипуляционной.

**Структурная** часть ставит в соответствие интуитивному понятию таблицы математический объект — отношение и определяет набор абстрактных языковых конструкций для описания отношений.

**Целостностная** часть содержит правила, которым должны подчиняться хранящиеся в любой реляционной БД непротиворечивые правдоподобные данные.

**Манипуляционная** часть предоставляет набор операций над отношениями и набор специальных языковых средств для точного формулирования требований к данным, которые должны быть извлечены из БД.

Под манипулированием данными в РМД понимается процесс исполнения запроса к РБД. Существует два вида запросов: на выборку данных и на обновление данных. В настоящей главе обсуждаются средства записи запросов на выборку – языки манипулирования данными (ЯМД).

Результатом исполнения запроса на выборку всегда является некоторое производное отношение, кортежи которого составлены из значений данных, содержащихся в кортежах других отношений – источников данных. В РМД имеется два абстрактных ЯМД: реляционная алгебра (РА) и реляционное исчисление (РИ).

# 

# 

# 13. Структурные понятия реляционной модели данных: домен, атрибут, схема отношения, кортеж, отношение. Свойства отношений.

Выделяется шесть базовых понятий РМД: тип данных, домéн, атрибут, схема отношения, кортеж, отношение. Первые три относятся к элементам данных, остальные – к структурам, объединяющим элементы.

**Тип данных** есть множество значений, не имеющих внутренней структуры.

**Домéн** есть подмножество элементов (значений) типа

Формально домéн определяется как пара (тип, предикат). Предикат задаёт условия принадлежности элемента типа домéну. На одном и том же типе можно определить произвольное число домéнов. Например, на символьном типе данных (CHAR, TEXT и т.п.) можно определить домéны: · рабочих дней недели как множество значений {‘пн’, ‘вт’, ‘ср’, ‘чт’, ‘пт’}; · номеров телефонов как множество десятизначных последовательностей цифр; · русских мужских имён (путём прямого перечисления) · и т.д.

**Атрибут** есть имя, поставленное в соответствие домéн

Если домéну поставлено в соответствие имя, то говорят, что на домéне определен атрибут. Атрибут принимает значения на домéне и наследует его свойства.

Схема отношения

Пусть D1, D2, …Dn — домéны (необязательно различные) и А1, А2, …, Аn — атрибуты, определенные на соответствующих домéнах.

Множество пар: R = {(D1, A1), (D2, A2), ..., (Dn, An)} называется схемой отношения.

Интуитивно схему отношения можно понимать как заголовок таблицы. Атрибуты — это имена столбцов таблицы, домéны — множества значений, которые могут встретиться в столбцах.

Кортеж.

Множество пар: SR = {(A1, a1), (A2, a2),…, (An, an)}, ai Î Di, i = 1, …, n, называется кортежем, соответствующим схеме R.

На интуитивном уровне кортеж представляется как строка таблицы с заданным заголовком. Например, пусть Номера — домéн трёхсимвольных строк, составленных из цифр ‘0’, ‘1’,...’9’, Имена — домéн строк, составленных из символов русского алфавита, пробелов и точек, СЛУЖАЩИЙ – схема отношения: {(Номера, НомерСлуж), (Имена, ИмяСлуж)}. Кортежи этого отношения могут быть такими: {(НомерСлуж, ‘345’), (ИмяСлуж, ‘Иванов И.И.’)}, {(НомерСлуж, ‘938’), (ИмяСлуж, ‘Петров П.П.’)}

Отношение

Множество кортежей SR, соответствующих одной и той же схеме R, называется отношением.

Отношение есть структурный тип данных. Тип определяется схемой отношения. Все кортежи — значения типа — удовлетворяют одной и той же схеме.

Можно говорить об экземпляре (текущем значении) отношения с заданной схемой. Экземпляр (значение) отношения — это набор кортежей с заданной схемой, существующий в некоторый фиксированный момент времени2 . Отношение характеризуется: · арностью (степенью) — числом пар (домен, атрибут) в схеме; · мощностью — числом кортежей, составляющих тело отношения. Степень — это характеристика типа, а мощность — характеристика экземпляра отношения. Степень отношения фиксирована. Мощность может изменяться во времени.

Отношение является единственной структурной единицей РМД.

Свойства отношений

Атомарность значений атрибутов. Схема отношения может включать только атрибуты, принимающие значения простых типов данных

Неупорядоченность атрибутов. Порядок следования атрибутов в схеме отношения не должен поддерживаться РСУБД

Уникальность атрибутов. Одноименные атрибуты в схеме отношения недопустимы.

Уникальность кортежей. Так как отношение есть множество кортежей, в нём не может быть дубликатов кортежей.

Неупорядоченность кортежей. Это следствие того, что отношение есть множество кортежей.

Изменяемость отношений. Тело отношения (набор кортежей) может изменяться во времени.

# 14. Понятие целостности данных. Внутренние ограничения целостности РМД. Требования целостности домена и атрибута. NULL-значения и целостность атрибута.

**Виды внутренних ограничений целостности РМД**

Внутренние ОЦ РМД — это набор средств описания делового регламента (бизнес-правил) предметной области. Мы уже упоминали об этих ОЦ, когда говорили о структурах. Резюмируем сказанное выше и обсудим роль внутренних ограничений целостности в РБД. Принято выделять четыре вида внутренних ОЦ РМД: ограничения домéна, атрибута, сущности и ссылочные

Ограничение целостности домена

Согласно определению домéн есть множество значений простого типа данных. Домéн может быть задан либо путём прямого перечисления всех его значений, либо указанием предиката, определяющего условие принадлежности значения домéну. С точки зрения программиста, домéн — это тип данных, определённый пользователем.

Ограничение целостности атрибута

Согласно определению атрибут есть имя, принимающее значения на домене.

Требование целостности атрибута: значения атрибута должны выбираться только из его домéна.

NULL-значения и целостность атрибута

Возможность появления неопределённых значений атрибутов в РБД разрушает концепцию реляционного домена. Теперь уже нельзя утверждать, что значение каждого атрибута должно выбираться из его домена. Оно может выбираться из домена или быть «недействительным» — NULL. В ряде случаев допустимость NULL-значения атрибута может быть оправдана его смыслом и правилами бизнеса, но далеко не всегда. В самом деле, можно понять, почему значение атрибута ДатаОкончанияЛечения в некотором кортеже БД больницы не определено. Но почему в соседнем кортеже не определено значение атрибута ДатаПоступленияБольного, понять невозможно. Больной ещё не поступил? Тогда зачем мы создали эту запись? Больной поступил, но не известно, когда? А что за раззява его приняла? А не уволить ли её с работы? И т.п. Для того чтобы не возникало подобных недоразумений, для каждого атрибута каждого отношения следует указать допустимость/недопустимость NULL-значений. С учётом этого требование целостности атрибута должно быть сформулировано так Каждое определённое значение атрибута должно выбираться только из его домéна

# 

# 15. Определение потенциального ключа отношения. Понятие первичного ключа и требование целостности сущности. NULL-значения атрибутов и целостность сущности. Практический смысл требования целостности сущности.

**Потенциальный ключ** — в реляционной модели данных — подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости).

Уникальность означает, что не существует двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают (равны).

Минимальность (несократимость) означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности. Иными словами, если из потенциального ключа убрать любой атрибут, он утратит свойство уникальности.

Поскольку все кортежи в отношении по определению уникальны, в нём всегда существует хотя бы один потенциальный ключ (например, включающий все атрибуты отношения).

В отношении может быть одновременно несколько потенциальных ключей. Один из них может быть выбран в качестве первичного ключа отношения, тогда другие потенциальные ключи называют альтернативными ключами

Теоретически, все потенциальные ключи равно пригодны в качестве первичного ключа, на практике в качестве первичного обычно выбирается тот из потенциальных ключей, который имеет меньший размер (физического хранения) и/или включает меньшее количество атрибутов.

**Первичным** **ключом** **называется** **совокупность** **атрибутов, однозначно** **идентифицирующих** **запись** **в** **таблице** **(сущности)**. Один из возможных ключей становится первичным ключом. На диаграммах первичные ключи часто изображаются выше основного списка атрибутов или выделяются специальными символами. Сущность на рисунке имеет как ключевые, так и обычные атрибуты. Альтернативные ключи. Любой возможный ключ, не являющийся первичным, называется альтернативным ключом.

**Требование** **целостности** **сущностей** заключается в том, чтобы каждый кортеж любого отношения отличался от другого кортежа этого отношения (т.е. любое отношение должно обладать первичным ключом).

NULL-значения и целостность атрибута

Возможность появления неопределённых значений атрибутов в РБД разрушает концепцию реляционного домена. Теперь уже нельзя утверждать, что значение каждого атрибута должно выбираться из его домена. Оно может выбираться из домена или быть «недействительным» — NULL. В ряде случаев допустимость NULL-значения атрибута может быть оправдана его смыслом и правилами бизнеса, но далеко не всегда. В самом деле, можно понять, почему значение атрибута ДатаОкончанияЛечения в некотором кортеже БД больницы не определено. Но почему в соседнем кортеже не определено значение атрибута ДатаПоступленияБольного, понять невозможно. Больной ещё не поступил? Тогда зачем мы создали эту запись? Больной поступил, но не известно, когда? А что за раззява его приняла? А не уволить ли её с работы? И т.п. Для того чтобы не возникало подобных недоразумений, для каждого атрибута каждого отношения следует указать допустимость/недопустимость NULL-значений. С учётом этого требование целостности атрибута должно быть сформулировано так: Каждое определённое значение атрибута должно выбираться только из его домéна.

Практический смысл требования целостности сущности

В РМД объект реального мира представляется кортежем некоторого отношения. Требование целостности сущностей – это любой кортеж каждого отношения отличается от всякого кортежа этого отношения, т.е. каждое отношение должно иметь первичный ключ.

Если указанное требование не выполняется (т.е. кортежи не уникальны в рамках одного отношения), тогда в БД содержится противоречивая информация об одном и том же объекте. Целостность сущностей поддерживается средствами СУБД. Это обеспечивается с помощью таких ограничений как:

● при добавлении в таблицу записей осуществляется проверка уникальности их первичных ключей;

● не допускается редактирование значений атрибутов, которые входят в первичный ключ.

Правило: атрибуты, которые входят в состав потенциального ключа не могут иметь null-значений.

# 

# 16. Определение внешнего ключа. Роль внешних ключей в реляционной базе данных. Типы связей отношений, поддерживаемые реляционной моделью данных. Требование ссылочной целостности. NULLзначения атрибутов и ссылочная целостность. Типовые правила ссылочной целостности.

**Внешний** **ключ** (FK) — это столбец или сочетание столбцов, которое применяется для принудительного установления связи между данными в двух таблицах с целью контроля данных, которые могут храниться в таблице **внешнего** **ключа**.

**Внешний** **ключ** – служит как ограничение целостности связей нескольких таблиц, конкретно подчиненная таблица не может ссылаться на несуществующие записи главной таблицы(что позволяет строить целостные модели **данных**).

Связь «Один к одному»

**Связь** "один-**к**-**одному**" — это **связь** между информацией из двух таблиц, когда каждая запись используется в каждой таблице только один раз. Например, **связь** типа "один-**к**-**одному**" может использоваться между сотрудниками и их служебными автомобилями. Каждый работник указан в таблице "Сотрудники" только один раз, как и каждый автомобиль в таблице "Служебный транспорт".

Связь «Один ко многим»

Это наиболее часто встречаемый тип связей. В этом типе связей несколько строк из дочерний таблицы зависят от одной строки в родительской таблице. Например, в одном блоге может быть несколько статей. В этом случае таблица блогов является родительской, а таблица статей - дочерней.

Связь многие-ко многим

**Связь** **многие**-**ко**-**многим** – это **связь**, при которой множественным записям из одной таблицы (A) могут соответствовать множественные записи из другой (B). Примером такой **связи** может служить школа, где учителя обучают учащихся. В большинстве школ каждый учитель обучает многих учащихся, а каждый учащийся может обучаться несколькими учителями. Связь между поставщиком пива и пивом, которое они поставляют – это тоже связь многие-ко-многим.

**Требование ссылочной целостности** состоит в следующем: В любой момент времени каждое определённое значение внешнего ключа должно принадлежать множеству значений ссылочного первичного ключа, содержащихся в существующих кортежах родительского отношения.

NULL-значения атрибутов и целостность сущности

Итак, NULL-значения первичных ключей запрещены. А как быть с NULL-значениями внешних ключей? Ведь вполне возможны такие ситуации, когда в поле внешнего ключа приходится «оставлять дырку». Например, в БД хранятся сведения о сотрудниках конструкторского бюро и о проектах, которые выполняются сотрудниками. По правилам организации проект выполняется несколькими сотрудниками, но сотрудник может участвовать только в одном проекте. Тогда все сведения о сотрудниках, проектах и участии сотрудников в проектах можно представить двумя отношениями: СОТРУДНИК (ТабельныйНомер, ФИО,…, КодПроекта); ПРОЕКТ (КодПроекта, …). Здесь ПРОЕКТ.КодПроекта — первичный ключ отношения ПРОЕКТ, а СОТРУДНИК.КодПроекта — соответствующий ему внешний ключ отношения СОТРУДНИК. Однако не каждый сотрудник КБ участвует в каком-то проекте. Есть ведь и вспомогательный персонал, об участии которого в конкретных проектах говорить нет смысла. Каким будет значение внешнего ключа в кортеже отношения СОТРУДНИК, представляющем вспомогательного сотрудника? Правильно, неопределённым, NULL. Тогда приходится требование ссылочной целостности «подправить» так: База данных не должна содержать определённых значений внешнего ключа, не существующих среди значений первичного ключа родительского отношения. Вот сколько проблем связано с «пустым» с точки зрения человека, работающего с бумажной таблицей, вопросом: «Как отображать в таблице неизвестные значения столбцов в некоторых строках?»

# 

# 17. Реляционный язык определения данных РМД.

## 17.1 Объявление домена

CREATE DOMAIN имя\_домена тип [(длина)]

{VALUES (список)} |

{FOR ALL VALUE (предикат)};

Здесь ***имя-домена*** — уникальное имя, под которым домен будет известен системе, и на которое можно ссылаться в определениях отношений;

***тип*** — один из поддерживаемых системой встроенных типов данных;

**длина** — длина поля данных в байтах;

***список*** — список значений, разделенных запятыми;

***предикат*** — логическое выражение, ссылающееся на переменную VALUE.

Эта декларация объявляет системе, что она должна внести в свой каталог имя нового объекта — домена, и указывает ограничения на значения, принадлежащие домену. Определение сохраняется в системном каталоге. При любой попытке обновления значения какого-либо атрибута, определенного на этом домене, будет вычислено значение предиката. Если оно окажется равным .FALSE., обновление будет отвергнуто.

Если есть возможность создать домен, должна быть и возможность удалить его. Следующая команда удалит из системного каталога определение домена:

DESTROY DOMAIN имя\_домена;

Она будет исполнена, если в схеме БД нет ни одного атрибута, определенного на удаляемом домене.

Если система поддерживает домены, то в ней есть возможность выполнения запросов, основанных на доменах. Например, запрос

*Какие отношения содержат какую-либо информацию о весе?*

в терминах доменов имеет вид:

*Какие отношения включают атрибуты, объявленные на домене* Вес*?*

Это запрос к системному каталогу. В системе, не поддерживающей домены, такие сведения практически невозможно получить.

## 17.2 Объявление отношения

Можно говорить о ***переменной***, заданной схемой отношения, и об её ***значении*** – отношении, существующем в БД в конкретный момент времени. Объявлять следует только схему, т.е. переменную. Значения отношения формируются системой в процессе выполнения запросов пользователей.

В реляционной БД всегда существует несколько видов отношений.

* ***Именованное отношение*** — это отношение, имя и определение схемы которого сохранены в системном каталоге. Именованное отношение может быть базовым или производным.
* ***Производное отношение*** есть отношение, определённое через именованные отношения. Производное отношение может быть именованным или неименованным.
* ***Базовое отношение*** — именованное отношение, не являющееся производным.

Любое производное отношение, в конце концов, выражается через базовые. Поэтому средства явного определения схемы нужны только для базовых отношений.

Предложение объявления базового отношения имеет вид:

CREATE BASE RELATION имя\_отношения

( список\_определений\_атрибутов

список\_определений\_возможных\_ключей

список\_определений\_внешних\_ключей

);

Здесь **список\_определений** ... — список разделённых запятыми строк, содержащих определения элементов схемы отношения.

строка\_определения\_атрибута ::=

имя\_атрибута DOMAIN имя\_домена

Указанный атрибут принимает значения на указанном домене.

Например, строка объявления атрибута ВесДетали в предложении объявления отношения ДЕТАЛЬ может иметь вид:

ВесДетали DOMAIN (Вес)

Предполагается, что домен Вес определён в схеме БД.

Имена атрибутов должны быть уникальными в пределах отношения.

строка\_определения\_возможного\_ключа ::=

PRIMARY KEY (список\_атрибутов) |

CANDIDATE KEY (список\_атрибутов)

Здесь PRIMARY KEY и CANDIDATE KEY объявляют, соответственно, первичный и альтернативный ключи отношения;

***список\_атрибутов*** — список разделённых запятыми имён атрибутов, образующих ключ

Строка объявления внешнего ключа имеет вид:

FOREIGN KEY (список\_атрибутов)

REFERENCES имя\_отношения

ON DELETE правило\_удаления

ON UPDATE правило\_обновления

Здесь ***список\_атрибутов*** — список разделённых запятыми имён атрибутов (составного) внешнего ключа, эквивалентный списку атрибутов родительского ключа;

***имя\_отношения*** — имя родительского отношения;

правило\_... следует понимать как ссылку на процедуру БД, реализующую определённое проектировщиком правило внешнего ключа для операции удаления или обновления родительского ключа. Это может быть либо процедура пользователя, либо одна из стандартных процедур СУБД — CASCADE (каскадировать) или RESTRICT (отложить).

Базовое отношение может быть уничтожено командой

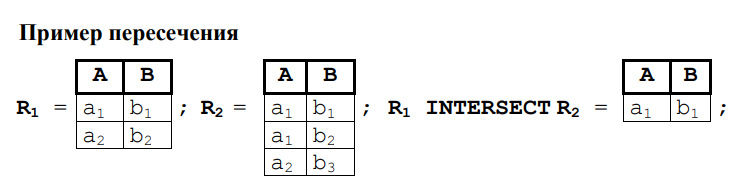
DESTROY BASE RELATION имя\_отношения;

Команда удалит из системного каталога все сведения об указанном отношении, но будет выполнена, только если тело отношения пусто. Будут удалены также все производные отношения, в определениях которых есть ссылки на удаляемое.

# 18. Определения теоретико-множественных операций реляционной алгебры: пересечение, объединение, разность, расширенное прямое произведение.

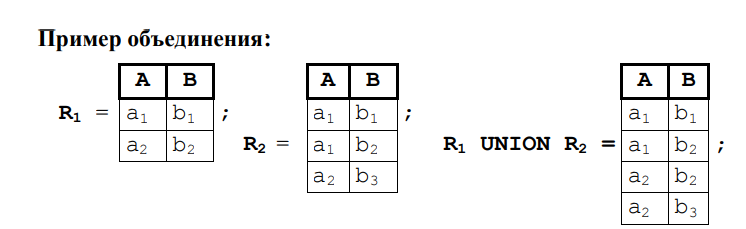
## 18.1 Пересечение

Пересечением отношений со схемами R1(X) и R2(X) называется отношение R = R1 INTERSECT R2, имеющее схему R(X) и тело, составленное только из тех кортежей X:x R1, которые принадлежат R2.



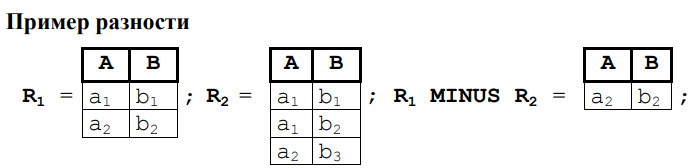
## 18.2 Объединение

Объединением отношений со схемами R1(X) и R2(X) называется отношение R = R1 UNION R2, имеющее схему R(X) и тело, составленное из всех кортежей X:x, принадлежащих хотя бы одному из операндов.



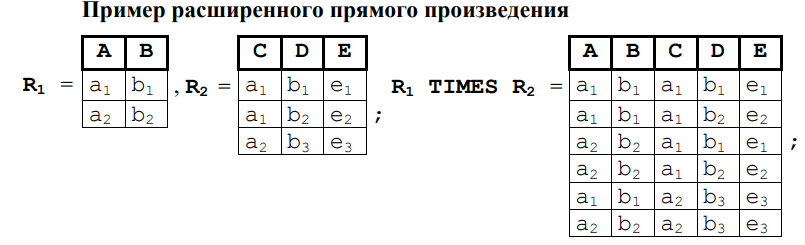
## 18.3 Разность

Разностью отношений со схемами R1(X) и R2(X) называется отношение R = R1 MINUS R2, имеющее схему R(X) и тело, составленное только из тех кортежей X:x R1, которые не принадлежат R2.



## 18.4 Расширенное прямое произведение

Пусть отношения со схемами R1(X) и R2(Y) совместимы по взятию расширенного прямого произведения. Расширенным прямым произведением отношений R1 и R2 называется отношение R = R1 TIMES R2, имеющее схему R(X, Y) и тело, образованное всеми кортежами {X:x, Y:y} такими, что в R1 существует кортеж X:x и в R2 существует кортеж Y:y.



# 

# 19. Определение реляционного исчисления с переменными-кортежами.

Основное понятие этого варианта РИ – понятие переменной-кортежа.

Пусть R – некоторое отношение. Величина t, значениями которой являются кортежи отношения R, называется переменной-кортежем, определённой на R. На интуитивном уровне переменная-кортеж это перемещающееся по строкам таблицы окно, в котором всегда видна одна строка – текущее значение переменной. Для записи выражения РИ кортежей нужно указать области определения переменных, определить схему целевого отношения и сформулировать условия, которым должны удовлетворять кортежи его тела.

# 

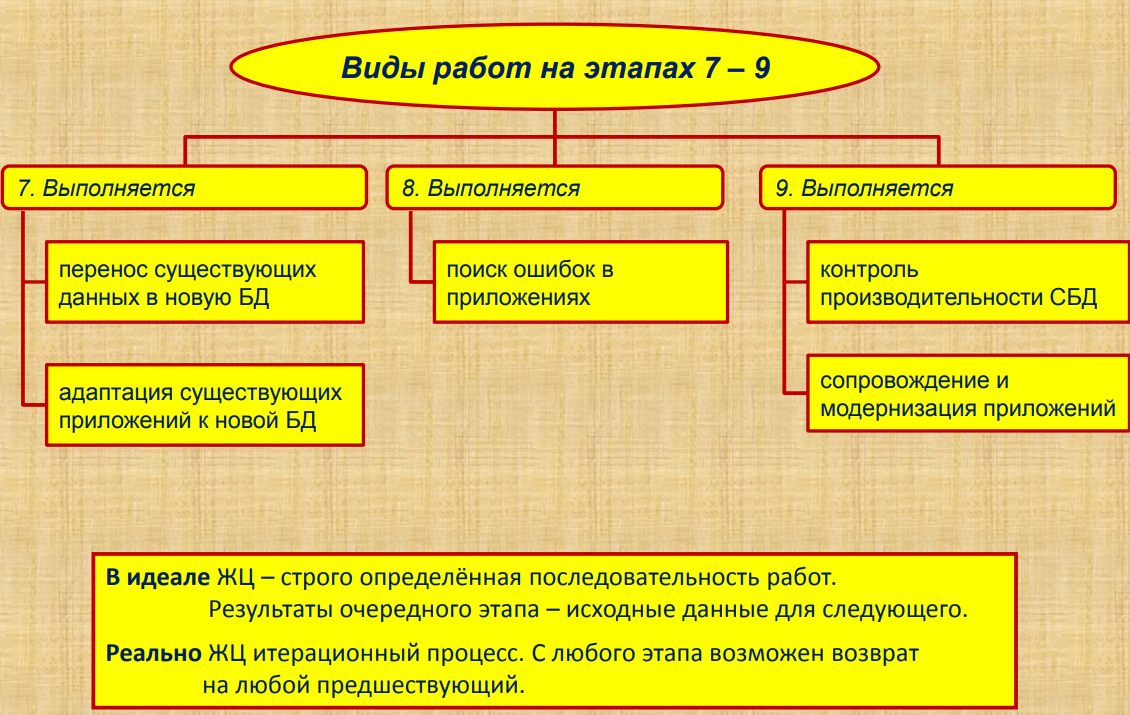
# 20. Понятие жизненного цикла системы с базами данных. краткая характеристика этапов.

**ОДИН В ОДИН 23 ВОПРОС, поэтому хер знает**

Под жизненным циклом (ЖЦ) системы баз данных (СБД) понимается непрерывный процесс, который начинается в момент принятия решения о создании СБД и заканчивается в момент полного изъятия системы из эксплуатации. Этот процесс принято подразделять на ряд этапов (фаз). На каждом этапе выполняется определённый перечень работ, направленных на создание или эксплуатацию и развитие системы. Следующая схема представляет идеальную временнýю последовательность этапов ЖЦ СБД

# 





# 21. Модель «сущность-связь». Назначение, основные понятия, нотации. Пример диаграммы с интерпретацией.

По существу ER-модель данных — это графический язык для описания объектов и отношений объектов. Спецификации требований пользователя представляются в виде диаграммы, показывающей объекты ПО, их связи и свойства объектов и связей — ER-диаграммы.

Какие бы нотации ни использовались, ER-диаграмма наглядно и точно отражает представления автора о требованиях пользователя к данным.

**ОСНОВНЫЕ ПОНЯТИЯ**

**Сущность** — это объект, представляющий интерес для пользователя БД.

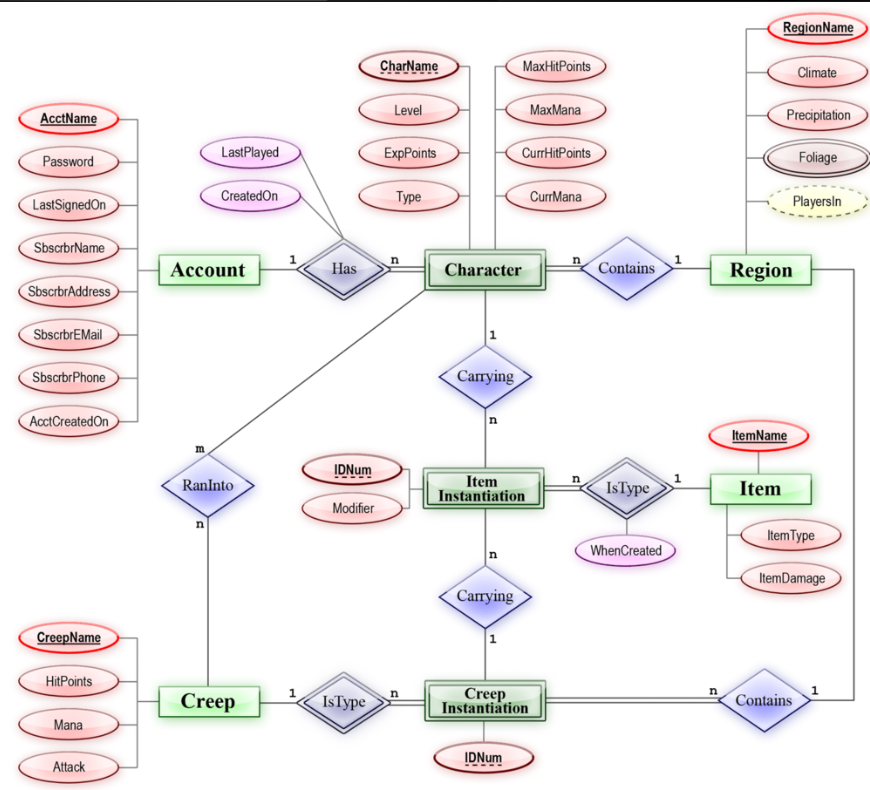
**Класс сущностей** — потенциальное множество объектов одного типа. Класс имеет имя, идентифицирующее тип объекта, и фиксированный набор свойств — атрибутов. Формально класс сущностей — это именованный набор атрибутов.

**Экземпляр сущности** — это набор значений атрибутов соответствующего класса сущностей (n-ка, кортéж).

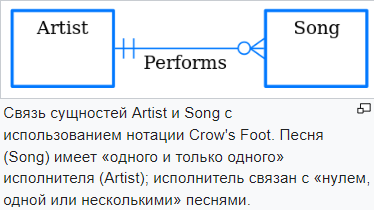
**Связь сущностей** — это осмысленная с точки зрения пользователя ассоциация сущностей.

**Класс связей** — ассоциация классов сущностей. Экземпляр класса связей есть ассоциация экземпляров связанных классов сущностей, в которой участвует точно один экземпляр каждого класса-участника связи.

**Атрибут** — это свойство сущности. Он имеет имя и принимает значения из некоторого множества значений определённого типа данных — домéна



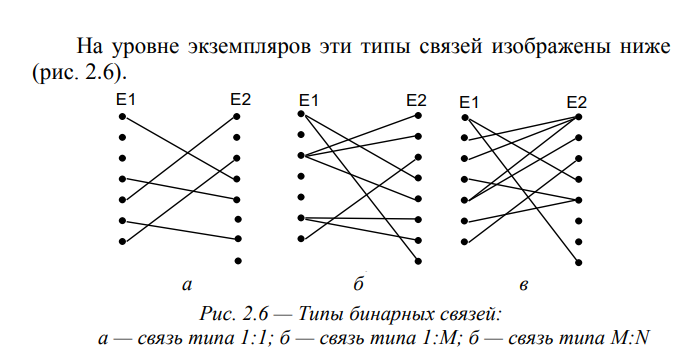
Нотации Чена

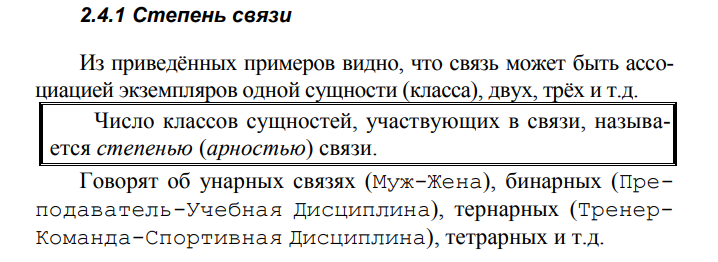


# 22. Модель «сущность-связь. Понятие связи. Типы связей. Свойства связей: степень (арность), мощность, обязательность / необязательность. Приведение связи высшей арности к совокупности бинарных связей.

**Связь сущностей** — это осмысленная с точки зрения пользователя ассоциация сущностей.

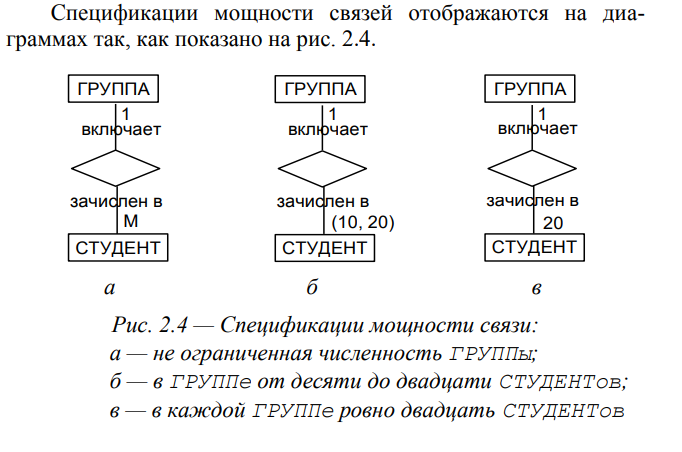
**Класс связей** — ассоциация классов сущностей. Экземпляр класса связей есть ассоциация экземпляров связанных классов сущностей, в которой участвует точно один экземпляр каждого класса-участника связи.





**Мощность связи**

Под мощностью понимается число экземпляров связи, в которые может участвовать один экземпляр сущности. Связь степени n характеризуется n показателями мощности — по одному со стороны каждой сущности. Можно уточнить понятие мощности так.



Любую n-арную связь или бинарную связь типа М:N можно редуцировать — преобразовать в эквивалентную (содержащую ту же информацию) совокупность бинарных связей типа 1:М. Для этого нужно заменить редуцируемую связь ассоциативной сущностью. Эта сущность должна быть потомком каждого участника редуцируемой связи в бинарной связи типа 1:М. В состав атрибутов этой сущности должны входить копии потенциальных ключей каждого участника редуцируемой связи. Любую унарную связь также можно представить с помощью двух бинарных связей типа 1:М. В этом случае ассоциативная сущность должна содержать две копии одного и того же потенциального ключа.

# 

# 23. Жизненный цикл системы баз данных. Основные этапы. Виды работ на этапах.

## 23.1 Жизненный цикл системы баз данных.

Проектирование базы данных (БД) — это один из этапов общего процесса проектирования системы баз данных организации. Поэтому, прежде чем говорить о процедурах проектирования БД, рассмотрим жизненный цикл системы баз данных в целом.

Под **жизненным циклом (ЖЦ) системы баз данных** (СБД) понимается непрерывный процесс, который начинается в момент принятия решения о создании СБД и заканчивается в момент полного изъятия системы из эксплуатации. Этот процесс принято подразделять на ряд этапов (фаз). На каждом этапе выполняется определённый перечень работ, направленных на создание или эксплуатацию и развитие системы. Следующая схема представляет идеальную временнýю последовательность этапов ЖЦ СБД (рис. 23.1.1).

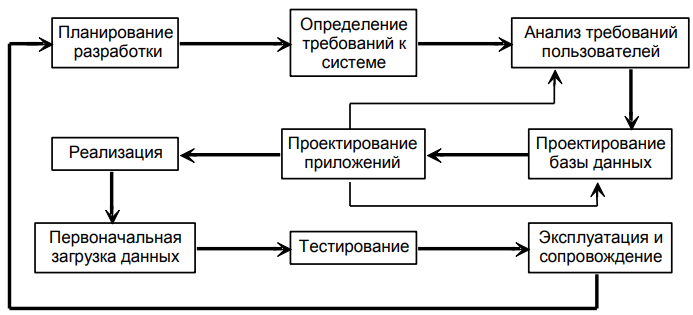


Рисунок 23.1.1 - Жизненный цикл СБД

В идеале результаты очередного этапа должны являться исходными данными для следующего. Однако реальный жизненный цикл — это итерационный процесс. На каждом этапе могут возникнуть проблемы, которые потребуют пересмотра решений, принятых на любом из предшествующих. Поэтому приведенную здесь схему следует рассматривать как перечень видов работ, а не как строгую временнýю последовательность их исполнения.

Рассмотрим фазы ЖЦ подробнее.

## 23.2 Виды работ на этапах жизненного цикла.

1. **Планирование разработки**.

Это подготовительный этап.

Здесь оцениваются:

* требуемый объем работ;
* требуемые ресурсы;
* общая стоимость проекта.

Эти оценки приближенные. Они нужны руководству организации для принятия решения о создании (или радикальной переработке) системы. Обычно этот этап выполняется небольшой группой (2~3 человека) высококвалифицированных и авторитетных специалистов. Возможно привлечение независимых экспертов.

1. **Определение требований к системе**.

На этом этапе определяются диапазон действия системы (поддерживаемые бизнес-процессы), её границы и состав пользователей.

1. **Анализ требований пользователей**.

Это наиболее ответственный этап ЖЦ. Недостаточно тщательно выполненный анализ приведёт к выходу за рамки бюджета проекта, превышению сроков разработки и, возможно, краху проекта. На этом этапе собирается и анализируется информация о деятельности организации, необходимая для проектирования БД и приложений. Выявляются:

* функции пользователей системы,
* данные, необходимые для выполнения функций,
* бизнес-правила, ограничивающие функции.

Эта информация собирается различными способами. В частности:

* путём опроса отдельных наиболее авторитетных сотрудников организации;
* посредством наблюдений за деятельностью организации;
* посредством изучения документов, в особенности тех, которые используются для сбора или представления информации;
* путём анкетирования широкого круга будущих пользователей системы.

Привлекается и опыт разработчиков в проектировании аналогичных систем.

Собранная информация по каждой области применения и каждой группе пользователей должна включать

* входную и выходную документацию организации;
* подробные сведения о выполняемых транзакциях;
* список требований пользователей с указанием приоритетов.

Получаемая аналитиками информация о деятельности организации всегда плохо структурирована. Задача аналитиков — структурировать её и определить спецификации требований пользователей, т.е. дать точное и однозначное описание функций пользователей и их потребностей в данных.

1. **Проектирование базы данных.**

Проектирование БД — это этап создания информационного ядра системы. Основные цели этапа следующие:

* Определить данные и связи между ними, необходимые для всех основных областей применения создаваемой СБД и любых существующих групп пользователей;
* Создать модель данных, способную поддерживать выполнение любых требуемых транзакций обработки данных;
* Разработать предварительный вариант проекта СБД, структура которого позволит удовлетворить основные требования к производительности системы.

Вообще говоря, эти цели противоречивы. Задача проектировщиков — найти удачный компромисс между универсальностью и производительностью системы.

1. **Проектирование приложений.**

Проектирование приложений — это процесс создания интерфейса конечного пользователя и прикладных программ, предназначенных для обработки данных.

Реально проектирование БД и приложений выполняется параллельно. БД предназначена для поддержки приложений. Поэтому её проект должен учитывать нужды приложений, отражающие потребности локальных пользователей. Однако и приложения невозможно создавать, не зная структуры БД. Поэтому между фазами проектирования БД и приложений идёт постоянный обмен информацией, отраженный на рис. 23.1.1 дугами обратных связей. Проектирование приложений невозможно завершить до завершения проектирования БД.

Главная задача этой фазы — создание интерфейса конечного пользователя для всех функций системы, предусмотренных спецификациями требований пользователей.

Современные технологии разработки приложений реализуют концепцию прототипов (рис. 23.3.1).

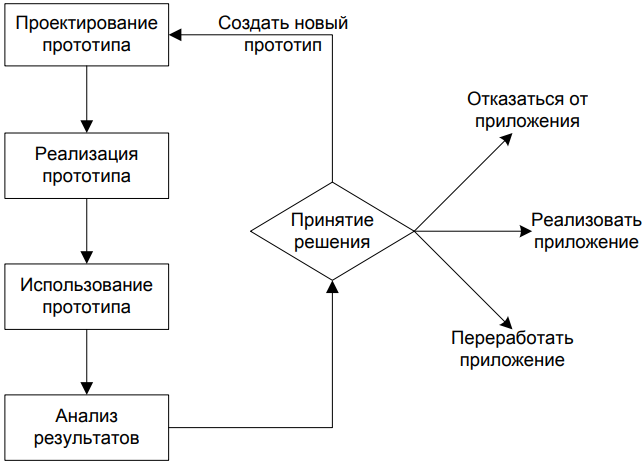


Рисунок 23.3.1 - Процесс прототипирования

Прототип — это рабочая модель приложения, обладающая лишь частью требуемой функциональности. Он создается для того, чтобы пользователь мог опробовать его в работе и определить, какие из реализованных функций отвечают своему назначению, а какие — нет. С помощью прототипов разработчики уточняют требования конечных пользователей к системе. Прототипы обычно развиваются в готовые приложения.

1. **Реализация**

На этом этапе:

* создается «пустая база данных»;
* определяются все специфические пользовательские представления;
* реализуются все приложения;
* реализуются все средства защиты данных и поддержания целостности.

Для реализации БД и внешних представлений создают их описание на языке определения данных (ЯОД) целевой системы управления базами данных (СУБД). Это описание используется системой для создания схем данных всех уровней, и пустых файлов физической базы данных (ФБД) на устройствах внешней памяти.

Приложения реализуются на языках программирования высокого уровня (ЯВУ). Транзакции обработки БД описываются на языке манипулирования данными (ЯМД) целевой СУБД. Они могут включаться непосредственно в текст программ, а могут быть реализованы в виде отдельного модуля. Многие современные СУБД имеют собственные среды быстрого создания приложений — языки четвертого поколения (4GL).

Средства защиты данных и поддержания целостности частично описываются средствами ЯОД. Однако возможности СУБД в этом отношении не всегда достаточны. Поэтому часто приходится создавать специальные программы, выполняющие проверки правил целостности (триггеры БД), а также утилиты и специальные приложения, обеспечивающие безопасность данных.

1. **Первоначальная загрузка данных**

Первоначальная загрузка — это перенос любых существующих данных в новую БД и модификация всех существующих приложений с целью обеспечения их совместной работы с новой БД. Работы на этом этапе зависят от того, есть ли в организации компьютерная информационная система. Если новая СБД заменяет старую, то файлы старой ФБД должны быть конвертированы в новые форматы. Каждая современная СУБД имеет утилиту загрузки существующих файлов в новую БД. Этой утилите нужно задать спецификации файлов.

Адаптация старых приложений к новой БД — гораздо более сложная задача. Нередко проще разработать приложение заново, чем адаптировать его.

Если ИС организации создается впервые, то администратор базы данных (АБД) должен:

* выделить подмножество данных организации, которые будут перенесены с бумажных носителей в БД;
* продумать процедуру ввода этих данных и создать вспомогательные приложения, предназначенные исключительно для обеспечения корректного ввода;
* обучить операторов и постоянно контролировать их работу.

1. **Тестирование.**

Тестирование — это процесс выполнения прикладных программ с целью поиска ошибок. Тестирование не может доказать безошибочность программ. С помощью тестов можно лишь обнаружить часть имеющихся ошибок. Если в результате тестирования не выявлено ошибок, то либо тесты были неудачными, либо программа с большой вероятностью работает в соответствии со спецификацией. Существует множество стратегий и методик тестирования программ. Мы не будем обсуждать их здесь, поскольку это предмет технологии программирования, а не технологии БД.

1. **Эксплуатация и сопровождение.**

Эксплуатация и сопровождение — это наблюдение за системой и поддержание её нормального функционирования по окончании развертывания. На этом этапе осуществляется контроль производительности системы и сопровождение и модернизация приложений.

Производительность системы может падать со временем. Основные причины снижения — увеличение объёма хранимых данных и увеличение числа одновременно работающих приложений. Если среднее время реакции системы на запрос приложения становится недопустимо большим, то АБД должен произвести дополнительную настройку системы с целью увеличения производительности. Например, создать дополнительные индексы, объединить или расщепить отдельные таблицы, изменить структуры файлов и т.п. Для того чтобы выполнить настройку, АБД должен располагать сведениями о различных показателях функционирования системы, в частности о частотах обращений к тем или иным фрагментам БД, об эффективности системы блокировок, о выбираемых стратегиях выполнения запросов и т.п. Сбор этой информации обеспечивают специальные утилиты администрирования БД. Как правило, они входят в состав типичной СУБД. Однако если СУБД не предоставляет такие вспомогательные службы, то АБД должен создать их сам, поскольку мониторинг системы необходимо осуществлять в течение всего её жизненного цикла.

В процессе эксплуатации СБД у пользователей появляются новые потребности. Они должны отслеживаться и удовлетворяться. Новые требования могут относиться к функциональности существующих приложений, составу полей отдельных таблиц, составу таблиц БД в целом и т.п. Необходимость учета этих требований возвращает разработчиков на начальный этап ЖЦ.

В конце концов, может наступить такой момент, когда нужно будет принять решение о проектировании новой СБД, поскольку существующая не в состоянии достичь требуемых эксплуатационных показателей.

Если новая система создана и введена в эксплуатацию, то старую не следует немедленно выводить из эксплуатации. Будет лучше, если некоторое время они будут работать параллельно. Это обеспечит подстраховку на случай непредвиденных проблем с новой системой.

# 24. Проектирование базы данных. Цели проектирования. Фазы (этапы) процесса проектирования. Виды работ на этапах.

**Проектирование БД** — это процесс создания БД, предназначенной для поддержки функционирования организации и способствующей достижению её целей.

## 24.1 Цели проектирования

Этот процесс направлен на достижение двух целей:

**Цель 1**) Создать структуры хранения, способные обеспечить накопление данных организации и выполнение всех требуемых видов обработки данных;

**Цель 2**) Создать все приложения, способные обеспечить интерфейс конечных пользователей с базой данных и специфическую обработку данных в соответствии с требованиями различных конечных пользователей.

## 24.2 Фазы процесса проектирования

Описания данных на внешнем и концептуальном уровнях — внешние и концептуальная схемы (ВС и КС) — не привязаны к реализации базы данных. Они описывают логические структуры данных, соответствующие объектам ПО и их отношениям, и не содержат ссылок на программно-техническую платформу проектируемой системы. На этапе разработки ВС и КС платформа ещё не определена. В основе схем лежат *представления конечных пользователей о предметной области.*

*Внешняя схема* соответствует локальному представлению конечного пользователя (КП), исполняющего в организации конкретные функции, например бухгалтера группы материального учета, инспектора отдела кадров, кладовщика и т.п.

*Концептуальная схема* обобщает локальные представления. Её можно понимать как описание представлений *абстрактного пользователя*, исполняющего функции *всех* КП. Любая внешняя схема может быть выведена из концептуальной.

*Внутренняя схема* — это *реализация* концептуальной схемы на конкретной программно-технической платформе. Как и КС, она описывает представления обобщенного пользователя БД, но, кроме того, содержит определения физического уровня.

В соответствии со сказанным, выделяют три фазы проектирования БД (см. рис. 24.2.1):

* концептуальное моделирование;
* логическое моделирование;
* физическое проектирование.

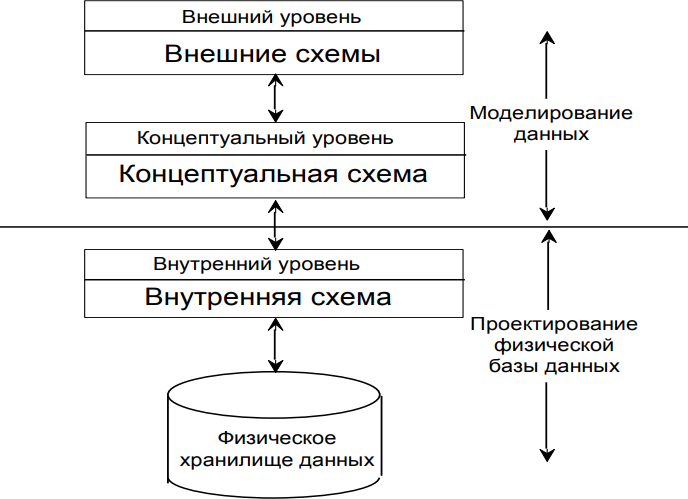


Рисунок 24.21 - Фазы проектирования БД

## 24.3 Виды работ на этих этапа

### 24.3.1 Концептуальное моделирование

**Концептуальное моделирование** — это процесс анализа информационных потребностей конечных пользователей системы. На этой фазе цель аналитика — сформировать в своей голове представления об информационных потребностях бизнеса, адекватные представлениям предполагаемых конечных пользователей системы. Эти представления формируются в процессе изучения деятельности организации. Используются следующие источники информации:

* документы организации (хозяйственные книги, счета, накладные, финансовые отчеты и т.п.);
* собственные наблюдения за работой организации;
* беседы с пользователями;
* опыт предыдущих разработок;
* собственные предположения об информационных потребностях бизнеса.

Обрабатывая полученную информацию, аналитик делает выводы о структуре и связях объектов предметной области. Эти выводы он документирует в модели данных и обсуждает с пользователем. Цель обсуждений — согласование представлений аналитика и пользователя о предметной области системы.

Основная проблема согласования в том, что пользователь и аналитик мыслят разными категориями. Пользователь знает, какие формы и отчёты и с какими данными ему нужны, но он ничего не может сказать о структурах и связях объектов, отражаемых в этих формах и отчетах. Разработчика же интересуют именно структуры и связи объектов. Он вынужден реконструировать объекты и связи из форм и отчётов.

Эта задача, как любая обратная задача, имеет множество решений. В ходе многократных обсуждений аналитик должен выбрать такое, которое адекватно модели данных, имеющейся в голове пользователя. В противном случае результат разработки — готовая система — вряд ли удовлетворит пользователя.

Процесс моделирования больше искусство, чем наука. Существует множество средств и способов моделирования данных. Можно изучить их все. Мы изучим некоторые. Но их использование — это искусство, требующее опыта и интуиции.

Результат концептуального моделирования — концептуальная модель данных пользователя — однозначно описывает структуры и связи объектов предметной области и бизнесправила организации. Это описание не зависит от:

* типа целевой СУБД;
* набора создаваемых программ;
* используемых языков программирования
* типа вычислительной платформы
* а также от любых других особенностей физической реализации.

### 24.3.2 Логическое моделирование

В процессе логического моделирования концептуальная модель уточняется и преобразуется в логическую с учетом базовой модели данных целевой СУБД. Например, если целевая СУБД базируется на РМД, то структуры и связи объектов, представленные в концептуальной модели, преобразуются в систему взаимосвязанных отношений, удовлетворяющих определенным формальным требованиям.

К началу логического моделирования должен быть определён тип целевой СУБД (реляционная, сетевая, объектно-ориентированная…). Однако все прочие аспекты целевой СУБД (физическая организация структур хранения, способы построения индексов и т.п.) полностью игнорируются.

Логическая модель содержит описания структур данных в терминах выбранной модели данных, а также описания ограничений целостности. Она должна:

* быть корректной;
* соответствовать требованиям пользователей, зафиксированным в концептуальной модели;
* обеспечивать поддержку всех необходимых пользователям транзакций.

Кроме того, она играет важную роль на этапе эксплуатации и сопровождения СБД. При правильно организованном сопровождении она составляет основную часть системного каталога и автоматически обновляется при внесении изменений в логические структуры данных. Благодаря этому АБД всегда может точно представить любые вносимые изменения и оценить их влияние на существующие приложения.

Моделирование данных — наиболее ответственный и трудоемкий этап разработки системы. Это практически бесконечный итеративный процесс исследования информационных потребностей предприятия. Каждая итерация вносит уточнения и улучшения в модели, что, в свою очередь, может потребовать изменений в других частях проекта.

Если созданные модели неадекватно отражают представления пользователей о предметной области, то будет очень трудно или даже невозможно определить все необходимые внешние схемы, организовать поддержку целостности данных и обработку необходимых транзакций. Опыт показывает, что успешные проекты используют около 2/3 бюджета времени на моделирование данных. Именно качественные модели являются залогом их успеха.

### 24.3.3 Физическое проектирование

На этой фазе принимаются решения о способах реализации БД. Она может начаться только после выбора конкретной целевой СУБД. Целью физического проектирования является описание способа физической реализации логической модели.

Например, если используется реляционная СУБД, то в её среде должен быть создан набор таблиц и ограничений, представленный в логической модели. Должны быть определены конкретные структуры хранения данных (файлы) и методы доступа к данным, обеспечивающие требуемую производительность системы. Кроме того, должны быть разработаны средства защиты системы.

В идеале фазы логического моделирования и физического проектирования следует разделять, однако реально это невозможно, поскольку решения, принимаемые на фазе физического проектирования с целью повышения производительности системы, могут повлиять на структуру логической модели.

Фазы моделирования и физического проектирования БД предъявляют различающиеся требования к знаниям и профессиональным навыкам проектировщиков. Для успешного моделирования нужно обладать навыками исследовательской работы и владеть методологиями моделирования. Для успешного проектирования ФБД нужно хорошо знать возможности целевой программно-технической платформы, владеть её инструментами и иметь навыки системного программирования..

#### 

# 25. Определение функциональной зависимости атрибутов отношения. Основания для заключения о наличии ФЗ между атрибутами отношения. Способы объявления функциональной зависимости в реляционной модели данных.

## 25.1 Определение функциональной зависимости

Пусть *R* отношение, *А* и *В* — подмножества его атрибутов. Говорят, что *А* функционально определяет *В* или *В* функционально зависит от *А*, если и только если для любого значения отношения *R* два его кортежа, совпадающие по значению *А*, совпадают и по значению *В*.

Обозначение ФЗ: *А→В*.

Атрибут *А* называется детерминантом ФЗ, атрибут *В* — зависимой частью.

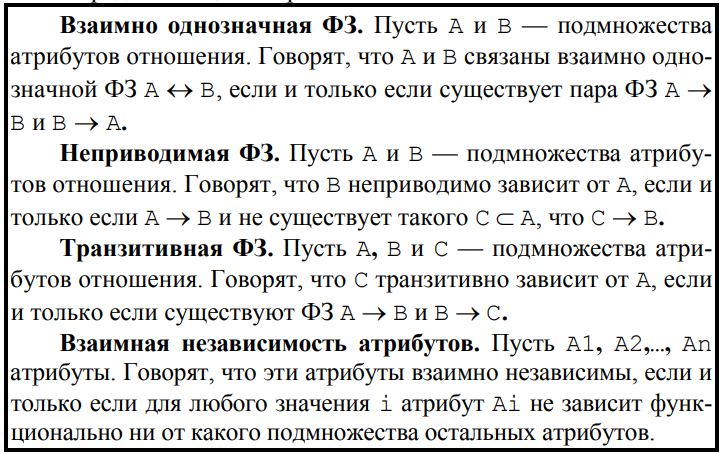
## 25.2 Способы объявления функциональной зависимости в реляционной модели данных

Реляционная СУБД имеет единственный механизм, который можно использовать для поддержки функциональных зависимостей атрибутов. Это механизм контроля уникальности значений. Поэтому возложить на СУБД ответственность за поддержание какой-либо ФЗ можно единственным способом. Нужно создать отношение, потенциальным ключом которого является детерминант ФЗ, а не ключевым атрибутом — зависимая часть.

Отсюда следует простой вывод. В общем случае множество хранимых атрибутов должно быть структурировано, т.е., представлено в виде схем нескольких отношений так, чтобы все ФЗ, заданные бизнес-правилами, могла поддерживать реляционная СУБД. Для этого отношения должны удовлетворять определённым требованиям. Эти требования ограничивают множество допустимых ФЗ в схеме отношения. Они называются нормальными формами отношений.

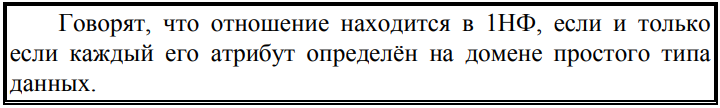
# 26. Понятия взаимной независимости, транзитивной и неприводимой функциональной зависимости. Требования 1НФ, 2НФ, 3НФ, НФБК. Определение многозначной зависимости. Требование 4НФ.

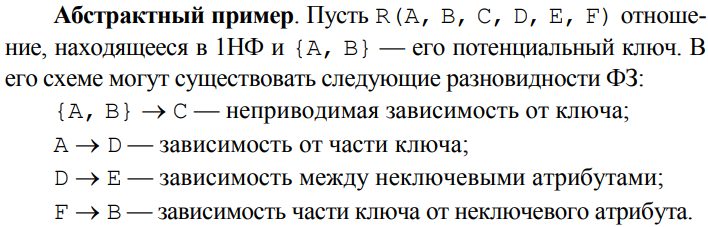
## 26.1 Понятия взаимной независимости, транзитивной, и неприводимой функциональной зависимости



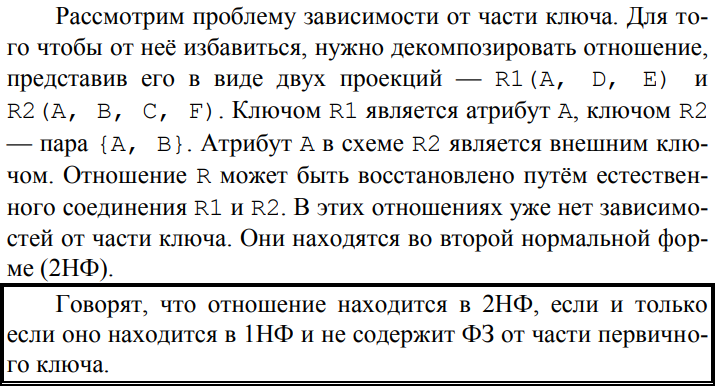
## 26.2 Нормальные формы отношений и проблема аномалий обновлений

### 26.2.1 Первая нормальная форма

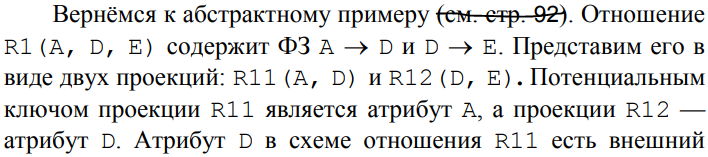


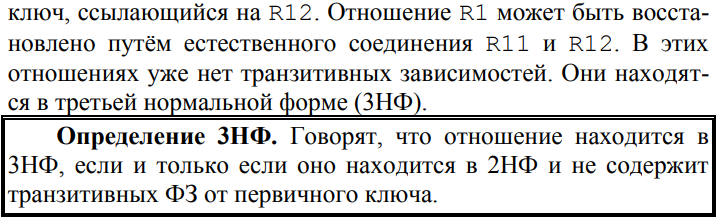


### 26.2.2 Вторая нормальная форма



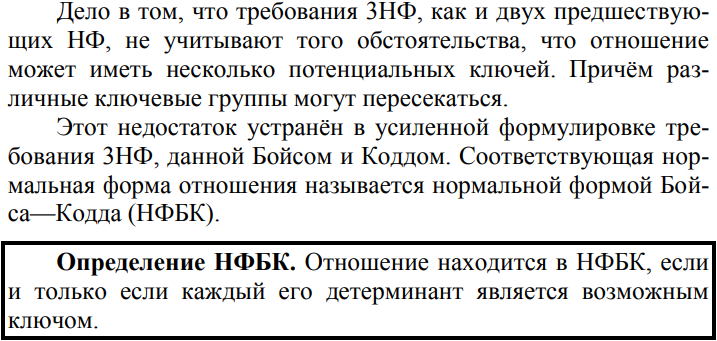
### 26.2.3 Третья нормальная форма



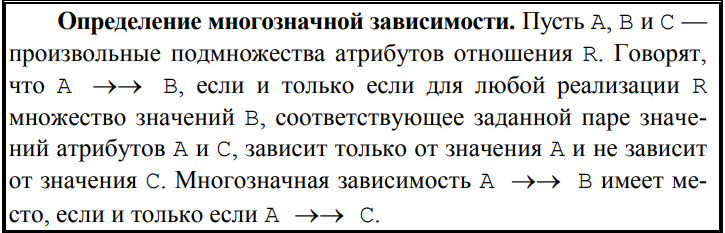


### 26.2.4 Нормальная форма Бойса-Кодда

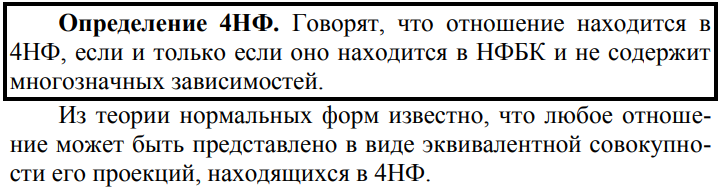
### 



## 26.3 Многозначная зависимость



## 26.4 Четвёртая нормальная форма



# 27. Нормализация отношения. Необходимость нормализации. Правила нормализации. Критерии выбора проекций. Требования к нормализованной структуре РБД.

## 27.1 Нормализация

Нормализация — это процедура декомпозиции без потерь, применяемая к «нежелательному» отношению в ходе проектирования базы данных.

## 27.2 Правила нормализации

Правило 1. Отношение в 1НФ следует разбить на проекции для исключения тех ФЗ от первичного ключа, которые не являются неприводимыми. Результатом будет набор отношений, находящихся, по крайней мере, в 2НФ.

Правило 2. Отношения в 2НФ следует разбить на проекции для исключения любых транзитивных зависимостей. Результатом будет набор отношений, находящихся, по крайней мере, в 3НФ.

*Совет*. Избавляясь от транзитивных ФЗ, создавайте проекции так, чтобы в них были представлены базовые ФЗ. Тогда транзитивные будут следствиями определений первичных и внешних ключей проекций.

Правило 3. Отношения в 3НФ следует разбить на проекции для исключения любых ФЗ, детерминанты которых не являются потенциальными ключами. Результатом декомпозиции будет набор отношений, находящихся, как минимум, в НФБК.

Вывод. Требования декомпозиции без потерь и декомпозиции на независимо обновляемые проекции могут конфликтовать. В некоторых случаях можно выполнить декомпозицию отношения 3НФ, не находящегося в НФБК, без потерь, но полученные проекции будут зависимыми по обновлению.

Правила 1 — 3 можно сконцентрировать в одном.

Правило нормализации до НФБК. Исходное отношение следует разбить на проекции, не содержащие ФЗ, детерминанты которых не являются возможными ключами.

Правило 4. Отношение, находящееся в НФБК, следует разбить на проекции для исключения всех многозначных зависимостей. Результатом будет набор отношений в 4НФ.

Из определений потенциальных ключей этих отношений следуют ФЗ 1 — 10 (см. стр. 105). Они будут поддерживаться механизмами РСУБД, если объявлены все потенциальные ключи. ФЗ 11 утеряна. Вася должен обеспечить её поддержку на уровне приложения куратора.

Важное замечание. На практике многозначные зависимости следует исключать до выполнения нормализации.

## 27.3 Критерии выбора проекций

а) декомпозиция без потерь;

б) возможность независимого обновления проекций.

Первый критерий удовлетворяется, если и только если исходное отношение содержит хотя бы одну ФЗ (теорема Хеза). На практике это требование выполняется для любой нетривиальной задачи.

Практическое правило декомпозиции без потерь: все атрибуты первичного ключа одной из проекций должны входить в состав атрибутов другой проекции.

Второй критерий удовлетворяется, если и только если все ФЗ, существующие в исходном отношении, являются логическими следствиями определений потенциальных ключей его проекций.

Любое отношение может быть нормализовано до 3НФ с сохранением ФЗ.

При нормализации до НФБК второй критерий может оказаться недостижимым. В этом случае следует выбрать одну из альтернатив:

– выполнить нормализацию и обеспечить процедурную поддержку утерянных ФЗ;

– включить в структуру БД ненормализованное отношение и обеспечить процедурную поддержку необъявленных ФЗ.

## 27.4 Требования к нормализованной РБД

Как мы видели на предыдущих лекциях, большую часть проблем поддержания зависимостей атрибутов можно возложить на РСУБД. Для этого нужно декомпозировать «нежелательное» (содержащее «не поддерживаемые» зависимости) отношение, представив его в виде эквивалентной совокупности его проекций.

Эта совокупность проекций должна удовлетворять двум требованиям.

1. Для любой реализации исходного отношения R естественное соединение проекций должно быть эквивалентно R (декомпозиция без потерь).

2. Все межатрибутные зависимости, существующие в исходном отношении, должны быть логическими следствиями определений потенциальных ключей проекций. отношении.

Если нарушается второе, то пр

Если нарушается первое требование, то естественное соединение проекций содержит кортежи, которых нет в исходномоекции не могут обновляться независимо

# 28. Модель IDEF1X. Назначение, основные понятия, нотации. Уровни модели. Примеры диаграмм различных уровней с интерпретацией.

**Модель данных** — графическое и текстуальное представление, идентифицирующее потребности организации в данных для достижения ее целей, выполнения функций и определения стратегий управления. Модель данных идентифицирует сущности, домены, атрибуты и связи между данными и поддерживает единый концептуальный взгляд на данные и их взаимосвязи.

**Глоссарий является обязательным дополнением.** Он содержит описания отдельных диаграмм модели и определения сущностей, атрибутов и доменов. Обязательные компоненты глоссария:

**– имена** — уникальные, осмысленные и соответствующие природе ПО наименования сущностей, атрибутов и доменов;

– **определения** — краткие, точные, однозначные тексты, обеспечивающие правильное понимание смысла имен, одинаковое для любого читателя.

**Цель моделирования** — На основании локальных представлений пользователей найти обобщенное представление информации, свойственное природе предметной области как целому.

В соответствии с этим определением IDEF1Х как язык описания данных содержит **два важнейших компонента**:

**– графический** — средства создания диаграмм, показывающих структуру и взаимосвязи данных;

**– текстовый** — правила создания и ведения текстовых документов, уточняющих и поясняющих графическую модель, — глоссариев, спецификаций, отчетов, заметок и т.п.

В IDEF1Х различают три уровня **графического представления** информации, или три уровня диаграмм.

**Уровень «сущность-связь» (ER level)**. Это уровень наименее детального представления информации. Он используется на начальной стадии моделирования, когда еще не выяснены или не поняты до конца свойства сущностей и связей. На диаграм- 121 мах ER-уровня сущности и связи представлены только их именами.

**Уровень ключей (Key-Based Level, KB)**. На этом уровне в диаграммах отображаются имена первичных и внешних ключей сущностей и спецификации связей. Диаграмма KB-уровня объявляет уникальные идентификаторы экземпляров сущностей и ограничения ссылочной целостности.

**Требования:**

* Различаются зависимые и независимые сущности, а также идентифицирующие/неидентифицирующие и обязательные/необязательные соединения.
* Неспецифические соединения запрещены.
* Каждая сущность должна иметь первичный ключ и альтернативные ключи, если они существуют.
* Каждая сущность должна содержать внешний ключ для каждого соединения или связи категоризации, в которой она участвует как потомок или категория.
* Каждый кластер категорий должен иметь дискриминатор.
* Диаграмма КВ-уровня может также содержать неключевые атрибуты.

**Уровень атрибутов (Fully Attributed Level, FA).** Диаграмма FA-уровня показывает имена всех атрибутов сущностей и связей и полностью определяет структуру и взаимосвязи данных. **Все сущности на FA-диаграмме должны находиться, по крайней мере, в 3НФ.**

**Результат моделирования:**

* **FA-диаграмма** - графическое представление структуры РБД содержащее полностью определённые схемы отношений.
* **Словарь данных** – набор определений имён сущностей, атрибутов, доменов и других элементов модели

**ER-уровень**

Диаграмма должна содержать сущности и связи, может показывать атрибуты и не должна показывать первичные, альтернативные или внешние ключи. На ER-уровне сущности не различаются как зависимые или независимые, а соединения — как идентифицирующие и неидентифицирующие. Сущности не содержат горизонтальных линий, отделяющих область ключей от области данных. Имена сущностей вписываются в обозначающие их прямоугольники. Диаграмма ER-уровня может показывать категории, но указывать дискриминаторы кластеров не обязательно. На ER уровне допустимы неспецифические соединения. Для изображения соединений можно использовать как сплошные, так и штриховые линии. Это не специфицирует соединения.

**КВ-уровень**

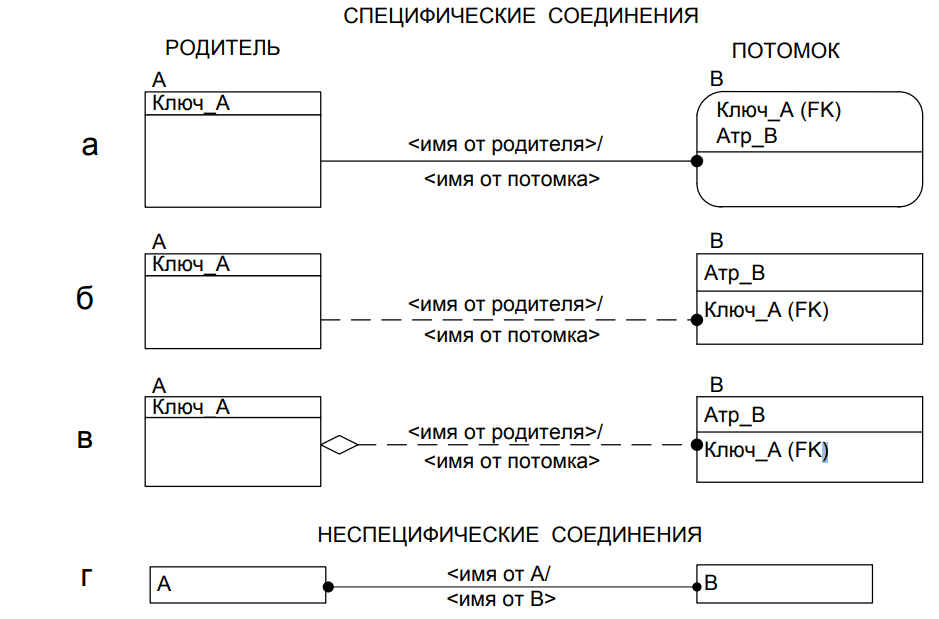
Диаграммы этого уровня изображают сущности, связи, первичные, альтернативные и внешние ключи. Различаются зависимые и независимые сущности, а также идентифицирующие/неидентифицирующие и обязательные/необязательные соединения. Неспецифические соединения запрещены. Каждая сущность должна иметь первичный ключ и альтернативные ключи, если они существуют. Каждая сущность должна содержать внешний ключ для каждого соединения или связи категоризации, в которой она участвует как потомок или категория. Каждый кластер категорий должен иметь дискриминатор. Диаграмма КВ-уровня может также содержать неключевые атрибуты.

**FA-уровень**

Диаграмма FA-уровня должна содержать все, что содержит диаграмма КВ-уровня и, кроме того, все неключевые атрибуты. На КВ- и FA-уровнях в полной мере действуют все правила синтаксиса, изложенные выше. Все сущности на FA-диаграмме должны находиться, по крайней мере, в 3НФ.

**Нотации**

****

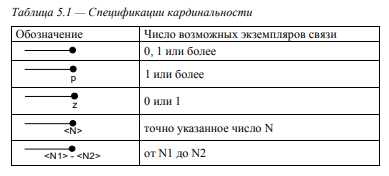
****

а — идентифицирующее соединение;

б — обязательное неидентифицирующее соединение;

в — необязательное неидентифицирующее соединение;

г — неспецифическое соединение



# 29. Модель IDEF1X. Понятие сущности. Сходства и различия понятий сущности в ER-модели и модели IDEF1X. Типы сущностей в модели IDEF1X. Изображения сущностей различных типов на диаграммах. Правила именования сущностей.

**Модель данных** — графическое и текстуальное представление, идентифицирующее потребности организации в данных для достижения ее целей, выполнения функций и определения стратегий управления. Модель данных идентифицирует сущности, домены, атрибуты и связи между данными и поддерживает единый концептуальный взгляд на данные и их взаимосвязи.

**Сущность (entity) в ER-модели** — это некоторый объект, выделяемый (идентифицируемый) пользователем в предметной области. Нечто, за чем пользователь хотел бы наблюдать и сохранять результаты наблюдений (данные)

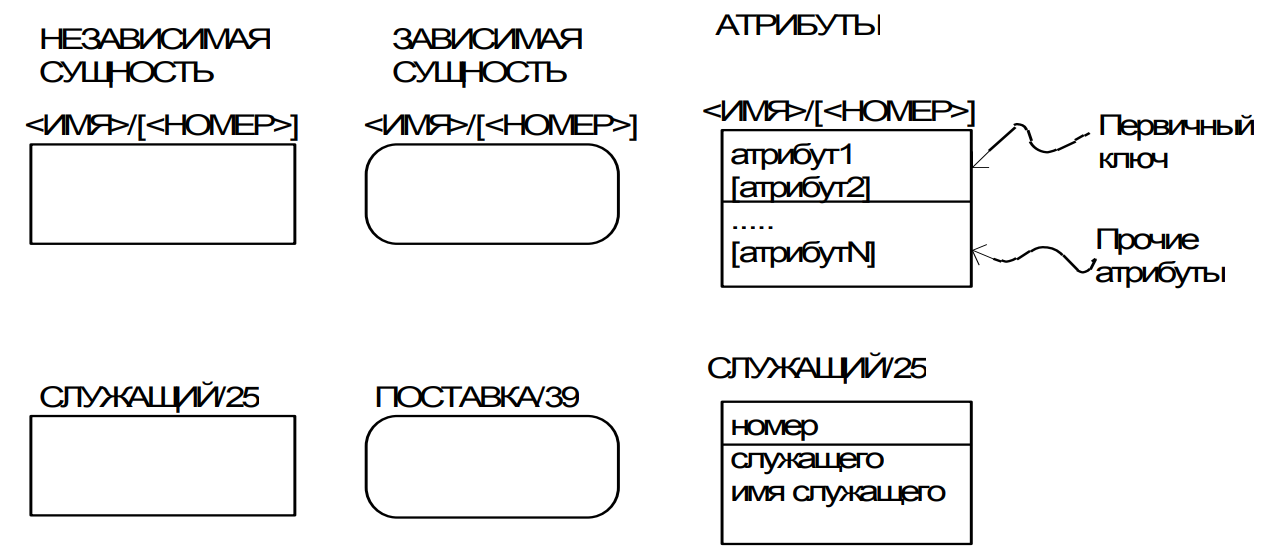
**Под сущностью в IDEF1Х** понимается отношение *РМД*. Сущность в IDEF1X описывает собой совокупность или набор экземпляров похожих по свойствам, но однозначно отличаемых друх от друга по одному или нескольким признакам. Каждый экземпляр является реализацией сущности.

**Типы сущностей**

Сущность является **независимой**, если каждый ее экземпляр может быть однозначно идентифицирован без определения его связей с другими сущностями.

Сущность называется **зависимой**, если однозначная идентификация ее экземпляра зависит от его связей с другими сущностями.

Сущность может обладать атрибутами, которые наследуются через связь с родительской сущностью. Последние обычно являются **внешними ключами** и служат для организации связей между сущностями. Если **внешний ключ** сущности используется в качестве ее первичного ключа или как часть составного первичного ключа, то сущность является зависимой от родительской сущности. Если внешний ключ не является первичным и не входит в составной первичный ключ, то сущность является независимой от родительской сущности.



**На уровне ER зависимые и независимые сущности не различаются.**

**Правила именования сущностей:**

* Сущности, атрибуты и домены обязательно именуются. Именем может быть только имя существительное, возможно с определениями. В качестве имён допускаются аббревиатуры и акронимы.
* Имя должно быть уникальным и осмысленным. Формальное определение имени обязательно включается в глоссарий.
* Имя сущности, атрибута или домена должно иметь единственный смысл и этот смысл всегда должен выражаться этим именем. Тот же смысл не может вкладываться в другое имя, если оно не является псевдонимом или синонимом основного
* Сущности и атрибуты всегда именуются в единственном числе. Имя должно относиться к одному экземпляру сущности или одному значению атрибута.

# 30. Модель IDEF1X. Понятие соединения. Типы соединений. Изображения соединений различных типов на диаграммах. Правила именования соединений. Маркировка свойств соединений.

**Соединение** — это один из двух видов связей, используемых в языке IDEF1X. Стандарт определяет соединение как ассоциацию между двумя сущностями или между экземплярами одной и той же сущности. Понятие соединения в IDEF1X совпадает с понятием бинарной связи в ER-модели.

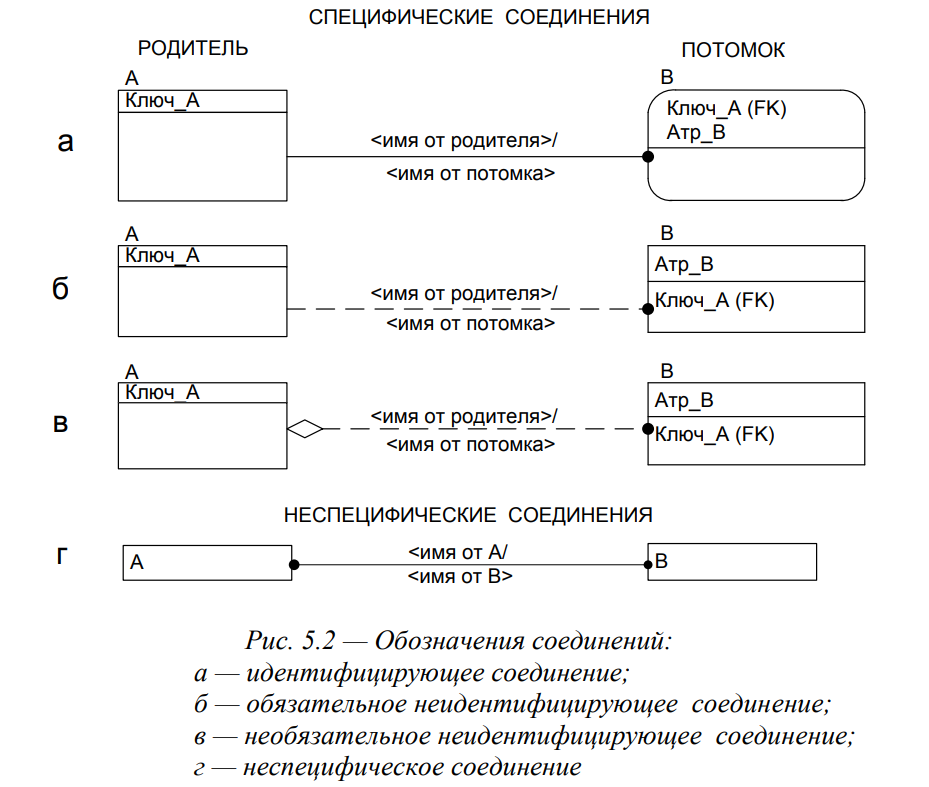
**Типы соединений**

**Неспецифическое соединение** — это бинарная связь типа М:N В такой связи нет родителя и потомка. Неспецифические соединения могут быть показаны только на диаграммах ERуровня. На KB- и FA-диаграммах они должны быть представлены эквивалентными парами специфических соединений.

**Специфическое соединение** — это бинарная связь типа 1:М. В такой связи всегда можно указать родителя и потомка. Родителем называется многосвязная сущность. Мощность специфического соединения со стороны потомка 0 или 1.

**Специфическое соединение называется идентифицирующим**, если соответствующий ему внешний ключ полностью входит в первичный ключ потомка. Это означает, что идентификация экземпляров потомка в идентифицирующем соединении возможна только через ссылку на соответствующий экземпляр родителя.

**Соединение называется неидентифицирующим**, если соответствующий ему внешний ключ не входит в состав первичного ключа потомка или входит в него частично.



**Правила именования соединений:**

* Соединению присваивается имя, выражаемое глагольным оборотом. Имя зрительно привязывается к дуге, изображающей соединение. Имена соединений могут быть неуникальными в пределах диаграммы.
* Имя каждого соединения одной и той же пары сущностей должно быть уникальным во множестве имен связей этих сущностей.
* Имена специфических соединений выбираются так, чтобы можно было построить осмысленную фразу, составленную из имени родительской сущности, имени связи, выражения кардинальности и имени потомка
* Связь может быть поименована «от родителя» и от «потомка». Имя «от родителя» обязательно.
* Если связь не именуется со стороны потомка, то имя «от родителя» должно выбираться так, чтобы связь легко читалась и со стороны потомка
* Неспецифические соединения обязательно именуются с обеих сторон.

# 31. Модель IDEF1X. Понятие атрибута. Сходства и различия понятий атрибута в ER-модели и модели IDEF1X. Правила именования атрибутов. Понятия первичного, альтернативного и внешнего ключей. Представление ключей на диаграммах.

Атрибут – это свойство сущности или связи. Идентифицируется именем существительным. Принимает конкретное значение для экземпляра сущности. Также атрибут может быть, в зависимости от точки зрения пользователя:

· простым или составным (композитным),

· однозначным или многозначным,

· первичным или производным

**Сходства и различия понятий атрибута в ER-модели и модели IDEF1X**

На ER - уровне допустимо отображать атрибуты сущностей, но они не являются ключевыми\не ключевыми.

Для FA-уровня обязательно разграничение атрибутов на ключевые (первичные, внешние и альтернативные ключи) и неключевые.

**Правила именования атрибутов:**

* На диаграммах KB- и FA-уровней каждая сущность имеет не менее одного атрибута.
* Каждый атрибут может быть собственным атрибутом сущности или присоединённым, полученным от другой сущности через связь (внешний ключ).
* Каждый атрибут является собственным атрибутом точно одной сущности.
* Присоединённый атрибут должен быть частью первичного ключа передавшей его родительской (или родовой) сущности.
* Присоединённый атрибут помечается символом “(FK)”, следующим за именем атрибута.
* Каждый экземпляр сущности должен иметь определённое (не NULL) значение каждого атрибута, являющегося частью первичного ключа.
* Не может быть экземпляра сущности, имеющего более чем одно значение какого-либо из атрибутов.
* На KB-диаграммах показываются только атрибуты, входящие в состав первичных, альтернативных и внешних ключей.
* Атрибуты альтернативных ключей обязательно помечаются символом “(AKn)”, где n - номер альтернативного ключа.
* На FA-диаграмме показываются все атрибуты каждой сущности.

**Первичный ключ** - это **атрибут** или набор **атрибутов**, выбранных для идентификации уникальных экземпляров сущности.

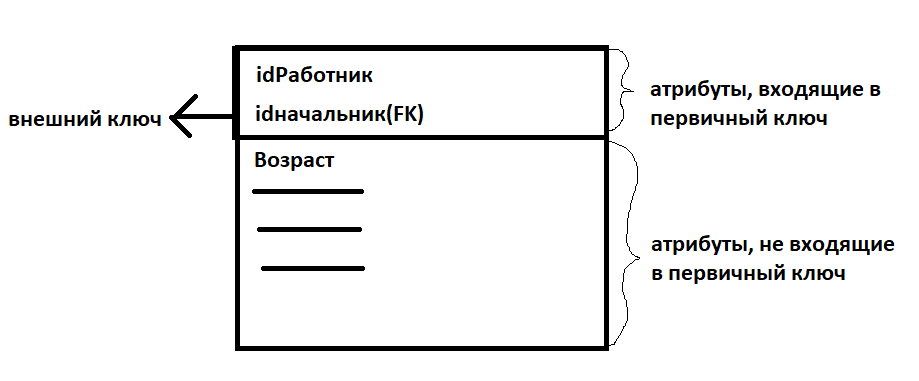
**Свойства первичного ключа:**

* однозначная идентификация записи – запись должна однозначно определяться значением ключа;
* отсутствие избыточности – никакое поле нельзя удалить из ключа, не нарушая при этом свойства однозначной идентификации записи.

**Внешний ключ** — это атрибут или набор атрибутов одной сущности, ссылающиеся на первичный ключ в другой сущности. Предназначен для связывания двух сущностей друг с другом.

**Альтернативный ключ** - потенциальный ключ, не ставший первичным.

**Потенциальный ключ** - атрибут ( или набор атрибутов), который может быть использован в качестве первичного ключа.



# 32 Модель IDEF1X. Понятия категории сущности и связи категоризации. Понятия категории, кластера категорий и дискриминатора кластера. Изображение связей категоризации на диаграмме.

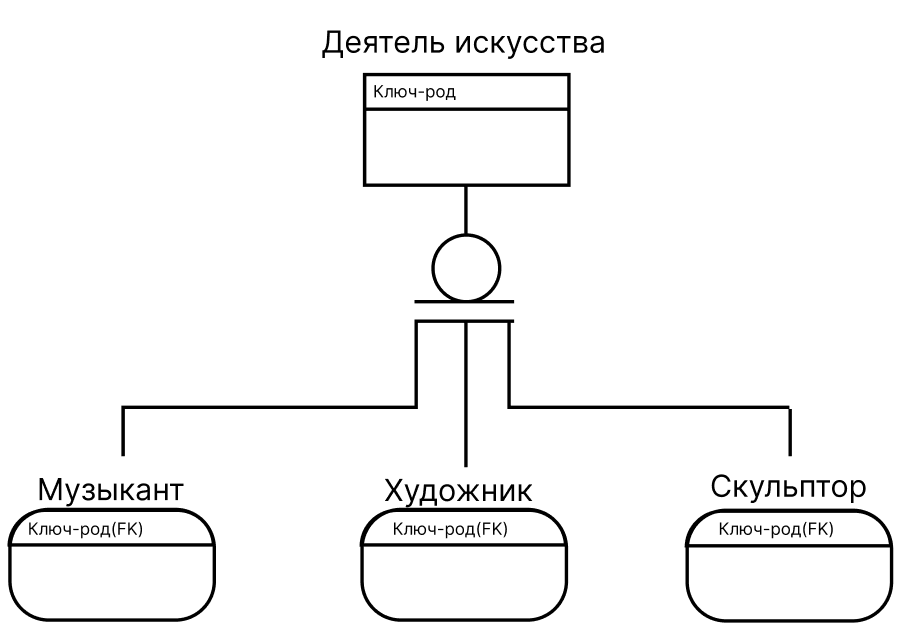
**Отношения категоризации** – отношения между двумя и более сущностями, в которых каждый экземпляр одной сущности, называемой общей, связан в точности с одним экземпляром сущности, называемой сущностью-категорией. Категория выделяется из общей сущности по определенному признаку. Различают полную и неполную категоризацию .

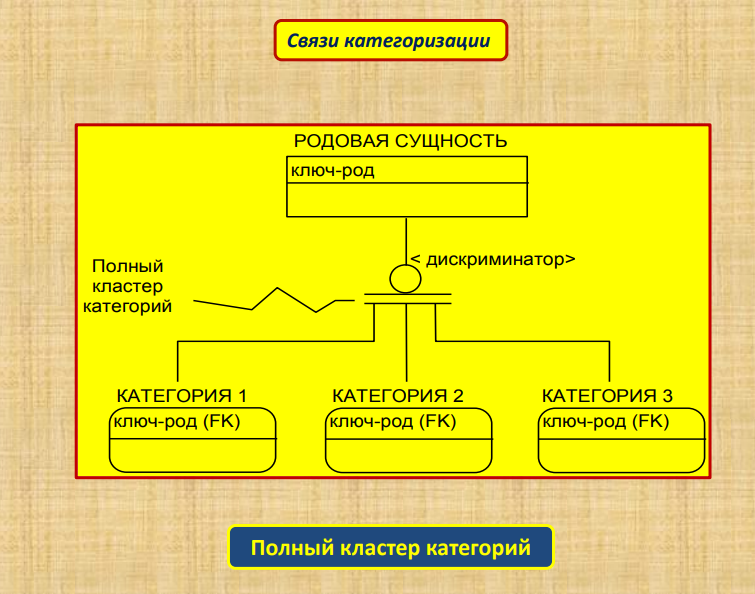
**«Кластер категории»** - это множество, состоящее из одной или более связей категоризации. Экземпляр генеральной сущности может быть связан только с одним экземпляром категории сущности в кластере, и каждый экземпляр категории сущности связан только с одним экземпляром генеральной сущности. Каждый экземпляр категории сущности представляет то же объект, что и связанный с ним экземпляр генеральной сущности.

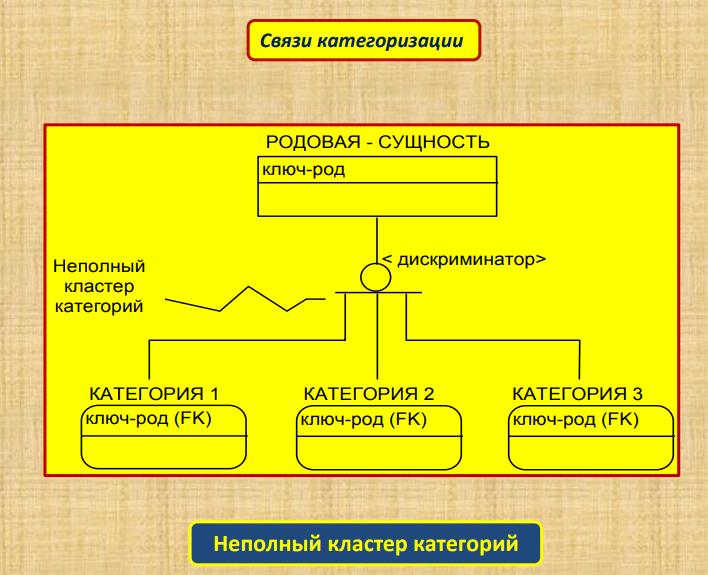
**Дискриминатор** - атрибут родовой сущности, позволяющий отнести экземпляр родовой сущности к одной из дочерних категорий.

**Родовая сущность** - сущность, содержащая в себе общие атрибуты для каждой категории

**Категория** - сущность, имеющая более конкретизированные ат

Пример:



****

**Правила для связей категоризации:**

* Категория может иметь только одну родовую сущность. Она может принадлежать только одному кластеру категорий.
* Категория в одной категоризационной связи может быть родовой сущностью в другой категоризационной связи.
* Сущность может быть родовой для любого числа кластеров категорий
* Все экземпляры категории имеют одно и то же значение дискриминатора, и все экземпляры различных категорий должны иметь различные значения дискриминатора.
* Не существует двух кластеров категорий одной и той же родовой сущности, имеющих один и тот же дискриминатор.
* Дискриминатором полного кластера категорий не может быть атрибут, значения которого могут быть неопределёнными.
* Первичный ключ любой категории должен совпадать с первичным ключом родовой сущности. Однако в качестве имён ключевых атрибутов категорий могут использоваться имена ролей.

ПОЗДРАВЛЯЮ, ВЫ ПРОШЛИ ИГРУ!