1. **Понятие базы данных, СУБД, виды БД.**

Под информацией будем понимать абсолютно любые сведения о каком-либо событии, явлении, процессе или сущности, характерные для рассматриваемой области.

Данные – информация, зафиксированная в определенной форме, и пригодная для последующего хранения, обработки и передачи.

Информационный процесс - сбор, хранение, преобразование и удаление информации о предметной области.

Информационная система может быть определена как вычислительная среда, обеспечивающая выполнение информационного процесса.

По целевому назначению можно разделить все информационные системы на:

1. Управляющие.

2. Информационно-справочные.

3. Системы поддержки принятия решений.

**База данных** – совокупность специальным образом организованных данных, хранимых в памяти вычислительной системы, и отображающих состояние объектов их взаимосвязей в рассматриваемой предметной области.

**СУБД** (система управления базами данных) – специальное ПО, с помощью которого пользователи могут определять, создавать и поддерживать базу данных, а также осуществлять к ней контролируемый доступ.

**Классификация баз данных (по модели данных):**

1. Иерархическая (на основе деревьев)
2. Сетевая (на основе графов)
3. Реляционная (на основе таблиц)
4. Постреляционная (таблицы с возможностью вложения одних таблиц в другие)
5. Объектно-ориентированная модель (на основе принципов ООП)
6. **Определение/Функции СУБД.**

* **Система управления базами данных (СУБД)** – это совокупность языковых и программных средств, предназначенных для создания, ведения и совместного использования БД многими пользователями.
* **Назначение (Функции) СУБД**:

1) Непосредственное управление данными во внешней памяти. Эта функция предполагает создание необходимых структур внешней памяти, как для хранения данных, непосредственно входящих в БД, так и для служебных целей.

2) Управление буферами оперативной памяти.

3) Управление транзакциями. Транзакция – последовательность операций над БД, рассматриваемых СУБД как единое целое. При использовании транзакций возможны лишь два исхода: либо транзакция успешно выполняется и СУБД фиксирует во внешней памяти изменения БД, произведенные этой транзакцией, либо ни одно из действий, произведенных транзакцией, никак не изменяет состояние БД. Понятие транзакции необходимо поддержание логической целостности БД. <…> Делает очень удобным использование понятия транзакции как единица активности пользователя по отношению к БД.

4) Журнализация. Одним из основных требований к СУБД является надежность хранения данных во внешней памяти. Под надежностью хранения понимается то, что СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого программного или аппаратного сбоя. Сбои могут быть мягкими и жесткими. Для восстановления БД нужно располагать некоторой дополнительной информацией, то есть поддержание надежности хранения данных требует некоторой избыточности хранения данных, причем та часть, которая используется для восстановления должна храниться особенно надежно. Наиболее распространенным методом поддержания такой избыточной информации является ведение журнала изменений в БД. Журнал — это особая часть БД, недоступная пользователям СУБД и поддерживаемая с особой тщательностью, в которую поступаю записи обо всех изменениях основной части БД.

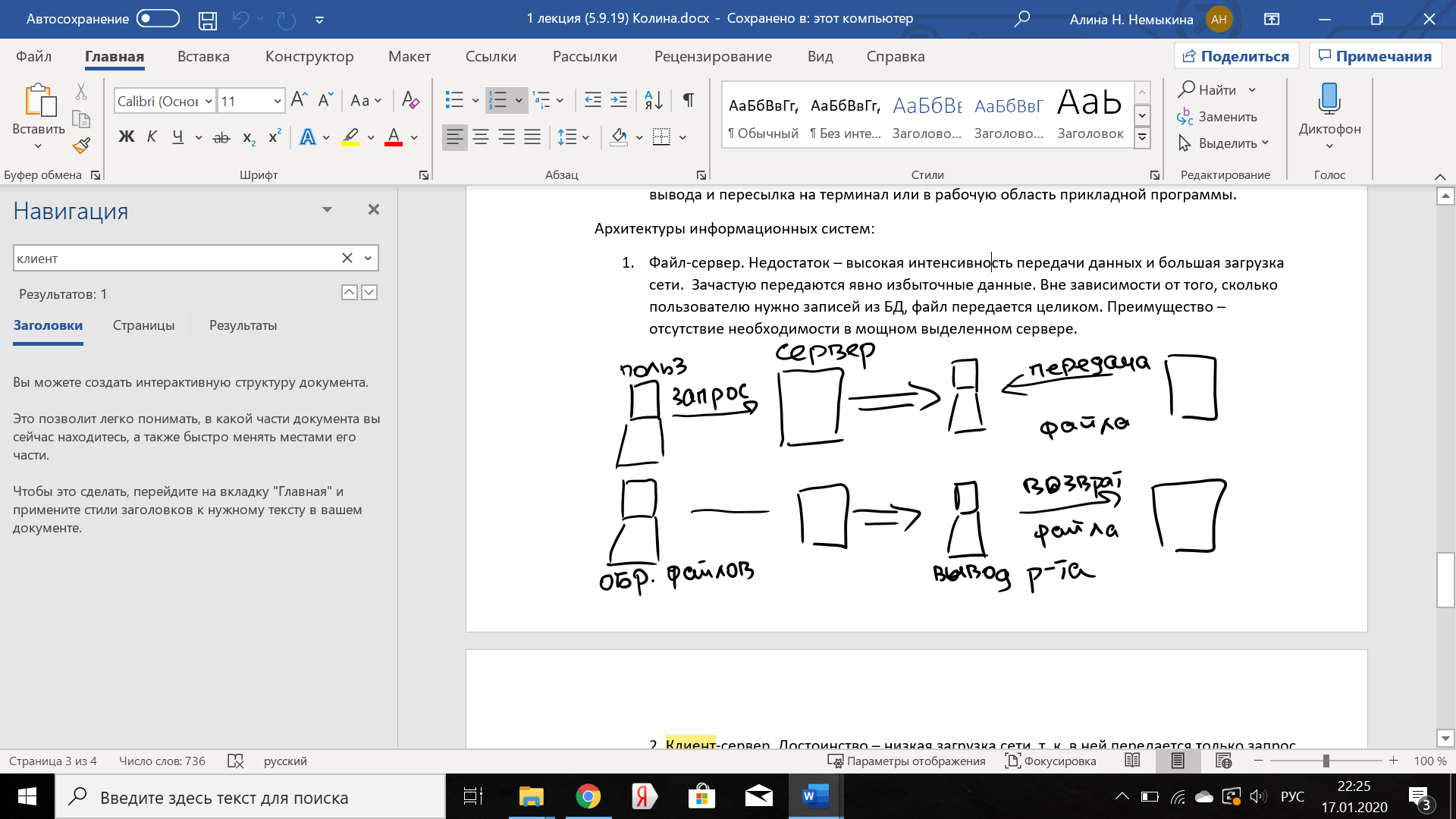
5) Поддержка языков БД. Для работы с БД используются специальные языки: язык определения схемы БД и язык манипулирования данными. SQL

6) Поддержка словаря данных. Словарь данных является сам по себе является БД и содержит, так называемые, метаданные, т.е. определение объектов системы, их свойств и отношений между ними для данной конкретной предметной области.

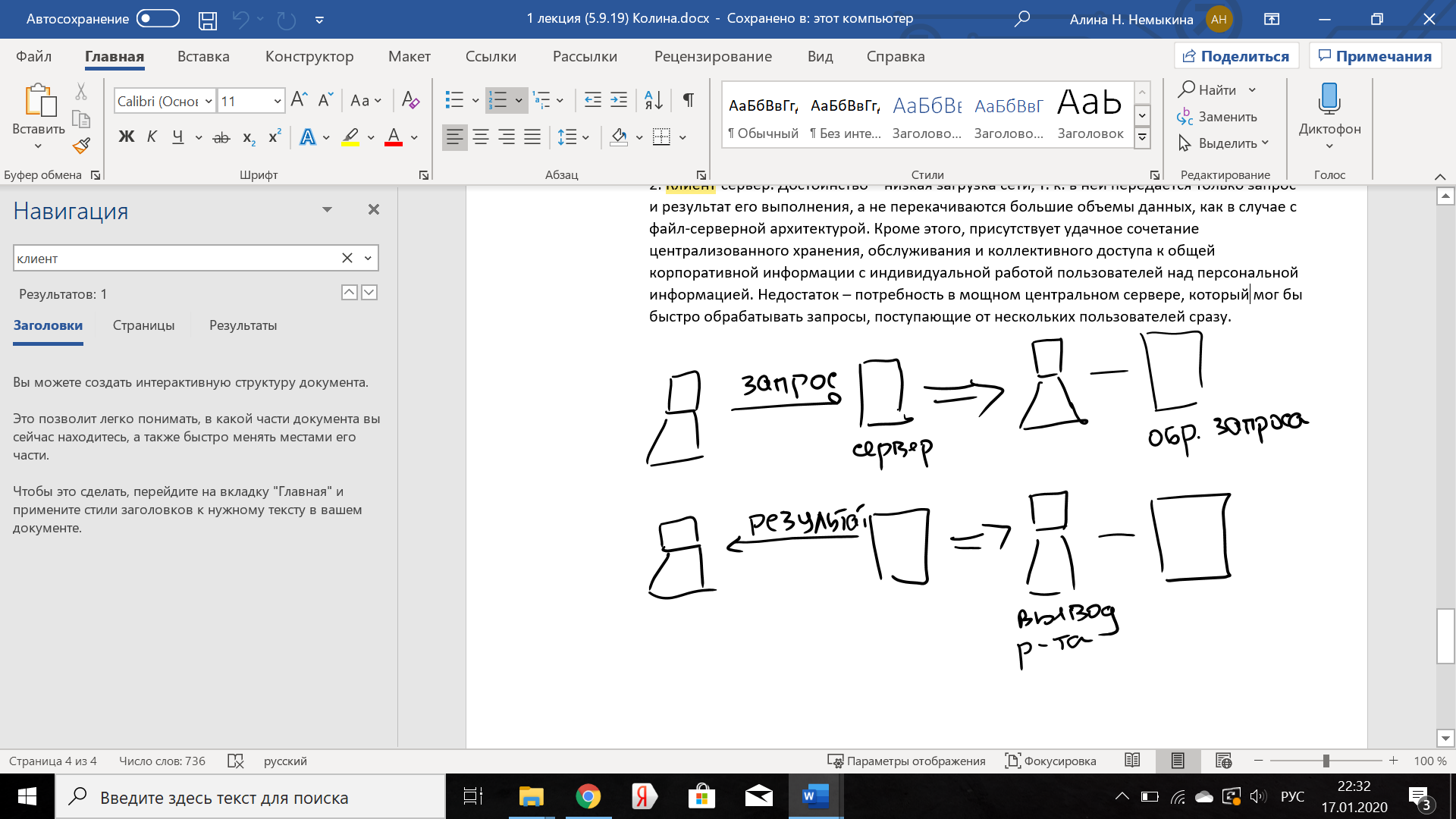
1. **Архитектура БД. Файловые/клиент-серверные БД.**

Архитектуры информационных систем:

1. Файл-сервер. По запросу осуществляется передача данных. Недостаток – высокая интенсивность передачи данных и большая загрузка сети. Зачастую передаются явно избыточные данные. Вне зависимости от того, сколько пользователю нужно записей из БД, файл передается целиком. Преимущество – отсутствие необходимости в мощном выделенном сервере *(сейчас не актуально)*.



1. Клиент-сервер. По запросу от клиента серверу выполняется обработка на сервере, результат отсылается обратно и выводится на экран. **Достоинство** – низкая загрузка сети, т. к. по ней передается только запрос и результат его выполнения, а не перекачиваются большие объемы данных, как в случае с файл-серверной архитектурой. Кроме этого, присутствует удачное сочетание централизованного хранения, обслуживания и коллективного доступа к общей корпоративной информации с индивидуальной работой пользователей над персональной информацией. Недостаток – потребность в мощном центральном сервере, который мог бы быстро обрабатывать запросы, поступающие от нескольких пользователей сразу.



1. **Инфологическое проектирование БД. Диаграмма «Сущность-Связь». Сущность.**

Инфологический аспект употребляется при рассмотрении вопросов, связанных со смысловым содержанием данных независимо от способов их представления в памяти системы. На этапе инфологического проектирования информационной системы выделяется часть реального мира, определяющая потребность информационной системы, т.е., ее предметная область. При этом решаются следующие вопросы:

- о каких объектах или явлениях реального мира требуется накапливать и обрабатывать информацию в системе.

- какие их основные характеристики и взаимосвязи между собой будут учитываться.

- уточняются все вводимые в информационную систему понятия об объектах, явлениях, их характеристиках и взаимосвязях.

Модель сущность-связь - это неформальная модель предметной области, которая работает на этапе инфологического проектирования СУБД. Она определяет сущности в информационной системе и их отношения между собой. Основным назначением этой модели является семантическое описание предметной области и представление информации для обоснования выбора видов моделей и структур данных. Понятия: сущность, атрибут, связь.

Сущность - это собирательное понятие, некоторая абстракция реально существующего объекта, процесса или явления, о котором необходимо хранить информацию в системе (теоретически - любое существительное). Существует тип сущности и экземпляр сущности, по аналогии с ООП.

По типу зависимости выделяют \*сильные\* и \*слабые\* сущности. Слабая сущность - сущность, существование которой зависит от какой-то другой сущности. Сильная сущность - напротив не зависит от существования других сущностей. На диаграмме ER (Entity-Relation) будем изображать сильные сущности в прямоугольнике, а слабые - в двойном прямоугольнике.

1. **Инфологическое проектирование БД. Атрибут. Виды атрибутов. Ключевые атрибуты.**

На этапе инфологического проектирования информационной системы выделяется часть реального мира, определяющая потребность информационной системы, т.е., ее предметная область. При этом решаются следующие вопросы:

* о каких объектах или явлениях реального мира требуется накапливать и обрабатывать информацию в системе.
* какие их основные характеристики и взаимосвязи между собой будут учитываться.
* уточняются все вводимые в информационную систему понятия об объектах, явлениях, их характеристиках и взаимосвязях.

**Атрибут** - это поименованная характеристика сущности, которая принимает значение из некоторого множества значений. Набор потенциально допустимых значений атрибута называют доменом атрибута.

**Простой атрибут** - атрибут, состоящий из одного компонента с независимым существованием.

**Составной атрибут** - это атрибут, состоящий из нескольких компонентов, каждый из которых характеризуется независимым существованием. Например, адрес: дом, улица, квартира, город и т. д.

**Однозначный атрибут** - атрибут, который для данной конкретной сущности может принимать только одно значение.

**Многозначный атрибут** - атрибут, который для конкретной сущности может принимать несколько значений (например, несколько телефонов).

**Производный атрибут** - атрибут, который может принимать значение, производное от значения связанного с ним атрибута (например, возраст, который зависит от даты рождения).

**Ключ** - элемент данных, позволяющих уникально идентифицировать отдельные экземпляры некоторой сущности.

**Потенциальный ключ** - это атрибут, или набор атрибутов, который уникально идентифицирует отдельные экземпляры сущности.

**Составной ключ** - это потенциальный ключ, который состоит из двух или более атрибутов. Например, рекламное объявление - издание, в котором оно помещено, дата выхода газеты и стоимость.

**Первичный ключ** - это потенциальный ключ, который выбран в качестве первичного ключа.

1. **Инфологическое проектирование БД. Связь. Показатель кардинальности.**

**Инфологический аспект** употребляется при рассмотрении вопросов, связанных со смысловым содержанием данных, независимо от способа их представления в памяти системы. На этапе инфологического проектирования информационной системы выделяется часть реального мира, определяющая информационную потребность системы, т.е. её предметную область. При этом решаются следующие вопросы:

1) О каких объектах или явлениях реального мира требуется накапливать и обрабатывать информацию в системе.

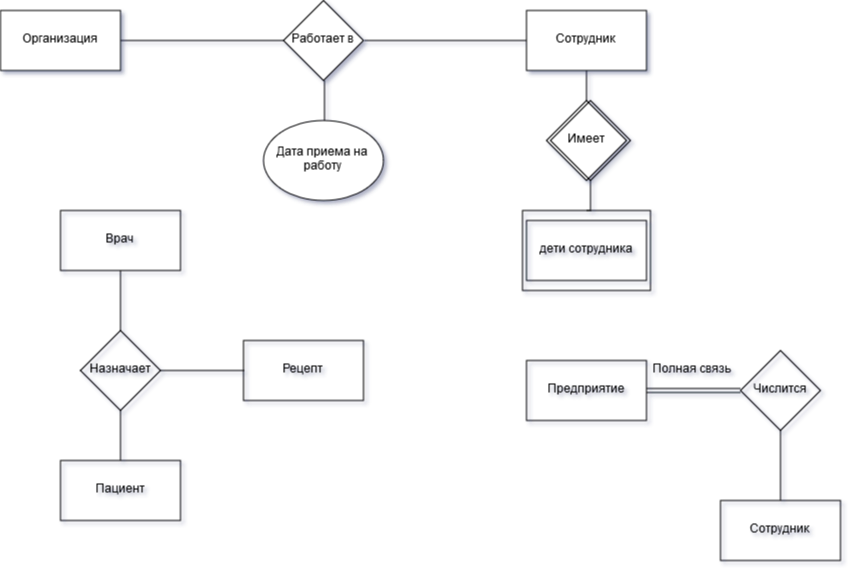
2) Какие их основные характеристики и взаимосвязи между собой будут учитываться.

3) Уточняются все вводимые в информационную систему понятия об объектах, явлениях, их характеристиках и взаимосвязях.

**Связь** – это осмысленная ассоциация между несколькими сущностями. В модели связи выступают в качестве средства, с помощью которого предоставляется отношение между сущностями, имеющими место в рассматриваемой предметной области.

Степень участия сущности в связи определяет, зависит ли существование некоторой сущности от существования в связи некоторой другой сущности. Степень участия является полной, если для существования некоторой сущности требуется существование другой сущности, связанной с ней определённым отношением. В противном случае связь является частичной.

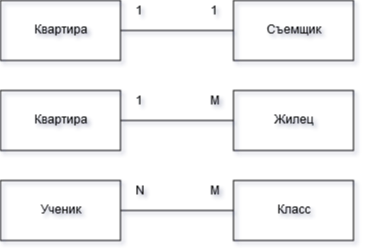
(двойная рамка - слабая сущность, второй рисунок - тернарная связь, двойная связь - полная)



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

Могут быть показатели:

* 1:1
* 1:M
* M:N



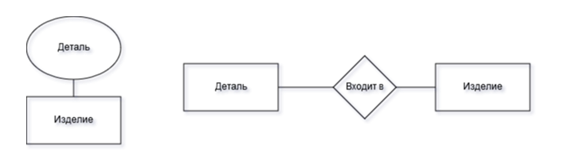
1. **Моделирование локальных представлений. Способы объединения моделей локальных представлений.**

****

Методика построения:

1.Выбор границ локального построения. Определение задач, соответствующих некоторому кругу потребностей пользователя. Необходимо с одной стороны избежать дробления данных, а с другой стороны не допустить чрезмерного раздувания предметной области для пользователя.

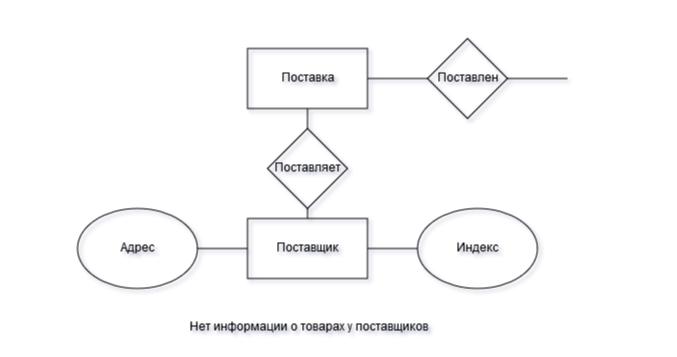
2.Формулирование сущности. Пример различных вариантов формулирования сущности:



Желательно рассмотреть все варианты и выбрать тот, в котором будет легче получать не только объекты, но и отдельные части информации. Пример улучшения схемы:



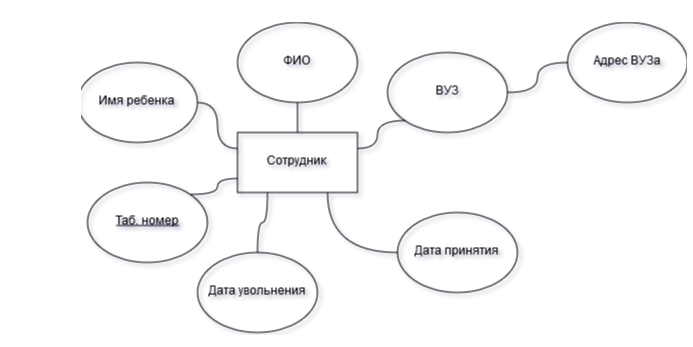


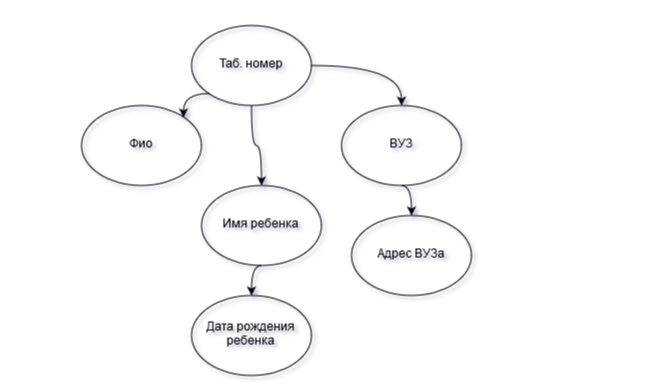


3.Выбор идентифицирующего атрибута. Для каждой выделенной сущности нужно указать идентификатор, служащий для однозначного распознавания отдельных экземпляров сущности. Возможны такие случаи, когда экземпляры сущности могут отличаться не только по атрибутам, но и по связям с другими сущностями.

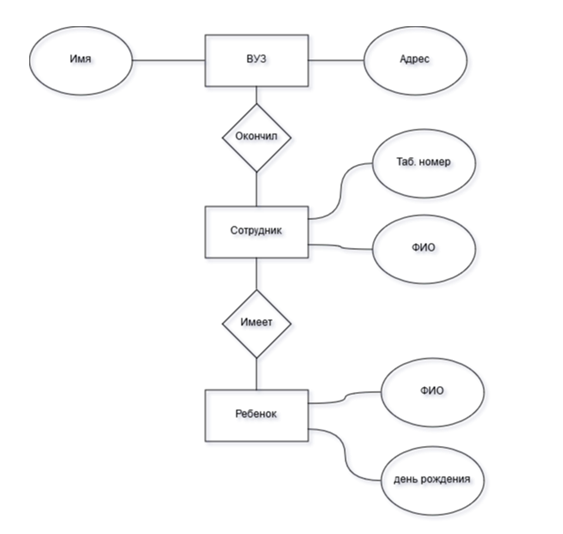
4.Назначение сущностям отрицательных атрибутов. Спецификация каждого атрибута будет включать в себя не только семантическое описание, но и описание домена (возможного набора значений)

5.Спецификация связей. Связь может быть не только между сущностями, но и между атрибутами. На данном шаге выявляются атрибуты, находящиеся в транзитивном отношении. Говорят, что три атрибута A1, A2, A3 находятся в транзитивном отношении, когда A1 однозначно определяет A2, а A2 однозначно определяет A3.





Между атрибутами этой сущности имеют место определенные связи. Так как все атрибуты описывают одну и ту же сущность "сотрудник", которая имеет ключевой атрибут "табельный номер", то все атрибуты имеют зависимость от данного атрибута. Это означает, что при описании конкретных экземпляров сущности "сотрудник" описательные атрибуты не могут принимать произвольные значения, а зависят от ключевого атрибута. Кроме этого, между атрибутами наблюдается ряд транзитивных зависимостей. При моделировании сущностей надо всегда стремиться к случаю, когда все описательные атрибуты зависят лишь от ключевого, т.е., избегать транзитивных зависимостей.



6.Объединение моделей локальных представлений. При объединении локальных представлений используется три приема:

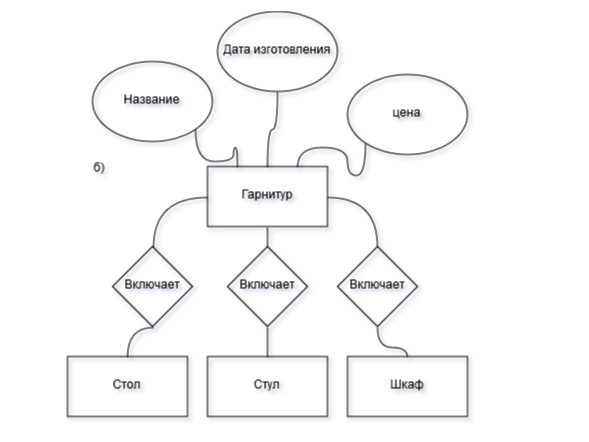
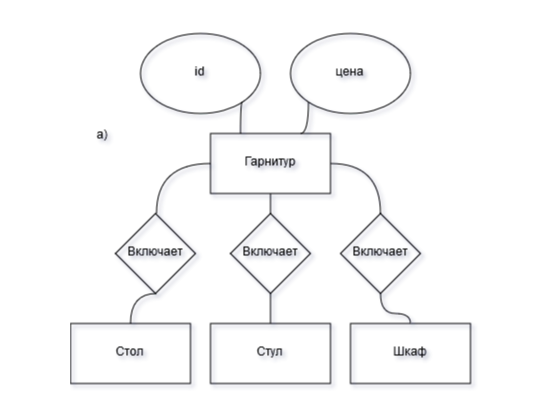
oИдентичность. Два и более элементов идентичны, если имеют одинаковое семантическое (смысловое) значение.

oАгрегация. Позволяет рассматривать связь между элементами модели как новый элемент. При объединении локальных представлений агрегация встречается в трех формах:

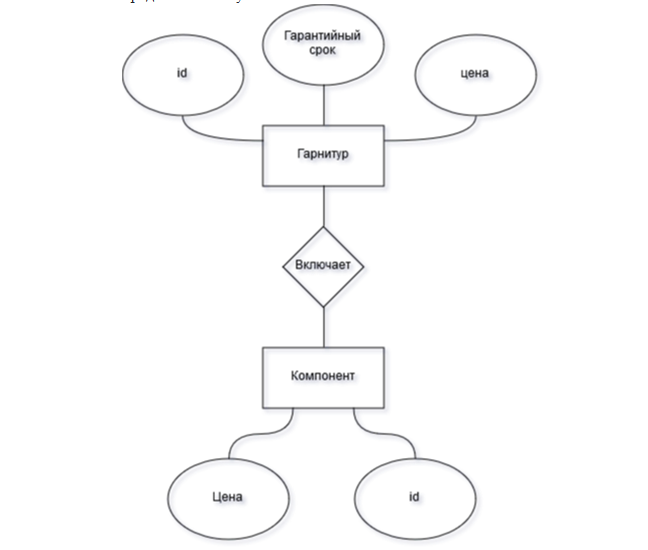
a.В одном представлении агрегатный объект определен как единое целое, а во втором рассматриваются его составные части.

b.Агрегатный объект как единое целое не определен ни в одном из представлений, но в обоих представлениях рассматриваются его составные части.

c.Один и тот же агрегатный элемент рассматривается в обоих представлениях, но составные части его различаются.



oОбобщение. Обобщением называется абстракция данных, позволяющая трактовать класс различных подобных типов объектов как один поименованный обобщенный объект. В обобщении подчеркивается общая природа объектов. Использование обобщения позволяет строить многоуровневые иерархии элементов, в которых элементы низлежащего уровня наследуют все характеристики своего предшественника. При этом потомок может иметь новые черты, не принадлежащие предшественнику.



1. **Реляционная модель данных. Атрибуты, домены, кортежи, отношения.**

**Реляционная модель** представляет собой средство описания данных на основе только их естественной структуры, то есть без потребностей введения каких-либо дополнительных структур для целей машинного представления (представление данных независимо от способа их физической организации).

В реляционной модели выделяют три части: структурная, манипуляционная, целостная.

**Атрибут** - именованный атомарный (неделимый) элемент данных.

**Домен** - множество допустимых значений атрибута, который определен на данном домене.

При этом допустимы одинаковые домены для различных атрибутов. Сравнивать между собой можно только значения атрибутов, определенных на одинаковых доменах.

В большинстве реляционных СуБД объект-домен не используется, но для каждого атрибута задается базовый тип и ряд ограничений, которые проверяются для каждого значения атрибута.

Схема отношения - именованное множество упорядоченных пар (имя атрибута : имя домена). (Схема базы данных - набор именованных схем-отношений)

Степень или n-арность схемы отношения - это мощность этого множества (схема отношений)

**Отношение -** множество кортежей, соответствующее одной схеме отношения.

**Кортеж -** множество упорядоченных пар (имя атрибута : значение атрибута), которое содержит по одному вхождению каждого имени атрибута, принадлежащего схеме отношения. Арность кортежа является совпадает с арностью соответствующей схемы отношения.

**Свойства отношения:**

* В отношении не может быть двух одинаковых кортежей
* Кортежи не упорядочены. атрибуты так же не упорядочены
* Имена всех атрибутов в пределах одного отношения должны быть уникальны

**Реляционная БД -** набор отношений, имена которых совпадают с именами схем отношений в схеме БД.

|  |  |
| --- | --- |
|  | SQL |
| Отношение | Таблица |
| Кортеж | Строка |
| Атрибут | Столбец |
| Домен, на котором определен атрибут | Множество допустимых значений столбца |
| Степень/арность | Кол-во столбцов |
| Кардинальное число | Кол-во строк |

1. **Реляционная модель данных. Потенциальные ключи. Первичный ключ. Внешний ключ.**

*В Реляционной модели достигается более высокий уровень абстракции данных, чем в иерархической или сетевой.* Реляционная модель *(на основе таблиц)* представляет средство описания данных на основе только их структуры, то есть без потребности введения каких-либо дополнительных структур для целей машинного представления, то есть представление данных не зависит от способа их физической реализации.

*Термин «реляционный» означает, что теория основана на математическом понятии* [*отношение*](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BD%D0%BE%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_(%D1%82%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2)) *(relation).*

*Таблица (общепринятый термин) -> отношения(в формализованном виде)(реляционный термин)*

*Строка -> кортеж*

*Столбец -> атрибут*

*Множество допустимых значений столбца -> домен*

*Количество столбцов -> степень/арность*

*Количество строк -> кардинальное число*

*Реляционная модель:*

*1. Структурная*

*2. Манипуляционная*

*3. Целостная*

*Атрибут определяется как именованный атомарный элемент данных. Множество допустимых значений атрибута – домен, на котором определён данный атрибут. При этом допустимы одинаковые домены для различных атрибутов. Сравнивать между собой можно только значения атрибутов, определённых на одинаковых доменах. В большинстве реляционных СУБД объект домен не используется, но для каждого атрибута задаётся базовый тип и ряд ограничений, которые проверяются для каждого значения атрибута*

*Схема отношения – именованное множество упорядоченных пар (имя атрибута : имя домена). Степень/арность схемы отношения – мощность этого множества.*

*Схема базы данных (в реляционном, формальном аспекте) – набор именованных схем отношений.*

*Кортеж, соответствующий данной схеме отношения – множество упорядоченных пар (имя атрибута : значение атрибута), которое содержит по одному вхождения каждого атрибута, принадлежащего схеме отношения. Значение атрибута является допустимым значением из домена данного атрибута. Арность кортежа (число элементов в нём) совпадает с арностью соответствующей схеме отношения.*

*Отношения – множество кортежей, соответствующих одной схеме отношения.*

*Свойства отношений:*

*1. В отношении не может быть двух одинаковых кортежей*

*2. Кортежи не упорядочены, атрибуты не упорядочены.*

*3. Имена всех атрибутов в пределах одного отношения должны быть уникальны.*

*Реляционная БД – набор отношений, имена которых совпадают с именами схем отношений в схеме БД.*

Потенциальным ключом отношения называют подмножество атрибутов отношения, которое удовлетворяет двум свойствам:

1. Уникальность (не существуют два одинаковых значения)
2. Безызбыточность (никакое подмножество потенциального ключа не является потенциальным ключом)

Различают ключи:

1. Простые (ИНН, табельный номер)
2. Составные (серия-номер паспорта)

В каждом отношении можно выделить один или несколько потенциальных ключей. Если таких ключей несколько, один из них выбирается в качестве первичного ключа

Первичные ключи (primary key) используются в качестве общих столбцов для связывания таблиц.

На практике часто невозможно выделить такой атрибут. В этом случае используют дополнительный атрибут, называемый суррогатным ключом, который автоматически заполняется уникальными значениями, никогда не изменяется (id).

Ни в одном из потенциальных ключей не допустимы NULL-значения (неопределённые, пустые значения данных, которые в реляционной теории трактуются как отсутствие информации. Их нельзя рассматривать как пустые значения строковые или нулевые числовые. Допустимость пустых значений в том или ином столбце указывается при определении таблиц (not NULL)).

Внешний ключ – такое подмножество атрибутов дочернего отношения, что для любого его непустого значения обязательно найдётся равное значение первичного ключа главного отношения. Обратное утверждение несправедливо, т. е. в родительском отношении можно найти такие кортежи, на которые не ссылаются кортежи дочернего отношения. Значения внешних ключей не обязаны быть уникальными в своём отношении. Во внешнем ключе NULL-значения допустимы в отличие от первичного ключа.

1. **Понятие целостности в реляционной модели.**

**Целостность данных** - это механизм поддержания соответствия базы данных предметной области. В реляционной модели данных определены два базовых требования обеспечения целостности:

* целостность ссылок
* целостность сущностей.

### **Целостность сущностей.**

Объект реального мира представляется в реляционной базе данных как кортеж некоторого отношения. Требование целостности сущностей заключается в следующем:

каждый кортеж любого отношения должен отличатся от любого другого кортежа этого отношения (т.е. любое отношение должно обладать первичным ключом).

Вполне очевидно, что если данное требование не соблюдается (т.е. кортежи в рамках одного отношения не уникальны), то в базе данных может хранится противоречивая информация об одном и том же объекте. Поддержание целостности сущностей обеспечивается средствами системы управления базой данных (СУБД). Это осуществляется с помощью двух ограничений:

* при добавлении записей в таблицу проверяется уникальность их первичных ключей
* не позволяется изменение значений атрибутов, входящих в первичный ключ.

### **Целостность ссылок**

Сложные объекты реального мира представляются в реляционной базе данных в виде кортежей нескольких нормализованных отношений, связанных между собой. При этом:

1. Связи между данными отношениями описываются в терминах функциональных зависимостей.
2. Для отражения функциональных зависимостей между кортежами разных отношений используется дублирование первичного ключа одного отношения (родительского) в другое (дочернее). Атрибуты, представляющие собой копии ключей родительских отношений, называются внешними ключами.

Требование целостности по ссылкам состоит в следующем:

для каждого значения внешнего ключа, появляющегося в дочернем отношении, в родительском отношении должен найтись кортеж с таким же значением первичного ключа.

Пусть, например, даны отношения ОТДЕЛ (N\_ОТДЕЛА, ИМЯ\_ОТДЕЛА) и СОТРУДНИК (N\_СОТРУДНИКА, N\_ОТДЕЛА, ИМЯ\_СОТРУДНИКА), в которых хранятся сведения о работниках предприятия и подразделениях, где они работают. Отношение ОТДЕЛ в данной паре является родительским, поэтому его первичный ключ "N\_отдела" присутствует в дочернем отношении СОТРУДНИК. Требование целостности по ссылкам означает здесь, что в таблице СОТРУДНИК не может присутствовать кортеж со значением атрибута "N\_отдела", которое не встречается в таблице ОТДЕЛ. Если такое значение в отношении ОТДЕЛ отсутствует, значение внешнего ключа в отношении СОТРУДНИК считается неопределенным.

Как правило, поддержание целостности ссылок также возлагается на систему управления базой данных. Например, она может не позволить пользователю добавить запись, содержащую внешний ключ с несуществующим (неопределенным) значением.

В заключение этого раздела отметим, что часто вместо выражения "целостность по ссылкам" употребляют его синонимы "ссылочная целостность", "целостность связей" или "требование внешнего ключа".

1. **Реляционная алгебра и ее операции.**

В реляционной модели определяются два базовых механизма манипулирования данными:

1. Основанная на теории множеств реляционная алгебра

2. Основанное на математической логике реляционное исчисление.

Реляционная алгебра - это язык операций, выполняемых над отношениями - таблицами реляционной базы данных. Операции реляционной алгебры позволяют на основе одного или нескольких исходных отношений, называющихся базовыми, создавать другое отношение (производное отношение) без изменения самих исходных отношений. Полученное другое отношение обычно не записывается в базу данных, а существует в результате выполнения SQL-запроса - массиве, создаваемом функциями для работы с базами данных в языках программирования.

Реляционная алгебра и реляционное исчисление различаются степенью их процедурности. Запрос, представленный на языке реляционной алгебры, может быть вычислен на основе вычисления элементарных алгебраических операция с учётом их приоритета и возможных скобок. Формула реляционного исчисления только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения, поэтому языки реляционного исчисления являются непроцедурными и декларативными. Язык SQL основывается на реляционной алгебре и при этом содержит некоторые элементы реляционного исчисления.

Можно ещё добавить картинки/пояснения для операций 1-4

1. Объединение R3 = R1ᴗR2

2. Пересечение R3 = R1ᴖR2 (R3=R1-(R1-R2))

3. Разность R3 = R1-R2

4. Декартово произведение R3 = R1 x R2

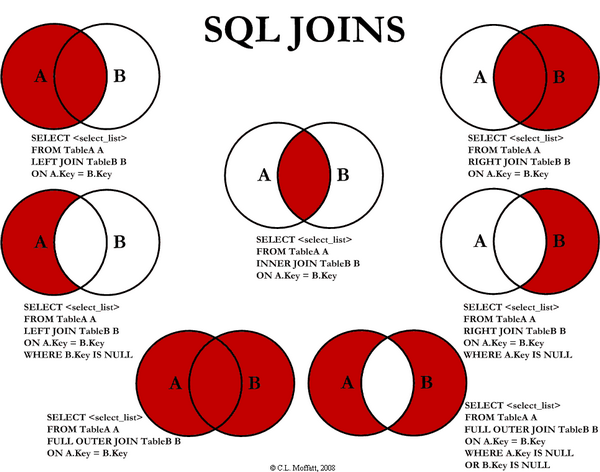
5. Операция проекции – отбор атрибутов отношения Степень результата R2 = модуль I. I – подмножество R1

работает, как и операция выборки, только с одним отношением  и определяет новое отношение *R*, в котором есть лишь те атрибуты (столбцы), которые заданы в операции, и их значения.

6. Операция выборки (селекции) В R2 включаются все кортежи из R1, для которых выражение истинно. При этом R2 может как не содержать ни одного кортежа, так и совпадать с отношением R1.

Операция выборки - унарная операция - и записывается следующим образом:,где *P* - предикат (логическое условие); условие отбора кортежей, в состав входят имена атрибутов, операций и константы.

7. Операция соединения. Аналог join на SQL. Данная операция определена над двумя отношениями, у которых есть общее подмножество атрибутов. В отличие от операции декартово произведение при соединение склеиваются только те кортежи R1 и R2, которые имеют одно и то же значение общего атрибута. При этом общий атрибут попадает в результат один раз. Операция эквивалентна операции выборки из декартова произведения. В результат операции соединения не входят кортежи отношений R1 и R2, для которых не находится одинаковых значений в общем атрибуте. Ввиду этой особенности операцию называют операцией внутреннего соединения (inner join (join)). Также в языке SQL поддерживается три операции внешнего соединения: левая, правая, полная. В левом внешнем соединении результат внутреннего соединения дополняется оставшимися кортежами отношения, стоящего слева. В правом внешнем соединении результат внутреннего соединения дополняется оставшимися кортежами отношения справа. В полном внешнем соединении в результат добавляются все несвязанные кортежи, дополненные неопределёнными значениями.



8. Операция деления. Для выполнения операции деления отношения R1 и R2 должны иметь общее подмножество атрибутов. Причём в отношении R2 это подмножество является множеством его атрибутов. В SQL прямо не задаётся (можно выразить)

Результатом операции деления () является набор кортежей (строк) отношения R1, которые соответствуют комбинации всех кортежей отношения R2. Для этого нужно, чтобы в отношении R2 была часть атрибутов (можно и один), которые есть в отношении R1. В результирующем отношении присутствуют только те атрибуты отношения R1, которых нет в отношении R2.

1. **Функциональные зависимости. Математические свойства ФЗ, теоремы.**

Нормализация основана на анализе функциональных зависимостей между атрибутами отношений.

Пусть даны два атрибута x, y некоторого отношения. y функционально зависит от x, если в любой момент времени каждому значению x соответствует ровно одно значение y. x -> y. При этом x, y могут быть не только единичными атрибутами, но и группами атрибутов некоторого отношения. Подмножество атрибутов x называют детерминантом функциональной зависимости. Функциональные зависимости представляют собой связи "один-ко-многим", существующие в данном отношении.

Пускай есть отношение Сотрудники(код, ФИО, пол, дата\_рождения).

В нем можно выделить зависимости:

код-> ФИО

код -> пол

код -> дата\_рождения

код->(ФИО, дата\_рождения)

При этом нельзя говорить о функциональной зависимости типа ФИО->код или дата\_рождения->ФИО

Однако, если в БД нет одинаковых ФИО, то можно сказать, что существует зависимость ФИО->дата\_рождения, а также можно попробовать построить зависимость ФИО->пол. Но в общем случае таких зависимостей нет.

Математические свойства функциональных зависимостей:

1) Рефлективность (рефлексивность?): если y является подмножеством x, то x определяет y. Такая функциональная зависимость называется тривиальной.

2) Дополнение: A -> B AC -> BC

3) Транзитивность: A -> B и B -> C, тогда A -> C, говорят C транзитивно зависит от A

4) Самоопределение: X -> X

5) Декомпозиция: A -> BC => A -> B, A -> C

6) Композиция: A -> B и С -> D => AC -> BD

7) Теорема о всеобщей зависимости или теорема всеобщего объединения:

если A -> B и C -> D => A u (C – B) -> BD

1. **Декомпозиция отношений. Декомпозиция без потерь.**

Процесс устранения нежелательной зависимости называется декомпозицией, или разбиением отношения. Количество таблиц (отношений) при этом увеличивается, но количество хранимых данных уменьшается.

**Декомпозиция отношения** - это взятие одной или нескольких проекций исходного отношения так, чтобы эти проекции в совокупности содержали все атрибуты исходного отношения. Если по декомпозированному отношению можно полностью восстановить отношение в исходном виде, используя операцию объединения, то данные не потеряны.

Проекции r1, r2 отношения r называют декомпозицией без потерь, если отношение r точно восстанавливается из них при помощи естественного соединения для любого состояния отношения r.

**Теорема Хеза**. Пусть R(A, B, C) - некоторое отношение с атрибутами A, B, C. Если имеется ФЗ A->B, то R1(A, B) и R2(A, C) образуют декомпозицию без потерь. То есть, любую функциональную зависимость отношения можно вынести в отдельное отношение, оставив ее детерминант в исходном отношении. При этом никакая информация не будет утеряна. Обратимый пошаговый процесс декомпозиции отношений с устранением нежелательных функциональных зависимостей называется нормализацией.

1. **Нормализация БД. Преимущества/недостатки/свойства.**

Данный процесс основан на анализе функциональных зависимостей между атрибутами отношений.

Пусть даны 2 атрибута x и y некоторого отношения. y функционально зависит от x, если в любой момент времени каждому значению x соответствует ровно одно значение y: . При этом x и y могут быть не только единичными атрибутами, но и группами атрибутов некоторого отношения. Подмножество атрибутов называют детерминантом функциональной зависимости.

Функциональный зависимости представляют собой связи типа 1:N, существующие внутри отношения.

Процесс нормализации нацелен на устранение избыточности данных. (Преимущество)

Нормализация имеет существенный недостаток – это замедление работы СУБД при выполнении запросов на извлечение данных. В нормализованные БД практически каждый запрос требует соединения данных нескольких таблиц. Соединение таблиц – это операция, требующая определённых затрат ресурсов (памяти и процессорного времени). Чем выше степень нормализованности БД, тем больше в ней таблиц, следовательно, тем медленнее выполняются запросы на выборку.

Также с ростом БД сильно затрудняется её восприятие.

(Свойства?!)

1. **Нормальные формы. (1-5NF + BCNF)**

****

**Первая нормальная форма**

Отношение находится в 1NF, если оно удовлетворяет следующим двум условиям:

1. Значения всех его атрибутов атомарны. Это условие выполняется автоматически согласно свойствам отношений. Атрибут считается атомарным, если для извлечения атрибута требуется одна операция доступа к бд.
2. Отсутствуют повторяющиеся группы атрибутов. Повторяющейся группой называется подмножество атрибутов отношения, определенных на одном и том же домене и имеющих одинаковую семантику.

Доходы сотрудников(код, ФИО, доходы янв, доходы фев, … доходы дек)

Здесь имеется повторяющаяся группа атрибутов, где доходы янв, … хранят одно и то же. Эта группа является повторяющейся и плохо подходит для реляционной алгебры.

Эту группу нужно устранить

Доходы сотрудников(код, ФИО, номер мес, доход)

В этом отношении первичный ключ будет состоять из код, номер мес

**Неключевой атрибут** функционально полно зависит от составного ключа, если он функционально зависит от ключа в целом, но не находится в функциональной зависимости от какого-либо подмножетсв составного ключа. Рассмотрим неприводимое множество функциональных зависимостей для отношения Доходы сотрудников

Первая ФЗ является полной

(код, месяц) -> доход

Вторая в качестве детерминанта имеет подмножество составного ключа

код->ФИО

Именно эта ФЗ создает дублирование значений атрибута ФИО, поэтому является нежелательной

Для устранения нежелательной зависимости выполним декомпозицию в соответствии с теоремой Хеза и выделим эту зависимость в отдельное отношение.

Сотрудники(код, ФИО)

Доход сотрудников(код, номер мес, доход)

В итоге получим нормализованную структуру из двух отношений.

#### **Вторая нормальная форма**

Отношение находится во второй нормальной форме, если оно находится в первой нормальной форме, и каждый неключевой атрибут функционально полно зависит от ключа.

#### **Третья нормальная форма**

Отношение находится в третьей нормальной форме, если оно находится во второй нормальной форме, и каждый неключевой атрибут нетранзитивно зависит от первичного ключа.

Сотрудники (код, ФИО, должность, оплата)

Составим неприводимое множество функциональных зависимостей при условии, что оклад зависит только от занимаемой должности.

код->ФИО

код->должность

должность->оклад

Выходит, что оклад транзитивно зависит от кода сотрудника код-> должность -> оклад

ФЗ должность-оклад нежелательна, т.к. приводит к дублированию атрибута Оклад.

При этом возникают следующие проблемы обновления данных:

если в данный момент ни один из сотрудников не работает в какой-либо должности, то нельзя ввести данные о соответствующем должностном окладе.

Соответственно, при изменении оклада необходим просмотр всего отношения и изменения кортежей всех сотрудников, занимающих должность, по которой изменяется оклад.

Для устранения этих проблем необходимо декомпозировать исходное отношение на два:

Сотрудники(код, ФИО, должность)

Должность(должность, оклад)

Первые три нормальные формы обязательны, а следующие - необязательны.

#### **Нормальная форма Бойса-Кодда BCNF**

Отношение находится в BCNF, если оно находится в третьей нормальной форме и в нем отсутствуют зависимости атрибутов первичного ключа от неключевых атрибутов. Ситуация, когда отношение находится в 3NF, но не находится в BCNF, возникает редко, когда существует составной ключ, включающий неключевые атрибуты. Поэтому, можно считать, что 3NF эквивалентно BCNF.

**Четвертая нормальная форма**

Отношение находится в четвертой нормальной форме, если оно находится в нормальной форме Бойса-Кодда, и в нем отсутствуют многозначные функциональные зависимости.

Четвертая нормальная форма касается отношений, в которых имеются повторяющиеся наборы данных. Как правило, нарушение 4NF происходит в таблицах связи, которые связывают более двух сущностей. В этом случае используется декомпозиция, основанная на многозначных зависимостях. Рассмотрим в качестве примера следующее отношение

Расписание занятия(день, номер пары, группа, препод, ауд)

В данном отношении можно выделить несколько альтернативных составных ключей.

(день, номер пары, группа)

(день, номер пары, препод)

(день, номер пары, ауд)

предмет ->> преподаватель - многозначная функциональная зависимость

предмет ->> группа

Наличие многозначных зависимостей привносит в отношение избыточность и привносит проблемы с обновлением данных.

В приведенном примере избавиться от мфз сложно, но возможно выделить в отдельное отношение сочетание ПредметПреподавательГруппа. Если выделить все эти возможные сочетания, то отношение расписания можно преобразовать к след. виду:

Расписание(день, номер пары, предм преп группа, ауд)

#### **Пятая нормальная форма**

Отношение находится в пятой нормальной форме тогда, и только тогда, когда любая зависимость по соединению в нем определяется только его потенциальными ключами.

Не всегда можно декомпозировать по отношению. Отношения, в которых имеются зависимости по соединению.

1. **Основные объекты БД. Определение.**

**База данных -** это контейнер, в котором будут содержаться таблицы и другие объекты. (DATABASE)

**Схема** - часть базы данных, в пределах которой имена создаваемых объектов должны быть уникальными. (имя схемы) - (имя объекта). (SCHEMA)

**Таблица** - мультимножество строк. Таблицу можно создать, не задавая первичный ключ. (TABLE)

**Индекс** - вспомогательный объект, который служит для ускорения поиска данных, однако замедляет операции вставки, удаления и обновления строк в таблице. (INDEX) Для потенциальных ключей индексы индексы создаются автоматически. Каждая СуБД реализует работу с индексами по-своему. Можно создавать индекс по нескольким столбцам.

**Представление** - именованный запрос на выборку, который хранится в БД и выполняется при любом обращении к нему по имени, создавая при этом виртуальную таблицу с отобранными данными. (VIEW)

**Хранимая процедура** - данный объект БД пишется на языке процедурного расширения языка SQL, который дополняет язык SQL такими управляющими структурами языков высокого уровня, как ветвление и циклы, и позволяет реализовывать любые алгоритмы обработки данных. Хранимый код постоянно хранится на сервере и выполняется по запросу на его запуск из приложений клиентов. (STORED PROCEDURE FUNCTION)

**Триггер** - особый вид хранимой процедуры, который срабатывает автоматически при наступлении определенных событий в БД ( TRIGGER)

Основные события в БД:

* Вставка строк в таблице
* Удаление строк в таблице
* Обновление строк в таблице

**Пользователь** (USER)

**Роль** - права пользователей на выполнение определенных действий (ROLE). Эти объекты служат для разграничения доступа информации многочисленным пользователям БД, которые совместно используют общие объекты БД и могут выполнять только те действия, которые определены их ролями.

1. **Команды DDL для работы с таблицами.**

*DDL (Data Definition Language, язык описания данных) — позволяет выполнять различные операции с базой данных, такие как CREATE (создание), ALTER (изменение) и DROP (удаление объектов).*

Прежде всего при создании таблицы задаём её имя и описываем структуру. **CREATE TABLE** имя\_табл (список определений столбцов [список ограничений на таблицу])

*[] – необязательная часть*

Список определений столбцов: имя столбца тип [DEFAULT значение] [ограничение на столбец]

Имя должно быть уникальным в пределах таблицы

Ограничения позволяют задать дополнительные условия проверки вводимых данных, который СУБД проверяет автоматически. Механизм ограничений позволяет поддерживать данные в непротиворечивом состоянии, соответствующим бизнес-правилам предметной области. Ограничения должны быть поименованы, иначе СУБД сделает это сама. Ограничения могут быть созданы для столбца или таблицы в целом.

Ограничения столбцов:

1) NOT NULL/NULL - допустимы ли пустые неопределённые значения в столбце или нет

2) PRIMARY KEY – при обнаружении дубликата операция прерывается

3) UNIQUE – ограничение уникальности

4) REFERENCES <имя таблицы> – внешний ключ, который задаётся для столбца подчинённой таблицы

5) CHECK (имя столбца *BETWEEN 1 AND 8*)– проверка логических выражений при изменении данных таблицы

**DROP TABLE** имя\_таблицы [ RESTRICT | CASCAD] - удаление таблицы из БД.

Данная команда позволяет удалить целиком таблицу и все данные из неё. RESTRICT – при наличии в БД объекта, существование которого зависит от удаляемой таблицы, то удаление будет отменено. CASCAD – вместе с таблицей удалятся все зависящие от ней объекты.

**ALTER TABLE**<имя> - изменение структуры существующей таблицы или ограничений целостности, задаваемых для данной таблицы.

Команда позволяет добавлять и удалять столбцы таблицы. Основное правило при добавлении столбца в таблицу: когда в таблице уже содержатся данные, добавляемый столбец не может быть определён как NOT NULL.

Некоторые реализации могут ограничивать разработчика при использовании некоторых функций ALTER TABLE: может оказаться невозможным удаление столбца, могут быть трудно удалить столбец, зависящий от столбца другой таблицы и т.д.

Например ALTER TABLE Студент ADD адрес VARCHAR[100].

1. **Типы данных в языке SQL.**

**Типы данных в SQL:**

1) Целочисленные: SMALLINT, INTEGER (INT), BIGINT

2) С плавающей точкой: FLOAT, REAL, DOUBLE PRESISION

3) Какие? : NUMERIC, DECEMAL (P, S) (общее кол-во разрядов, кол-во разрядов после точки), NUMBER

4) Символьные: CHAR [(p-p)], CHARACTER[(p-p)] (по умолчанию размер – 1 байт, ограничение 256 байт), VARCHAR (n) (строка переменной длины)

Данные типов CHAR и VARCHAR хранятся по-разному, под CHAR всегда отводится столько байт, сколько указано в размере, а под данные типа VARCHAR память на диске выделяется в соответствии с реальными размерами текса + 4 байта на хранение этого размера.

5) Дата и время: DATE, TIME, TIME STAMP (позволяет хранить время с высокой точностью, до миллисекунд)

6) BLOB (двоичный большой объект) – позволяет хранить не только хорошо структурированную фактографическую информацию, но и такие данные как фото, видео, тексты, Веб-страницы и др.

1. **Команды манипулирования данными. INSERT/UPDATE/DELETE.**

В SQL для операций ввода данных в таблицу, их изменения и удаления предназначены три команды языка манипулирования данными: Insert (вставить), Update (обновить), Delete (удалить).

Команда Insert осуществляет вставку в таблицу новой строки. В простейшем случае она имеет следующий вид: INSERT INTO <имя таблицы> VALUES (<значение>, <значение>, <значение>…). При такой записи указанные в скобках после ключевого слова VALUES значения вводятся в поля добавленной в таблицу новой строки в том порядке, в котором соответствующие столбцы указаны при создании таблицы. В случае, если требуется ввести данные в каком-либо другом порядке или не все, то используется следующий синтаксис: INSERT INTO <имя таблицы> (поля таблицы) VALUES (<значение>, <значение>, <значение>…). Также можно извлечь значения из одной таблицы и разместить его в другой. INSERT INTO <имя таблицы> SELECT …

Команда DELETE осуществляет удаление строк из таблицы. В простейшем случае DELETE FROM <таблица>; удаляет все строки таблицы. Для удаления из таблицы сразу нескольких строк, удовлетворяющих условию, можно воспользоваться WHERE, например, DELETE FROM <таблица> WHERE <условие>.

Команда UPDATE позволяет изменять, то есть обновлять значения некоторых или всех полей в существующей строке или строках таблицы.

UPDATE <имя таблицы> SET <<поле = значение>, <поле = значение>, <поле = значение>…> (возможно, условие через WHERE)

1. **Команды манипулирования данными. SELECT.**

В реляционной модели определяется два базовых механизма манипулирования данными.

1. Основанная на теории множеств реляционная алгебра

2. Основанное на математической логике реляционное исчисление.

Новое отношение. Полученное из одного или нескольких имеющихся отношений называется производным отношением.

Исходные отношения, входящие в состав БД называются базовыми.

Реляционная алгебра и реляционные исчисления отличаются степенью их процедурности. Запрос, представленный на языке реляционной алгебры, может быть вычислен на основе вычисления элементарных алгебраических операций с учётом их приоритетов и возможных скобок. Формула реляционного исчисления только устанавливает условия, которым должны удовлетворять кортежи результирующего отношения, поэтому языки реляционного исчисления являются более непроцедурными(??) и декларативными(??).

Язык SQL основывается на реляционной алгебре и при этом содержит некоторые элементы реляционного исчисления.

(Рисунок 5.1.)

(продолжение п.7) Так же в языке SQl поддерживаются 3 операции внешнего (?) соединения:

1) Левое: результат внутреннего соединения дополняется оставшимися кортежами отношения, стоящего слева, т.е. (Рисунок 5.2)

2) Правое: результат внутреннего соединения дополняется оставшимися кортежами отношения, оставшимися справа (Рисунок 5.3)

3) Полное: в результат добавляются все несвязанные кортежи, дополненные неопределёнными значениями (Рисунок 5.4)

8. Операция деления: для выполнения операции отношения R1 и R2 должны иметь общее подмножество атрибутов, причем отношение в R2 это подмножество является множеством его атрибутов. (Рисунок 5.5)

Реляционные исчисления

Даны Сотрудники и Отделы (Рисунок 5.6.). Узнать номера и имена сотрудников, являющихся начальниками отдела с количеством работников более 10:

1) Выполнить соединение отношений сотрудники – отелы по условию сотр\_номер = отд\_нач.

c1 = сотрудники [сотр\_номер = отд\_нач](отделы?).

2) Из полученного соединения произвести выборку по условию отд\_кол > 10.

3) Спроецировать результаты предыдущей выборки на атрибуты сотр\_имя и сотр\_номер

c3 = c2 [сотр\_имя, сотр\_номер]

На языке реляционного исчисления данный запрос может быть задан так:

выдать сотр\_имя и сотр\_номер таких, что существует отдел с таким (?), что сотр\_имя (?) отдел\_нач и (?) отд\_кол > 10

Команда выборки данных SELECT реализует все операции реляционной алгебры и некоторые предложения реляционного исчисления. Результатом команды выборки всегда является новая таблица (Возможно, пустая или содержащая одно значение, которое трактуется как таблица из одного столбца и одной ячейки). Новая таблица является виртуальной, т.е. она существует в оперативной памяти до тех пор, пока в ней есть необходимость, она может участвовать в другой команде SQL, в другой команде SELECT, UPDATE, DELETE, как результат вложенного запроса, или передаётся клиенту для последующей обработки.

SELECT [DISTINCT]<список выражений> FROM <список таблиц/представлений/запросов> [WHERE <условие отбора>] [GROUP BY <список выражений для группировки>] [HAVING <условие отбора…>] [ORDER BY <…>]

SELECT \* FROM student // \* означает, что поиск будет осуществлён (по всем таблицам?)

SELECT \* FROM student where fio = ‘Иванов’ //возможны однофамильцы - неправильно

SELECT DISTINCT name\_sub from subject // уникальные поля?

SELECT fio FROM student ORDER BY fio /\* по умолчанию значения сортируются по возрастанию, сортировать можно любые типы данных кроме бинарных (строки сортируются лексикографически, даты в хронологическом порядке). Если не надо по возрастанию, то DESC –

сортировка по убыванию (порядок сортировки нужно указывать после каждого поля: fio DESC; phone ASK – по возрастанию) ??\*/

SELECT fio, phone FROM student ORDER BY 1

Условие отбора может включать в себя следующие операции: <, >, =, !=, <=, >=, is NULL, is not NULL, AND, OR, NOT, BETWEEN x AND y, IN (операция принадлежности множеству), LIKE

SELECT fio, phone FROM student WHERE fio is not null and phone between 123 and 345 and fio in (‘Иванов’, ‘Петров’) or fio LIKE ‘%ов\_’

fio LIKE ‘%ов\_’ – все фамилии с окончанием на ов\_

LIKE позволяет реализовать регулярные выражения: % означает любое количество подряд идущих символов, \_ означает обязательное наличие одного символа.

Создание вычисляемых столбцов (операция расширения из реляционной алгебры). Вычисляемый столбец – это фиктивный столбец, данные которого не хранятся в БД, а вычисляются на основе других столбцов. Вычисляемому столбцу, как правило, присваивается какое-то имя псевдоним).

SELECT <выражение для выч. столбцов> AS <псевоним>

В выражениях для вычисляемых столбцов могут использоваться имена столбцов, операторы, функции и скобки.

SELECT (mark-1) AS mark2

Псевдоним моно присвоить любому столбцу, не обязательно вычисляемому.

SELECT ‘СТУДЕНТ’ FROM student /\* СТУДЕНТ – константа, значит её и получим, а получим столько раз, сколько записей просмотрит команда SELECT, т.е. по количеству строк в таблице student \*/

SELECT fio || ‘,’ || phone AS fio,phone from student where phone is not null and fio is not nul

Функции:

1) Текстовые функции для управления текстовыми строками, например, функция ltrim обрезает незначащие пробелы слева (rtrim – справа), substr берёт подстроку строки s с позиции n до позиции m, upper и lower – верхний и нижний регистр

2) Числовые функции для выполнения математических операций: mod, тригонометрические функции, логарифм, abs, round, trunc

3) Функции даты и времени: получить текущую дату, время, приведение, получение компонент даты

4) Функции преобразования типов данных: date(x) – строку в дату

5) Системные функции

6) Оператор CASE: SELECT code\_st, code\_sub CASE mark WHEN 5 then ‘отл’ WHEN 4 then ‘хор’… END from mark

1. **Способы соединения таблиц в sql-запросах.**

Соединение таблиц запросов Способы объединения таблиц в SELECT-запросах:

1. Декартово произведение - это операция, при выполнении которой каждая строка одной таблицы склеивается с каждой строкой другой таблицы. SELECT \* from students, marks
2. Внутреннее, или естественное соединение таблиц. Таким способом можно соединять только таблицы, имеющие общие столбцы. При выполнении данной операции соединяются только строки, имеющее общее значение в столбце связи. Как правило. таким способом соединяются таблицы, связанные отношением один-ко-многим, а в качестве столбцов связи используются первичный ключ главной таблицы и внешний ключ подчиненной.

Вариант 1: выборка из декартового произведения. select \* from students, marks where students.cod\_st = marks.cod\_st Вариинт 2: операция соединения select \* from students from students join marks on students.cod\_st=marks.cod\_st Соединение может быть не только по ключам, любые два столбца, подходящие по типу, могут соединяться. Но лучше так не делать.

***Существуют следующие типы соединения таблиц:***

**− *CROSS JOIN* (перекрёстное соединение)** — выполняется декартово произведение строк таблиц. Размер результирующего набора данных равен произведению количества строк заданных таблиц;

**− *INNERJOIN* (внутреннее соединение)** — в результирующий набор данных попадают пары строк таблиц, для которых вы- полняется условие соединения таблиц. Результирующий набор может не содержать строк;

**− *LEFT JOIN* или *LEFT OUTER JOIN* («левое» соединение)** — в результирующий набор данных попадают пары строк таблиц, для которых выполняется условие соединения таблиц, а также строки таблицы *table\_1*, которым не найдено соответствующих по условию строк из таблицы *table\_2*. В последнем случае в этих строках всем полям, соответствующих столбцам таблицы *table\_2*, будет присвоено значение *NULL*. Результирующий набор гарантированно содержит все строки таблицы *table\_1*;

**− *RIGHT JOIN* или *RIGHT OUTER JOIN* («правое» соединение)** — в результирующий набор данных попадают пары строк таблиц, для которых выполняется условие соединения таблиц, а также строки таблицы *table\_2*, которым не найдено соответствующих по условию строк из таблицы *table\_1*. В последнем случае в этих строках всем полям, соответствующих столбцам таблицы *table\_1*, будет присвоено значение *NULL*. Результирующий набор гарантированно содержит все строки таблицы *table\_2*;

**− *FULL JOIN* или *FULL OUTER JOIN* (полное соединение)** — в результирующий набор данных попадают пары строк таблиц, для которых выполняется условие соединения таблиц, а также строки таблицы *table\_1*, которым не найдено соответствующих по условию строк из таблицы *table\_2*, и строки таблицы *table\_2*, которым не найдено соответствующих по условию строк из таблицы *table\_1*. Результирующий набор аналогичен объединению результатов выполнения «левого» и «правого» соединения заданных таблиц.

1. **Вложенные запросы. Кореллированные и некореллированные запросы.**

Вложенные запросы – это запросы, которые могут содержаться в теле другого запроса, они могут находится в from, where, having и в качестве вычисляемого столбца в select.

Правила использования вложенных запросов:

1. Тело вложенного запроса заключается в скобки
2. Вложенные запросы могут содержать другие вложенные запросы, при этом выполнение начинается с самого глубокого вложенного запроса и заканчивается выполнением внешнего запроса. Во вложенном запросе не следует использовать order by, поскольку сортировка результатов запроса должна быть выполнена 1 раз при выполнении внешнего запроса.

Вложенные запросы, которые не используют данные внешнего запроса называются некоррелированными и наоборот. Коррелированные запросы более понятны, но менее производительны (вычисляется отдельно для каждого внешнего значения).

В случае некоррелированного запроса вложенные запрос выполняется 1 раз и его результаты используются во внешнем запросе в качестве констант.

ALL и ANY применяются во вложенных запросах после where и having. ALL – все строки вложенного select запроса, ANY – хотя бы одна строка.

Select cod\_st, name\_st from student where all (select mark from marks where cod\_st = students, cod\_st) = 4 -- студенты со всеми четвёрками

EXIST служит для проверки результатов вложенного запроса на пустоту, возвращает истину, если вложенный запрос имеет хотя бы одну строку. Пусть нужно получить всех студентов, у которых нет оценок

Select s.cod\_st, s.name\_st from student , mark m where NOT EXIST (select cod\_st from marks where cod\_st = s.cod\_st)

UNION [ALL] - объединение

INTERSECT [ALL] - пересечение

EXCEPT [ALL] - исключение

ALL запрещает удаление дубликатов из результатов операций.

Получение всех однофамильцев, не удаляя дубликаты:

Select name\_st from students1 UNION ALL select name\_st from students

1. **Представления (View). Создание. Удаление. Обновление.**

****

Представление (VIEW) - это именованный запрос на выборку, который хранится в бд и выполняется на сервере при любом обращении к нему по имени, создавая при этом виртуальную таблицу с отобранными данными.

Представления иногда называют «виртуальными таблицами». Такое название связано с тем, что представление доступно для пользователя как таблица, но само оно не содержит данных, а извлекает их из таблиц в момент обращения к нему. Если данные изменены в базовой таблице, то пользователь получит актуальные данные при обращении к представлению, использующему данную таблицу.

Для создания представления используется оператор CREATE VIEW, имеющий следующий синтаксис:

*CREATE [OR REPLACE]*

*[ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]*

*VIEW view\_name [(column\_list)]*

*AS select\_statement*

*[WITH [CASCADED | LOCAL] CHECK OPTION]*

**ВЕРСИЯ ПОКОРОЧЕ ДЛЯ ИНТЕЛЛЕКТУАЛОВ Okayga:**

*CREATE VIEW view\_name [(column\_list)]*

*AS select\_statement*

view\_name — имя создаваемого представления. select\_statement — оператор SELECT, выбирающий данные из таблиц и/или других представлений, которые будут содержаться в представлении

Оператор CREATE VIEW содержит 4 необязательные конструкции (**не думаю, что нужно**):

OR REPLACE — при использовании данной конструкции в случае существования представления с таким именем старое будет удалено, а новое создано. В противном случае возникнет ошибка, информирующая о существовании представления с таким именем и новое представление создано не будет. Следует отметить одну особенность — имена таблиц и представлений в рамках одной базы данных должны быть уникальны, т.е. нельзя создать представление с именем уже существующей таблицы. Однако конструкция OR REPLACE действует только на представления и замещать таблицу не будет.

ALGORITM — определяет алгоритм, используемый при обращении к представлению (подробнее речь об этом пойдет ниже).

column\_list — задает имена полей представления.

WITH CHECK OPTION — при использовании данной конструкции все добавляемые или изменяемые строки будут проверяться на соответствие определению представления. В случае несоответствия данное изменение не будет выполнено. Обратите внимание, что при указании данной конструкции для необновляемого представления возникнет ошибка и представление не будет создано. (подробнее речь об этом пойдет ниже).

По умолчанию колонки представления имеют те же имена, что и поля возвращаемые оператором SELECT в определении представления. При явном указании имен полей представления column\_list должен включать по одному имени для каждого поля разделенных запятой.

Для просмотра содержимого представления мы используем оператор SELECT (полностью аналогично как в случае простой таблицы), с другой стороны, оператор SELECT есть в самом определении представления, т.е. получается вложенная конструкция — запрос в запросе. При этом, некоторые конструкции оператора SELECT могут присутствовать в обоих операторах. Возможны три варианта развития событий: они обе будут выполнены, одна из них будет проигнорирована и результат не определен. Рассмотрим подробнее эти случаи:

· Если в обоих операторах встречается условие WHERE, то оба этих условия будут выполнены как если бы они были объединены оператором AND.

· Если в определении представления есть конструкция ORDER BY, то она будет работать только в случае отсутствия во внешнем операторе SELECT, обращающемся к представлению, собственного условия сортировки. При наличии конструкции ORDER BY во внешнем операторе сортировка, имеющаяся в определении представления, будет проигнорирована.

· При наличии в обоих операторах модификаторов, влияющих на механизм блокировки, таких как HIGH\_PRIORITY, результат их совместного действия не определен. Для избежания неопределенности рекомендуется в определении представления не использовать подобные модификаторы.

Представление называется обновляемым, если к нему могут быть применимы операторы UPDATE и DELETE для изменения данных в таблицах, на которых основано представление. Для того, чтобы представление было обновляемым должно быть выполнено 2 условия:

1. Соответствие 1 к 1 между строками представления и таблиц, на которых основано представление, т.е. каждой строке представления должно соответствовать по одной строке в таблицах-источниках.

2. Поля представления должны быть простым перечислением полей таблиц, а не выражениями col1/col2 или col1+2.

Синтаксис этих запросов идентичен синтаксису при работе с таблицами, а именно:

* UPDATE `view\_name` SET `column\_name` = `new\_value' [WHERE condition];
* DELETE FROM `view\_name` [WHERE condition];

**В ЖОПУ ФУЛЛ ВЕРСИИ, ЛЕНЬ ИХ ИСКАТЬ**

1. **Триггеры. Пример.**

**Триггер** - это особый вид хранимой процедуры, который запускается автоматически при наступлении определенных событий в БД.

Дополнительным его преимуществом является то, что он работает независимо от того, какое из клиентских приложений вызвало событие, активирующее этот триггер.

Триггер уровня таблицы характеризуется таблицы характеризуется следующими признаками, которые должны быть заданы при его создании:

* Уникальное имя триггера
* Активизирующее действие - это команда, которая вызывает запуск триггера (INSERT, UPDATE, DELETE, ...)
* Время активизации выполнения триггера (до или после активизирующего действия)
* Область действия (выполнение триггера либо один раз для каждого оператора модификации таблицы, либо для каждой строки)
* Условие активизации (необязательное дополнительное условие, которое должно выполняться для запуска триггера)
* Тело триггера

На каждое событие может быть несколько триггеров.

**Создание триггера:**

CREATE [OR REPLACE] TRIGGER <имя> <время активизации> ON <имя таблицы>

[FOR EACH ROW]

[WHEN <доп. усл>]

AS

<блок кода>

**Пример:**

CREATE TRIGGER st\_def BEFORE

DELETE ON Students FOR EACH ROW

BEGIN

INSERT INTO ArchiveStudents VALUES (

: OLD.cod\_st,

: OLD.name\_st,

: OLD.born,

: OLD.phone)

END

DROP TRIGGER st\_def

ALTER TRIGGER st\_def DISABLE/ENABLE

Можно обратиться к столбцам таблицы, используя предопределенные имена OLD и NEW. OLD.col\_name обращается к столбцу существующей строки прежде, чем она модифицируется или удалится. NEW.col\_name обращается к столбцу новой строки, которая будет вставлена, или же к существующей строке после того, как она модифицируется. (Это предложение из инета на всякий)

1. **Разграничение доступа. Привилегии и роли.**

Для создания пользователя:

Create User <name>

Identified by <password>

Создание индивидуальных профилей для пользователей базы данных позволяет:

1. Вести индивидуальную политику паролей (время жизни паролей).

2. Ограничить ресурсы (количество одновременно открытых сессий, процессорное время, отводимое пользователю, время ожидания ответа сервера)

Для создания профиля:

Create profile <name> [параметры (количество сессий, параметры пароля) ]

ALTER и DROP конечно тоже есть

Для выполнения любых действий в БД требуются права на их выполнение. Для этих целей вводятся понятия *привилегии и роли.*

Привилегии позволяют использовать определенные команды языка SQL по отношению к определенным ее объектам и предоставляются с помощью команды grant. Упрощают работу с привилегиями роли или так называемые группы привилегий, которые также предоставляются пользователям, с помощью команды grant, что и отдельные привилегии. Отзыв привилегии происходит при помощи команды Revoke. Привилегии, роли могут назначаться пользователям или ролям.

Create Role <name> - создать роль

GRANT/REVOKE <список привилегий/ролей> TO/FROM <список пользователей, получающих права ИЛИ список ролей или public> [with grant option]

With grant option означает, что создатель таблицы хочет, чтобы другие пользователи имели право передавать полномочия на работу с указанной таблицей.

Подключение пользователя к бд: Grant connect to student

Переход всех прав от student to user:

Create role user

Grant student to user

Привилегии:

1. Системные привилегии – права на выполнение Create, Alter, Drop применительно к различным объектам БД в рамках разных схем. *Пример:* Grant create table to student.
2. Объектные привилегии позволяют выполнять конкретные действия на конкретном объекте (например запрос на выборки из таблицы). *Пример:* **GRANT SELECT, INSERT ON** Student **ТО** Р2 **WITH GRANT OPTION**;

GRANT CREATE **ANY** TABLE TO student – для любой схемы

Grant/revoke <список объектных привилегий> on <имя таблицы/представление/проц./триг.> to/from <список ролей/ пользователей> public [with grant option]

Если public, то всем пользователям, если указана необязательная конструкция, то может выполнять другим пользователям свои привилегии.

Привилегии объектного уровня: 1) **Select** позволяет другому пользователю выполнить запрос на выборку к данным указанной таблицы или представления. 2) **Update** – обновление строк в таблице вне зависимости от того, кто эти строки создал. 3) **Insert** – позволяет вставлять строки в таблицу, возможно используя для этих целей обновляемое представление с помощью команды INSERT. 4) **Delete** – позволяет удалять из таблицы любые существующие строки. С использование представления можно ограничить то, какие именно строки будут удалены. 5) **EXECUTE** – даёт возможность пользователю, владеющему хранимым кодом БД, позволить другому пользователю вызывать его процедурные объекты 6) **Alter** – даёт возможность пользователю изменять определение заданной таблицы или последовательности. 7) **Index** – позволяет пользователю создавать индексы на указанную таблицу, владельцем которой он не является. (на свои таблицы он имеет эти привилегии по умолчанию)

1. **Хранимые процедуры и функции. Пример.**

Postgres (возможно) не поддерживает.

Преимущества и недостатки хранимого кода:

1) Сокращение объёма программирования при разработке приложений, т.к. одна однажды созданная подпрограмма может использоваться разными приложениями.

2) Уменьшение сетевого трафика, т.к. если программа включает в себя несколько обращений к БД, то по сети передается не каждый запрос и его результат, а только вызов процедуры и её конечный результат.

3) Повышение производительности, т.к. на сервере есть больше возможностей оптимизации локально выполняющихся запросов и могут быть применены средства, недоступные для клиентской части.

4) Гарантия того, что задача, решаемая хранимой процедурой, будет одинаково выполняться для всех клиентских приложений и для всех клиентских платформ при любых настройках клиентов.

5) Главный минус в том, то перенести хранимый код на другую СУБД - очень трудоемкая задача, требующая практически полного переписывания кода.

**Создание хранимой процедуры**

Create [or replace] procedure/function <имя> [(список параметров)] as <блок кода> return <значение>

Явно должен быть прописан характер параметра: IN, OUT, IN OUT.

Create procedure change\_phone (fio\_stud students.name new\_phone students.phone result OUT number)

As old\_phone students.phone

Begin

Select phone into pld\_phone from students where name\_st = fio\_stud

If old\_phone <> new\_phone update students set phone = new phone where name\_st = fio\_stud

Result := 0;

Else

Result := 1;

End if;

Exeption

When NO\_DATA\_FOUND then result := 2;

When TOO\_MANY\_ROWS result := 3;

When OTHER result := 4;

End;

Вызов хранимых подпрограмм:

e number

Begin

Change\_phone (‘Иванов’, ‘123456789’, e)

End;

**Триггер** – это особый вид хранимой процедуры, которая запускается автоматически при наступлении определенных событий в БД.

Дополнительным преимуществом триггера является то, что он работает независимо от того, какое из клиентских приложений вызвало событие, активизирующее этот триггер.

На каждое событие может быть создано несколько триггеров, и они будут выполняться в порядке их создания. Так как триггер является хранимой процедурой, то он тоже может содержать вложенные операции select, update, delete, который так же могут вызвать триггер.

Create [or replace] trigger <имя>

Время активации актив команда on <имя таблицы>

[for each row]

[when <доп условие>]

As

<блок кода>

1. **Транзакции. Свойства транзакций. Журнал транзакций.**

*Транзакции* – единица работа СУБД, которая может быть либо выполнена целиком, либо вообще не выполнена. Объем транзакции может варьироваться от 1 SQL оператора до всех действий из БД, выполняемых приложением.

*Свойства (ACID):*

1) Atomicity – Атомарность – Это свойство означает, что результаты всех операций должны быть атомарными (неделимыми) в состоянии БД, либо действий не должно быть совершено вообще.

2) Consistency – Согласованность – Транзакция может быть совершена только в том случае, когда действия операций на нарушают целостность БД. То есть, БД фиксирует только допустимые результаты.

3) Isolation – Изоляция – Требуется, чтобы 2 одновременно выполняемые транзакции никоим образом не действовали одна на другую.

4) Durability – Долговечность. После успешного завершения транзакции все изменения, которые были внесены в БД, должны гарантированно сохраняться даже при сбое аппаратуры.

*Журнал транзакций* - все изменения в базе данных записываются в журнал транзакций. Используя эту информацию, СУБД может определить какая транзакция внесла изменения в данные. Информация, записывающая в журнал транзакций, включает: 1) Время каждой транзакции, 2) Изменения внутри каждой транзакции и информация для отката, 3) Информация о завершении или откате каждой транзакции.

1. **Транзакции. Уровни изоляции. Феномены, возникающие при работе с данными.**

*Транзакции* – единица работа СУБД, которая может быть либо выполнена целиком, либо вообще не выполнена. Объем транзакции может варьироваться от 1 SQL оператора до всех действий из БД, выполняемых приложением. Ее характеристики – уровень изоляции, режим доступа, размер области диагностики.

Режимы доступа транзакции – readOnly и readWrite, При первом нельзя осуществлять операции обновления таблиц и определения новых объектов БД. Второй режим установлен по умолчанию и позволяет читать элементы и писать их.

*Уровни изоляции*:

1) Read uncommitted – чтение незафиксированных данных – феномен грязного чтения. Обработка реально несуществующих данных (подробнее ниже – в феноменах).

2) Read committed – чтение зафиксированных данных – феномен неповторяемого чтения – при попытке обращения повторно к данным одной из транзакций оказывается, что их на самом деле уже нет или они изменились.

3) Repeatable read – повторяемость чтения – феномен фантомов – при повторном чтении транзакцией появляются новые подходящие под условие строки, хотя в изначальной таблице их не было.

4) Serializable – упорядочиваемость – транзакции полностью изолируются друг друга. Каждая выполняется так, будто параллельного режима нет.

*Феномены:*

1) *Грязного чтения.* В момент времени t0 были образованы две транзакции Т1 и Т2. В момент времени t1 транзакция Т1 выполняет операцию модификации некоторого объекта БД. В момент времени t2 вторая транзакция Т2 читает этот же объект, после чего успешно завершается в момент времени t3. Транзакция Т1 завершается в момент времени t4, причём завершается выполнением оператора rollback, что приводит к ликвидации в БД последствий изменения объекта О. В результате оказывается, что в транзакции Т2 обрабатывались данные, которые реально не существуют в БД.

2) *Феномен неповторяемого чтения.* Этому феномену подвержены транзакции, читающие некоторые объекты БД и допускающие изменение уже прочитанных объектов другими транзакциями. В момент времени t0 были созданы 2 транзакции. В момент времени t1 вторая транзакция выполняет чтение некоторого объекта БД. В момент времени t2 первая транзакция изменяет этот объект БД или даже удаляет его. В момент времени t3 вторая транзакция повторно считывает этот объект и обнаруживает, что он либо изменился, либо вовсе отсутствует. В SQL феномен неповторяемого чтения может наблюдаться у транзакций, выполняемых на уровне read committed.

3) *Феномен фантомов.* Данному феномену подвержены транзакции, производящие выборку строк и таблиц БД и допускающие добавление к данным таблицам другими транзакциями строк, которые удовлетворяют условию выборки. В момент времени t1 вторая транзакция выполняет выборку из таблицы R по условию C, в результате этой выборки появляются строки, удовлетворяющие условию C. В момент времени t2 транзакция 1 выполняет над таблицей операцию вставки дополнительных строк, в результате которой в таблице появляются новые строки, удовлетворяющие условию C. В момент времени t3 Т2 повторно выполняет выборку из таблицы R и обнаруживает наличие фантомных строк, удовлетворяющих условию C. Данный феномен может наблюдаться у транзакций, выполняемых на уровне изоляции Repeatable read

4) Для транзакций, выполняемых на последнем уровне невозможно проявление феномена фантомов, так как при работе на данном уровне суммарный эффект выполнения набора транзакций идентичен эффекту некоторого последовательно выполнения этих транзакций, что означает их предельную изолированность.

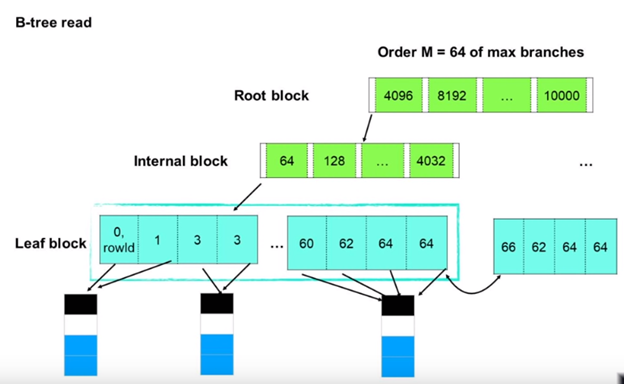
1. **Индексы. Использование B-деревьев при хранении индексов. Операции добавления/изменения/удаления.**

Index – вспомогательный объект, который служит для ускорения поиска данных (where), однако замедляет операции вставки, удаления и обновления строк таблиц (из-за обновления самих индексов). Для потенциальных ключей создаются автоматически. Create Index age ON users(age).

Ускорение работы с использованием индексов достигается в первую очередь за счёт того, что индекс имеет структуру, оптимизированную под поиск — например, [B-дерева](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%B0%D0%BB%D0%B0%D0%BD%D1%81%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE). B–дерево – сбалансированное (длина любых двух путей от корня до листьев отличается не более, чем на единицу), сильно ветвистое дерево (каждый узел дерева ссылается на большое число узлов-потомков).

В контексте B-дерева для хранения индексов в БД: оно имеет внутренние (индексные) и листовые страницы. Индексные вершины содержат упорядоченные ключи, в которых содержится диапазон ключей для искомого (Например, значение в блоке 4096 означает, что оно является разделяющим для двух блоков (к примеру (64 … 4032) и (4160 …)), ниже уровнем (ближе к листьям)) и указатели на страницы-потомки. Листовые вершины на самом нижнем уровне дерева, в каждой из которых хранятся упорядоченные ключи и ссылки на записи из таблицы.

Поиск в таком дереве индекса N происходит следующим образом: начинается с верхнего-корневого блока, который содержит диапазон значений (ключей) и ссылки на блоки нижестоящего уровня. В этом блоке находим необходимый диапазон (min значение > N) переходим от этого блока к нижестоящему блоку в нужный диапазон. Поиск нужного индекса на уровне листьев осуществляется линейно. То есть эффективность поиска в несколько шагов.



*Вставка* (добавление) в такое дерево – осуществляется в несколько этапов:

1) Поиск листовой страницы. То есть, обычный поиск по такому дереву. Если ключа с заданным значением нет, то будет получен номер листовой страницы и позиция внутри этой страницы, куда необходимо произвести вставку.

2) Помещение записи на место. Листовая страница, в которую необходимо занести запись, помещается в буфер оперативной памяти и уже в нем выполняется функция вставки. Размер этого буфера изначально должен превышать размер страницы.

3) Если при выполнении вставка размер используемой части буфера не превышает размера страницы, то на этом вставка заканчивается. Буфер мгновенно выбрасывается во внешнюю память.

4) Если возникло переполнение буфера, то выполняется расщепление страницы. Для этого запрашивается новая страница и используемая часть буфера делится пополам и вторая половина заносится во вновь выделенную страницу. При возврате во внешнюю память модифицируются ссылки между страницами (они находятся в одном и том же односвязном или двусвязном списке).

5) Во внешней памяти необходимо обеспечить доступ от корня к заново заведенной странице. Необходимо модифицировать внутреннюю страницу, являющуюся предком ранее существующей страницы. Нужно вставить соответствующее значение ключа и ссылку на внутреннюю страницу. В этой странице может произойти переполнение тоже, так что она может быть расщепленной как в пункте 4 и операция может повторяться рекурсивно до корня.

6) Если переполняется корень, то эта корневая страница расщепляется на 2 и заводится новая корневая страница, то есть высота дерева увеличивается на 1.

*Изменение или же модификация* – заключается в поиске по дереву нужной листовой страницы, внутри которой (структура – односвязный/двусвязный список) происходит поиск нужного нам значения. Оно содержит в себе ссылку на ссылающиеся данные, по которым и можно произвести изменение.

*Удаление* – в несколько этапов

1) Поиск записи по ключу. Если записи нет, то и удалять ничего не нужно.

2) Удаление записи в буфере, в который была прочитана соответствующая листовая страница.

3) Проверка сумм размеров: буфер + левыйБрат; буфер+правыйБрат. Если он больше размера листовой страницы, значит операция завершается. Буфер возвращается без удаленного элемента во внешнюю память.

4) Если меньше одной из сумм, то происходит слияние буфера с левым или правым братом. Ставшая ненужной листовая страниц переходит в список свободных страниц. Буфер возвращается на место левого или правого брата теперь уже свободной листовой страницы. Корректируется ссылки в списке листовых страниц.

5) Для устранения возможности доступа от корня к освобожденной странице, нужно удалить ключ и ссылку на нее в странице предке. При этом может случиться случай слияния этой страницы с левым или правым братом.

6) Предельный случай – слияние двух оставшихся детей корневой страницы дерева, тогда корневая страница освобождается (так как в ней содержался всего 1 индекс как раз разделяющий этих двух детей) и глубина дерева уменьшается на 1.

1. **Индексы. Использование хэш-таблиц, битовых карт и индексных таблиц при работе с индексами.**

Index – вспомогательный объект, который служит для ускорения поиска данных (where), однако замедляет операции вставки, удаления и обновления строк таблиц (из-за обновления самих индексов). Для потенциальных ключей создаются автоматически. Create Index age ON users(age).

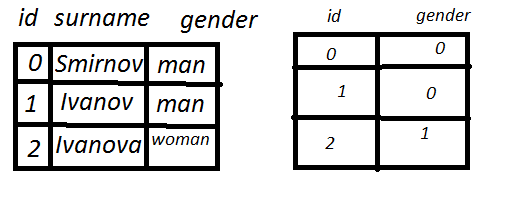
*Хэш-таблицы*

Хешированные индексы реализованы в виде хешированных кластеров. Хешированный кластер — специализированный вид организации данных, обеспечивающий быстрый доступ к строкам разных таблиц, хранящихся вместе. При обращении к хешированному кластеру по значению кластерного ключа (определенные поля) применяется функция хеширования, результатом которой является значение хешированного ключа и адрес блока данных. Хешированный кластер группирует в одном блоке строки, содержащие одинаковые значения хэш-функции от ключей. На любой таблице можно построить только один хешированный индекс.

Хешированные кластеры целесообразно использовать для больших таблиц, поиск по которым, как правило, осуществляется с условиями равенства по ключевому столбцу (столбцам).

*Битовые карты*

Подходят для столбцов с малой селективностью-избирательностью (малым спектром значений). Битовый индекс построен в виде двоичной карты по значениям ключа. Для каждой строки таблицы в определенном бите стоит значение 0 или 1 в соответствии со значением ключа конкретной строки.



*Индексные таблицы*

Это таблица, которая физически является двоичным деревом, относительно своего первичного ключа. Индексные таблицы используют таблицы с короткими строками, обращение к которым всегда производится по первичному ключу. Данные в таких таблицах отсортированы по значениям столбца первичного ключа и сохраняются так, как если бы вся таблица целиком содержалась в одном индексе. Не рекомендуется использовать, если строки имеют большую длину.