**1. Базы данных и системы управления базами данных. Определения, основные функции и классификация.**

***База данных*** - совокупность взаимосвязанных данных некоторой предметной области, хранимых в памяти ЭВМ и организованных таким образом, что эти данные могут быть использованы для решения многих задач многими пользователями.

Основные требования к организации данных:

1. Неизбыточность данных - каждое данное присутствует в БД в единственном экземпляре.
2. Совместное использование данных многими пользователями.
3. Эффективность доступа к БД - высокое быстродействие, т. е. малое время отклика на запрос.
4. Целостность данных - соответствие имеющейся в БД информации её внутренней логике, структуре и всем явно заданным правилам.
5. Безопасность данных – защита данных от преднамеренного или непреднамеренного искажения или разрушения данных.
6. Восстановление данных после программных и аппаратных сбоев.
7. Независимость данных от прикладных программ.

***Система управления базами данных (СУБД)*** - приложение, обеспечивающее создание, хранение, обновление и поиск информации в базах данных.

***Основные функции СУБД***

1. Управление данными во внешней памяти
2. Управление буферами оперативной памяти
3. Управление транзакциями (Транзакция – это последовательность операций над данными, рассматриваемая СУБД как единое целое. Реализуется принцип «либо все, либо ничего»)
4. Журнализация (СУБД должна быть в состоянии восстановить последнее согласованное состояние БД после любого аппаратного или программного сбоя)
5. Поддержка языков БД

***Классификация СУБД***

1. По модели данных:
   1. Дореляционные
      1. Инвестированные списки
      2. Иерархические БД
      3. Сетевые
   2. Реляционные (управляют реляционными БД)
   3. Постреляционные (значения полей не являются неделимыми, типо таблица в таблице)
2. По архитектуре организации хранения данных
   1. Локальные СУБД (все части локальной СУБД размещаются на одном компьютере)
   2. Распределенные СУБД (части СУБД могут размещаться на двух и более компьютерах)
3. По способу доступа к данным
   1. Файл-серверные (БД и приложение расположены на файловом сервере сети. Вся тяжесть выполнения запросов к базе данных и управления целостностью базы данных ложится на приложение пользователя)
   2. Клиент-серверные (БД в этом случае помещается на сетевом сервере, как и в архитектуре "файл-сервер", однако прямого доступа к базе данных из приложений не происходит В итоге снижается нагрузка на сеть.
   3. Встраиваемые (Встраиваемая СУБД — библиотека, которая позволяет унифицированным образом хранить большие объёмы данных на локальной машине. Доступ к данным может происходить через SQL либо через особые функции СУБД. Встраиваемые СУБД быстрее обычных клиент-серверных и не требуют установки сервера, поэтому востребованы в локальном ПО, которое имеет дело с большими объёмами данных)

## 2. Семантическое моделирование данных

**Таблица** — один из самых понятных форматов восприятия данных. Табличное представление:

* не дает полного представления о смысле данных
* с трудом позволяет приложениям моделировать предметную область
* не предоставляет каких-либо средств для представления зависимостей
* не предлагает какого-либо аппарата для разделения сущностей и связей

**Семантическое моделирование данных** — представление данных в графическом виде.

Главным назначением семантических моделей является обеспечение возможности выражения семантики данных. На практике семантическое моделирование используется на первой стадии проектирования базы данных.

Наиболее известным представителем класса семантических моделей предметной области является модель «сущность- связь» или ER-модель, предложенная Питером Ченом в 1976 году

**Основные понятия ER-модели:**

1. ***Сущность*** — это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна. Имя сущности - это имя типа, а не некоторого конкретного экземпляра этого типа.
2. ***Связь*** — это ассоциация, устанавливаемая между сущностями. Эта ассоциация может существовать между разными сущностями или между сущностью и ей же самой (рекурсивная связь). Сущности, включенные в связь, называются ее *участниками*, а количество участников связи называется ее *степенью*. Связи типа «один к одному» и «один ко многим» всегда могут быть представлены с помощью механизма внешнего ключа, помещаемого в одно из отношений.
3. ***Свойством*** сущности (и связи) является любая деталь, которая служит для уточнения, идентификации, классификации, числовой характеристики или выражения состояния сущности (или связи).

Ключи:

1. Первичный ключ - набор атрибутов, которые однозначно идентифицируют каждую запись отношений (Натуральный первичный ключ - не сгенерированный (сгенерированный - id))
   1. Однозначен(уникален)
   2. Неизбыточен
2. Внешний ключ - набор атрибутов, которые соответствуют первичному ключу той таблицы, на которую ссылаются

**Получение реляционной схемы из ER-схемы осуществляется с помощью следующей пошаговой процедуры**

**Шаг 1.** Каждая простая сущность превращается в таблицу. Простая сущность - сущность, не являющаяся подтипом и не имеющая подтипов. Имя сущности становится именем таблицы.

**Шаг 2.** Каждый свойство (атрибут) становится возможным столбцом с тем же именем; может выбираться более точный формат. Столбцы, соответствующие необязательным атрибутам, могут содержать неопределенные значения; столбцы, соответствующие обязательным атрибутам, - не могут.

**Шаг 3.** Компоненты уникального идентификатора сущности превращаются в первичный ключ таблицы. Если имеется несколько возможных уникальных идентификаторов, выбирается наиболее используемый. Если в состав уникального идентификатора входят связи, к числу столбцов первичного ключа добавляется копия уникального идентификатора сущности, находящейся на дальнем конце связи (этот процесс может продолжаться рекурсивно). Для именования этих столбцов используются имена концов связей и/или имена сущностей.

**Шаг 4.** Связи «многие к одному» (и «один к одному») становятся внешними ключами. Т.е. делается копия уникального идентификатора с конца связи «один», и соответствующие столбцы составляют внешний ключ. Необязательные связи соответствуют столбцам, допускающим неопределенные значения; обязательные связи - столбцам, не допускающим неопределенные значения.

**Шаг 5.** Индексы создаются для первичного ключа (уникальный индекс), внешних ключей и тех атрибутов, на которых предполагается в основном базировать запросы.

**Шаг 6.** Если в концептуальной схеме присутствовали подтипы, то возможны два способа:

1. все подтипы в одной таблице,
2. для каждого подтипа - отдельная таблица.

## 3. Реляционная модель данных: структурная, целостная, манипуляционная части. Реляционная алгебра. Исчисление кортежей

Реляционная модель состоит из трех частей: структурной, целостностной и манипуляционной.

1. **Структурная часть** — описывает, из каких объектов состоит реляционная модель. Основными понятиями структурной части реляционной модели являются
   1. тип данных
   2. домен (Можно считать уточнением типа данных. Домен можно рассматривать как подмножество значений некоторого типа данных, имеющих определенный смысл)
   3. атрибут отношения — это пара вида <имя\_атрибута, имя\_домена >. Имена атрибутов должны быть уникальны в пределах отношения. Часто имена атрибутов отношения совпадают с именами соответствующих доменов.
   4. схема отношения — это именованное множество упорядоченных пар <имя\_атрибута, имя\_домена>
   5. схема базы данных — это множество именованных схем отношений. Понятие схемы отношения близко к понятию структурного типа в языках программирования (например, record в языке Pascal или struct в языке C).
   6. кортеж — это множество упорядоченных пар <имя\_атрибута, значение\_атрибута>, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения.
   7. отношение — определенное на множестве из n доменов (не обязательно различных), содержит две части: заголовок (схему отношения) и тело (множество из m кортежей). Значения n и m называются соответственно степенью и кардинальностью отношения.
   8. потенциальный, первичный и альтернативные ключи
   9. реляционная база данных — это набор отношений, имена которых совпадают с именами схем отношений в схеме базы данных.
2. **Целостная часть** — фиксируются два базовых требования целостности, которые должны выполняться для любых отношений в любых реляционных базах данных. Это целостность сущностей и ссылочная целостность (или целостность внешних ключей).
   1. Требования для целостности сущностей:
      1. при добавлении записей в таблицу проверяется уникальность их первичных ключей
      2. не позволяется изменение значений атрибутов, входящих в первичный ключ.
   2. Требования для ссылочной целостности:
      1. для каждого значения внешнего ключа, появляющегося в дочернем отношении, в родительском отношении должен найтись кортеж с таким же значением первичного ключа.

**Манипуляционная часть** описывает два эквивалентных способа манипулирования реляционными данными — реляционную алгебру и реляционное исчисление. Реляционная алгебра в явном виде предоставляет набор операций, а реляционное исчисление представляет систему обозначений для определения требуемого отношения в терминах данных отношений.

**Реляционная алгебра**

Реляционная алгебра является основным компонентом реляционной модели, опубликованной Коддом, и состоит из восьми операторов, составляющих две группы по четыре оператора:

1. Традиционные операции над множествами: объединение (UNION), пересечение (INTERSECT), разность (MINUS) и декартово произведение (TIMES). Все операции модифицированы, с учетом того, что их операндами являются отношения, а не произвольные множества.
2. Специальные реляционные операции: ограничение (WHERE) , проекция (PROJECT), соединение (JOIN) и деление (DIVIDE BY).

Результат выполнения любой операции реляционной алгебры над отношениями также является отношением. Эта особенность называется свойством реляционной замкнутости.

S(Поставщик)(Sno: integer, Sname: string, Status: integer, City: string)

P(Дедаль)(Pno: integer, Pname: string, Color: string, Weight: real, City: string)

SP(Поставка)(Sno: integer, Pno: integer, Qty: integer)

( ( SP JOIN S ) WНEPE Рno = 2 ) [ Sname ]. Получить имена поставщиков, которые поставляют деталь под номером 2

**Дополнительные операторы реляционной алгебры**

* SEMIJOIN (полусоединение)
* SEMIMINUS (полувычитание)
* EXTEND (расширение)
* SUMMARIZE (обобщение)
* TCLOSE (транзитивное замыкание)
* TRANSFORM
* ROLLUP
* CUBE

**Исчисление кортежей**

Реляционное исчисление является альтернативой реляционной алгебре. Внешне два подхода очень отличаются – исчисление описательное, а алгебра предписывающая, но на более низком уровне они представляют собой одно и то же, поскольку любые выражения исчисления могут быть преобразованы в семантически эквивалентные выражения в алгебре и наоборот. Выражение исчисления кортежей содержит заключенный в скобки список целевых элементов и выражение WНERE, содержащее формулу WFF ("правильно построенную формулу"). Такая формула WFF составляется из кванторов (EXISTS и FORALL), свободных и связанных переменных, литералов, операторов сравнения, логических (булевых) опе- раторов и скобок. Каждая свободная переменная, которая встречается в формуле WFF, должна быть также перечислена в списке целевых элементов.

**Синтаксис исчисления кортежей**

объявление-кортежной-переменной ::=RANGE OF переменная IS список-областей

область ::= отношение | реляционное-выражение

реляционное-выражение ::= (список-целевых-элементов)[WHERE wff]

целевой-элемент ::= переменная | переменная.атрибут [AS атрибут]

wff ::= условие |NOT wff |условие AND wff |условие OR wff |IF условие THEN wff | EXISTS переменная (wff) | FORALL переменная (wff) | (wff)

условие ::= (wff) | компаранд операция-отношения компаранд

Если f и g - формулы WFF, то выражение логической импликации

IF f TНEN g также будет формулой WFF с семантикой, идентичной семантике формулы ( NOT f ) OR g

( SХ.Sno ) WHERE SX.City = 'Смоленск' AND SX.Status > 20. Получить номера поставщиков из Смоленска со статусом больше 20

## 4. Теория проектирования реляционных баз данных: функциональные зависимости, нормальные формы

Классический подход к проектированию реляционных баз данных заключается в том, что сначала предметная область представляется в виде одного или нескольких отношений, а далее осуществляется процесс нормализации схем отношений, причем каждая следующая нормальная форма обладает свойствами лучшими, чем предыдущая.

Каждой нормальной форме соответствует некоторый определенный набор ограничений, и отношение находится в некоторой нормальной форме, если удовлетворяет свойственному ей набору ограничений. Примером набора ограничений является ограничение первой нормальной формы – значения всех атрибутов отношения атомарны. Поскольку требование первой нормальной формы является базовым требованием классической реляционной модели данных, будем считать, что исходный набор отношений уже соответствует этому требованию.

В теории реляционных баз данных обычно выделяется следующая последовательность нормальных форм: ◦ первая нормальная форма (1НФ или 1NF); ◦ вторая нормальная форма (2НФ или 2NF); ◦ третья нормальная форма (3НФ или 3NF); ◦ нормальная форма Бойса-Кодда (НФБК или BCNF); ◦ четвертая нормальная форма (4НФ или 4NF); ◦ пятая нормальная форма, или нормальная форма проекции-соединения (5НФ или 5NF или PJ/NF).

Основные свойства нормальных форм такие: ◦ каждая следующая нормальная форма в некотором смысле лучше предыдущей; ◦ при переходе к следующей нормальной форме свойства предыдущих нормальных свойств сохраняются.

Процесс проектирования реляционной базы данных на основе метода нормализации преследует две основные цели: ◦ избежать избыточности хранения данных; ◦ устранить аномалии обновления отношений.

В основе метода нормализации лежит декомпозиция отношения, находящегося в предыдущей нормальной форме, в два или более отношения, удовлетворяющих требованиям следующей нормальной формы. Считаются правильными такие декомпозиции отношения, которые обратимы, т. е. имеется возможность собрать исходное отношение из декомпозированных отношений без потери информации.

Наиболее важные на практике нормальные формы отношений основываются на фундаментальном в теории реляционных баз данных понятии функциональной зависимости.

## ****Функциональные зависимости****

**Определение 1.** Пусть R - это отношение, а Х и Y - произвольные подмножества множества атрибутов отношения R. Тогда Y функционально зависимо от Х, что в символическом виде записывается как X -> Y ⇔ ∀ значение множества Х связано в точности с одним значением множества Y.

Левая и правая стороны ФЗ будут называются детерминантом и зависимой частью соответственно.

**Определение 3.** ФЗ (X -> Y) тривиальная ⇔Y ⊆X.

**Определение 4**. Множество всех ФЗ, которые задаются данным множеством ФЗ S, называется замыканием S и обозначается символом S+.

Правила вывода определяются следующим образом:

1. Правило рефлексивности: (B ⊆A) ⇒(А -> В)

2. Правило дополнения: (A -> B) ⇒АС -> BC

3. Правило транзитивности: (А -> В) и (В -> С) ⇒(А -> C).

1. Правило самоопределения: А -> А.
2. Правило декомпозиции: (A -> BC) ⇒(А -> В) и (А -> С)
3. Правило объединения:(А -> В) и (А -> С) ⇒(А -> BC).

7. Правило композиции: (А -> В) и (С -> D) ⇒(АС -> BD).

1. Общая теорема объединения: (А->В)и(С->D)⇒(А(С–B)->BD).

**Определение 5.** Суперключ переменной-отношения R - это множество атрибутов переменной-отношения R, которое содержит в виде подмножества (но не обязательно собственного подмножества), по крайней мере, один потенциальный ключ.

**Определение 6.** Два множества ФЗ S1 и S2 эквивалентны тогда и только тогда, когда они являются покрытиями друг для друга, т. е. S1+ = S2+.

**Определение 7.** Множество ФЗ является неприводимым тогда и только тогда, когда оно обладает всеми перечисленными ниже свойствами.

* Каждая ФЗ этого множества имеет одноэлементную правую часть.
* Ни одна ФЗ множества не может быть устранена без изменения замыкания этого множества.
* Ни один атрибут не может быть устранен из левой части любой ФЗ данного множества без изменения замыкания множества

## ****Нормальные формы, основанные на функциональных зависимостях****

**Определение 1**. Переменная отношения находится в первой нормальной форме (1НФ) тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов.

**Определение 2.** Функциональная зависимость R.X -> R.Y называется полной, если атрибут Y не зависит функционально от любого точного подмножества X.

**Определение 3.** Функциональная зависимость R.X -> R.Y называется транзитивной, если существует такой атрибут Z, что имеются функциональные зависимости R.X -> R.Z и R.Z -> R.Y и отсутствует функциональная зависимость R.Z -> R.X.

**Определение 4.** Неключевым атрибутом называется любой атрибут отношения, не входящий в состав потенциального ключа (в частности, первичного).

**Определение 5.** Два или более атрибута взаимно независимы, если ни один из этих атрибутов не является функционально зависимым от других.

## ****2NF****

**Определение 6.** (В этом определении предполагается, что единственным ключом отношения является первичный ключ.) Отношение R находится во второй нормальной форме (2НФ) в том и только в том случае, когда оно находится в 1НФ, и каждый неключевой атрибут полностью зависит от первичного ключа.

## ****3NF****

## Отношение R находится в третьей нормальной форме (3НФ) в том и только в том случае, если оно находится в 2НФ, и каждый неключевой атрибут не является транзитивно зависимым от какого-либо ключа R.

## 5. Теория проектирования хранилищ данных. Основные принципы построения. ETL и ELT процессы

**Храни́лище да́нных** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) Data Warehouse) — предметно-ориентированная информационная база данных, специально разработанная и предназначенная для подготовки отчётов и бизнес-анализа с целью поддержки принятия решений в организации.

Любая передача внутри хранилищ данных — ETL (аббревиатура от Extract, Transform, Load).

Трехуровневая архитектура состоит из:

• Нижний уровень содержит сервер базы данных используемый для извлечения данных из множества различных источников.

• Средний уровень содержит сервер OLAP, который преобразует данные в структуру, подходящую для анализа и сложных запросов.

• Верхний уровень — уровень клиента, содержащий инструменты, используемые для высокоуровневого анализа данных, создания отчетов и анализа данных.

Есть два хранилища данных:

1) Подход Ральфа Кимбалла основывается на важности витрин данных, которые являются хранилиавми данных, облегчающие отчетность и анализ. При этом организация хранилища пространственная, где существуют «виртуальные» объекты. Коллекция витрин данных, которые могут быть пространственно разобщенны.

2) Подход Билла Инмона основывается на том, что хранилище данных является централизованным хранилищем всех корпоративных данных. При таком подходе сначала создают нормализованную модель хранилища данных, где объектами являются физически целостными. Затем создаются витрины размерных на основе модели хранилища. Это известно как нисходящий подход к хранилищу данных.

Схемы «звезда» и «снежинка» — это два способа структурировать хранилище данных. Схема типа «звезда» имеет централизованное хранилище данных, которое хранится в таблице фактов. Схема разбивает таблицу фактов на ряд денормализованных таблиц измерений. Таблица фактов содержит агрегированные данные, которые будут использоваться для составления отчетов, а таблица измерений описывает хранимые данные. Денормализованные проекты менее сложны, потому что данные сгруппированы. Таблица фактов использует только одну ссылку для присоединения к каждой таблице измерений. Более простая конструкция звездообразной схемы значительно упрощает написание сложных 14 запросов.

Схема типа «снежинка» отличается тем, что использует нормализованные данные. Нормализация означает эффективную организацию данных так, чтобы все зависимости данных были определены, и каждая таблица содержала минимум избыточности. Таким образом, отдельные таблицы измерений разветвляются на отдельные таблицы измерений.

ETL & ELT

ETL и ELT — два разных способа загрузки данных в хранилище.

ETL (Extract, Transform, Load) начала извлекают данные из пула источников данных. Данные хранятся во временной промежуточной базе данных. Затем выполняются операции преобразования, чтобы структурировать и преобразовать данные в подходящую форму для целевой системы хранилища данных. Затем структурированные данные загружаются в хранилище и готовы к анализу.

Инструменты ETL используются для интеграции данных, чтобы удовлетворить требованиям систем управления реляционными базами данных и/или традиционных хранилищ данных с поддержкой OLAP (online analytical processing, аналитической онлайн-обработки). Инструменты OLAP и запросы (SQL) требуют, чтобы массивы данных структурировались и стандартизировались при помощи серии преобразований, выполняемых до того, как данные попадут в хранилище.

ELT (Extract, Load, Transform) данные сразу же загружаются после извлечения из исходных пулов данных. Промежуточная база данных отсутствует, что означает, что данные немедленно загружаются в единый централизованный репозиторий. Данные преобразуются в системе хранилища данных для использования с инструментами бизнес-аналитики и аналитики.

Этапы ETL & ELT:

1) Процесс загрузки данных из пулла источников.

2) Процесс валидации (отвечает за выявление ошибок и пробелов в данных).

3) Процесс мэппинга данных.

4) Процесс агрегации данных. Этот процесс нужен из-за разности детализации данных в OLTP и OLAP системах. OLTP система может содержать несколько сумм для одного и того же набора элементов справочников, a OLAP-системы — это, по сути, полностью денормализованная таблица фактов и окружающие ее таблицы справочников.

5) Процесс выгрузки данных в целевую систему.

## 6 .Транзакции. Определение, свойства и уровни изоляции транзакций. Неблагоприятные эффекты, вызванные параллельным выполнением транзакций, и способы их устранения. Управление транзакциями и способы обработки ошибок

Транзакция — это последовательность операций, выполняемая как единое целое. Транзакции нужны для поддержания целостности сущностей.

Для поддержания целостности транзакция должна обладать четырьмя свойствами АСИД. Эти свойства называются также **ACID**-свойствами (in English):

* **А**томарность (**A**tomicity) Либо выполняется целиком, либо не выполняется вовсе.
* **С**огласованность (или Непротиворечивость) (**C**onsistency)

Не нарушает бизнес-логику и отношения между элементами данных.

* **И**золяция (**I**solation)

Результаты транзакции самодостаточны, они не зависят от предыдущих или следующих транзакций.

* **Д**олговечность (или Устойчивость) (**D**urability)

После заверщения, происходит фиксация транзакции, и становится невозможным откат до состояния до начала транзакции.

По признаку определения границ различают:

* Автоматические транзакции
* Неявные транзакции
* Явные транзакции

**Автоматические транзакции**

Режим автоматической фиксации транзакций является режимом управления транзакциями по умолчанию. В этом режиме каждая инструкция T-SQL выполняется как отдельная транзакция. Если выполнение инструкции завершается успешно, происходит фиксация; в противном случае происходит откат. Если возникает ошибка компиляции, то план выполнения пакета не строится и пакет не выполняется.

**Неявные транзакции** Если соединение работает в режиме неявных транзакций, то после фиксации или отката текущей транзакции автоматически начинает новую транзакцию. В этом режиме явно указывается только граница окончания транзакции с помощью инструкций COMMIT TRANSACTION и ROLLBACK TRANSACTION.

**Явные транзакции**

Для определения явных транзакций используются следующие инструкции:

* BEGIN TRANSACTION
* COMMIT TRANSACTION или COMMIT WORK
* ROLLBACK TRANSACTION или ROLLBACK WORK
* SAVE TRANSACTION

### **Управлением параллельным выполнением транзакций**

Когда множество пользователей одновременно пытаются модифицировать данные в базе данных, необходимо создать систему управления, которая защитила бы модификации, выполненные одним пользователем, от негативного воздействия модификаций, сделанных другими. Выделяют два типа управления параллельным выполнением:

* **Оптимистическое** управление параллельным выполнением. Похоже на Git
* **Пессимистичное** управление параллельным выполнением. Блокирование ресурса

## ****Проблемы****

## Проблема последнего изменения: Несколько пользователей изменяют одну и ту же строку, основываясь на ее начальном значении; тогда часть данных будет потеряна.

**Проблема грязного чтения (Dirty read):** Пользователь выполняет сложные операции обработки данных, требующие множественного изменения данных перед тем, как они обретут логически верное состояние. Если во время изменения данных другой пользователь будет считывать их, то может оказаться, что он получит логически неверную информацию.

**Проблема неповторяемого чтения:** Во время выполнения первой транзакции другая может внести в данные изменения, поэтому при повторном чтении первая транзакция получит уже иной набор данных, что приводит к нарушению их целостности или логической несогласованности.

**Проблема чтения фантомов (Phantom):** Одна транзакция выбирает данные из таблицы, а другая вставляет или удаляет строки до завершения первой. Выбранные из таблицы значения будут некорректны.

### Уровни изоляции транзакций

**Уровень 0** – запрещение «загрязнения» данных. Этот уровень требует, чтобы изменять данные могла только одна транзакция; если другой транзакции необходимо изменить те же данные, она должна ожидать завершения первой транзакции.

**Уровень 1** – запрещение «грязного» чтения. Если транзакция начала изменение данных, то никакая другая транзакция не сможет прочитать их до завершения первой.

**Уровень 2** – запрещение неповторяемого чтения. Если транзакция считывает данные, то никакая другая транзакция не сможет их изменить. Таким образом, при повторном чтении они будут находиться в первоначальном состоянии.

**Уровень 3** – запрещение фантомов. Если транзакция обращается к данным, то никакая другая транзакция не сможет добавить новые или удалить имеющие строки, которые могут быть считаны при выполнении транзакции. Реализация этого уровня блокирования выполняется путем использования блокировок диапазона ключей. Подобная блокировка накладывается не на конкретные строки таблицы, а на строки, удовлетворяющие определенному логическому условию.

Чем выше уровень изоляции – тем меньше параллелизма в системе, так как процессы будут выстраиваться в очередь к ресурсу.

Способы обработки ошибок. Если ошибка делает невозможным успешное выполнение транзакции, SQL Server автоматически выполняет ее откат и освобождает ресурсы, удерживаемые транзакцией. Если сетевое соединение клиента с SQL Server разорвано, то после того, как SQL Server получит уведомление от сети о разрыве соединения, выполняется откат всех необработанных транзакций для этого соединения. В случае сбоя клиентского приложения, выключения либо перезапуска клиентского компьютера соединение также будет разорвано, а SQL Server выполнит откат всех необработанных транзакций после получения уведомления о разрыве от сети. Если клиент выйдет из приложения, выполняется откат всех необработанных транзакций. На случай возникновения ошибок код приложения должен содержать исправляющее действие: COMMIT или ROLLBACK. Эффективным средством для обработки ошибок, включая ошибки транзакций, является конструкция языка Transact-SQL TRY...CATCH

## 7.Блокировки. Определение, свойства, иерархии, гранулярность и взаимоблокировки, алгоритмы обнаружения взаимоблокировок

Блокировка — это механизм, с помощью которого компонент Database Engine синхронизирует одновременный доступ нескольких пользователей к одному фрагменту данных.

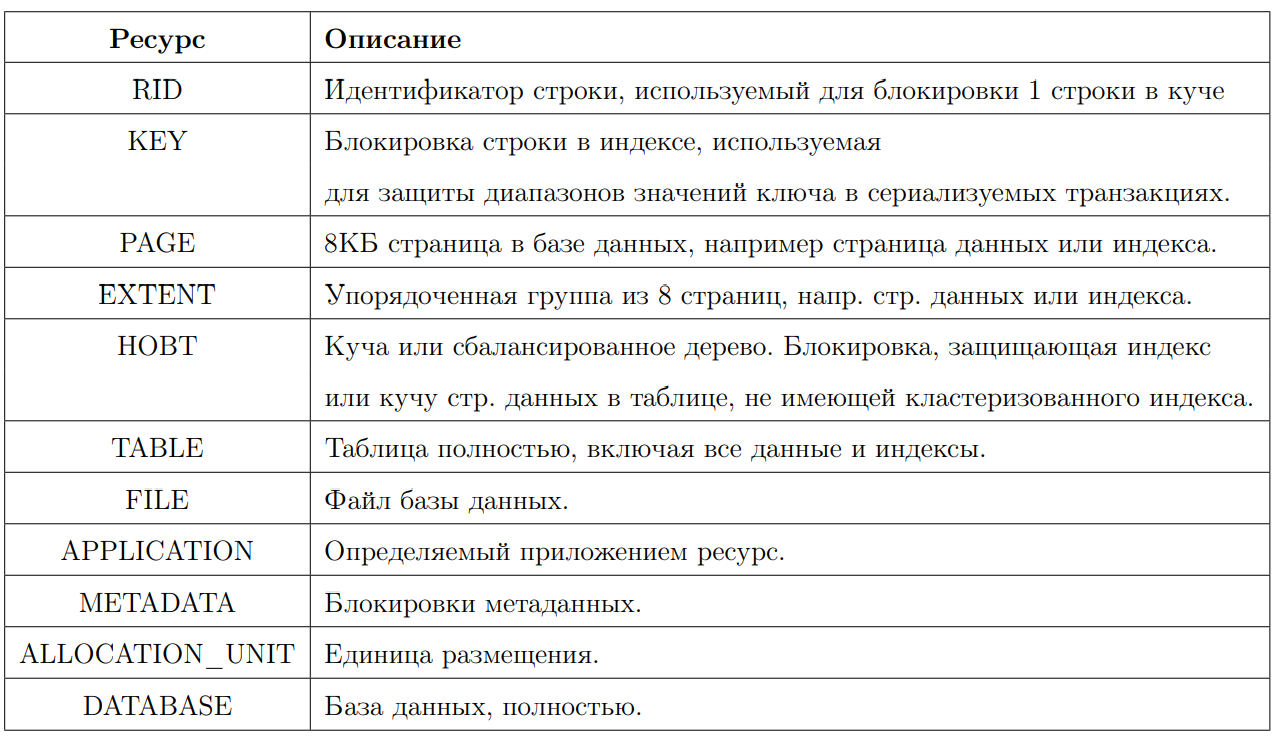
### Свойства

* Гранулярность (размер блокировки)
* Режим (тип блокировки)
* Продолжительность

Иерархии:

Компонент Database Engine часто получает блокировки на нескольких уровнях гранулярности одновременно, чтобы полностью защитить ресурс. Такая группа блокировок на нескольких уровнях гранулярности называется иерархией блокировки.

Гранулярность : Гранулярность блокировки определяет, какой ресурс блокируется в одной попытке блокировки.



Взаимоблокировки

Взаимоблокировки или тупиковые ситуации (deadlocks) возникают тогда, когда одна из транзакций не может завершить свои действия, поскольку вторая транзакция заблокировала нужные ей ресурсы, а вторая в то же время ожидает освобождения ресурсов первой транзакцией. Как SQL Server обнаруживает взаимоблокировки? Обнаружение взаимоблокировки выполняется потоком диспетчера блокировок, который периодически производит поиск по всем задачам в экземпляре компонента Database Engine.

Следующие пункты описывают процесс поиска:

1) Значение интервала поиска по умолчанию составляет 5 секунд.

2) Если диспетчер блокировок находит взаимоблокировки, интервал обнаружения взаимоблокировок снижается с 5 секунд до 100 миллисекунд в зависимости от частоты взаимоблокировок.

3) Если поток диспетчера блокировки прекращает поиск взаимоблокировок, компонент Database Engine увеличивает интервал до 5 секунд.

4) Если взаимоблокировка была только что найдена, предполагается, что следующие потоки, которые должны ожидать блокировки, входят в цикл взаимоблокировки. Первая пара 22 элементов, ожидающих блокировки, после того как взаимоблокировка была обнаружена, запускает поиск взаимоблокировок вместо того, чтобы ожидать следующий интервал обнаружения взаимоблокировки. Например, если текущее значение интервала равно 5 секунд и была обнаружена взаимоблокировка, следующий ожидающий блокировки элемент немедленно приводит в действие детектор взаимоблокировок. Если этот ожидающий блокировки элемент является частью взаимоблокировки, она будет обнаружена немедленно, а не во время следующего поиска взаимоблокировок.

5) Компонент Database Engine обычно выполняет только периодическое обнаружение взаимоблокировок. Так как число взаимоблокировок, произошедших в системе, обычно мало, периодическое обнаружение взаимоблокировок помогает сократить издержки от взаимоблокировок в системе.

6) Если монитор блокировок запускает поиск взаимоблокировок для определенного потока, он идентифицирует ресурс, ожидаемый потоком. После этого монитор блокировок находит владельцев определенного ресурса и рекурсивно продолжает поиск взаимоблокировок для этих потоков до тех пор, пока не найдет цикл. Цикл, определенный таким способом, формирует взаимоблокировку.

7) После обнаружения взаимоблокировки компонент Database Engine завершает взаимоблокировку, выбрав один из потоков в качестве жертвы взаимоблокировки. Компонент Database Engine прерывает выполняемый в данный момент пакет потока, производит откат транзакции жертвы взаимоблокировки и возвращает приложению ошибку 1205. Откат транзакции жертвы взаимоблокировки снимает все блокировки, удерживаемые транзакцией. Это позволяет транзакциям потоков разблокироваться, и продолжить выполнение. Ошибка 1205 жертвы взаимоблокировки записывает в журнал ошибок сведения обо всех потоках и ресурсах, затронутых взаимоблокировкой.

## 8.Журнализация. Операции журнала транзакций и его логическая и физическая архитектуры. Модели восстановления. Метаданные

Базы данных SQL Server содержат **файлы трех типов**:

* Первичные файлы данных (содержат сведения, необходимые для запуска базы данных, и ссылки на другие файлы в базе данных; в каждой базе данных имеется один первичный файл данных)
* Вторичные файлы данных (необязательные определяемые пользователем файлы данных; данные можно распределить по нескольким дискам, поместив каждый файл на другой диск; обеспечивает распределение нагрузки по чтению и записи по нескольким дискам)
* Файлы журналов (журнал содержит информацию для восстановления базы данных; для каждой базы данных должен существовать хотя бы один файл журнала; используется для записи всех изменений, вносимых в данные, до того как изменения будут реально записаны в файлы данных)

**Журнализация изменений** - это функция СУБД, которая сохраняет информацию, необходимую для восстановления базы данных в предыдущее согласованное состояние в случае логических или физических отказов.

Журнал транзакций поддерживает следующие **операции**:

* Восстановление отдельных транзакций
* Восстановление всех незавершенных транзакций при запуске SQL Server
* Накат восстановленной базы данных, файла, файловой группы или страницы до момента сбоя
* Поддержка репликации транзакций
* Поддержка решений с резервными серверами

Логическая архитектура

На логическом уровне журнал транзакций состоит из последовательности записей. Двумя основными типами записи Log-файла являются:

* 1. код выполненной логической операции.

• Для наката логической операции выполняется эта операция.

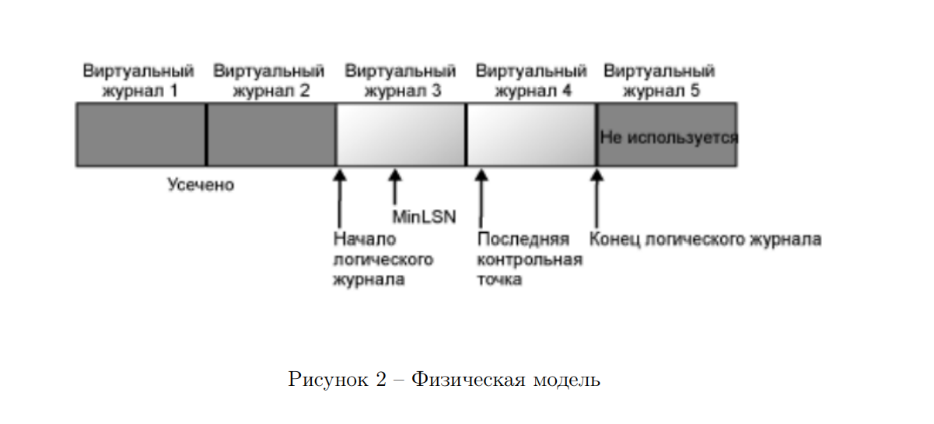
• Для отката логической операции выполняется логическая операция, обратная зарегистрированной.

* 1. исходны и результирующий образ измененных данных.

• Для наката операции применяется результирующий образ.

• Для отката операции применяется исходный образ.

Исходный образ записи – это копия данных до выполнения операции, а результирующий образ – копия данных после ее выполнения.

Физическая архитектура На физическом уровне журнал транзакций состоит из одного или нескольких физических файлов. Каждый физический файл журнала разбивается на несколько виртуальных файлов журнала (VLF). VLF не имеет фиксированного размера. Не существует также и определенного числа VLF, приходящихся на один физический файл журнала. Компонент Database Engine динамически определяет размер VLF при создании или расширении файлов журнала. Компонент Database Engine стремится обслуживать небольшое число VLF. Администраторы не могут настраивать или устанавливать размеры и число VLF. Журнал транзакций является оборачиваемым файлом. Рассмотрим пример. Пусть база данных имеет один физический файл журнала, разделенный на четыре виртуальных файла журнала. При создании базы данных логический файл журнала начинается в начале физи25 ческого файла журнала. Новые записи журнала добавляются в конце логического журнала и приближаются к концу физического файла журнала. Усечение журнала освобождает любые виртуальные журналы, все записи которых находятся перед минимальным регистрационным номером восстановления в журнале транзакций (MinLSN). MinLSN является регистрационным номером самой старой записи, которая необходима для успешного отката на уровне всей базы данных. Журнал транзакций рассматриваемой в данном примере базы данных будет выглядеть примерно так же, как на следующей иллюстрации.

Часть журнала, начинающаяся с номера MinLSN и заканчивающаяся последней записью, называется активной частью журнала, или активным журналом. Этот раздел журнала необходим для выполнения полного восстановления базы данных. Ни одна часть активного журнала не может быть усечена. Все записи журнала до номера MinLSN должны быть удалены из частей журнала.

**Модель восстановления** — это свойство базы данных, которое управляет процессом регистрации транзакций, определяет, требуется ли для журнала транзакций резервное копирование, а также определяет, какие типы операций восстановления доступны.

Есть три модели восстановления:

* Модель полного восстановления (FULL) – все транзакции записываются в журнал транзакций, но они остаются в журнале транзакций и после завершения транзакции. Записи остаются в журнале транзакций до тех пор, пока не будет сделана резервная копия журнала.
* Модель восстановления с неполным протоколированием (BULK\_LOGGED) – минимизирует использование пространства журнала транзакций при операциях с неполным протоколированием, подобных BULK INSERT, SELECT INTO или CREATE INDEX. Функциональность этой модели подобна полной модели восстановления за исключением того, что записи в журнал транзакции минимально протоколируются при выполнении указанных операций.
* Простая модель восстановления (SIMPLE) – каждая транзакция по-прежнему записывается в журнал транзакций. Записи журналов транзакций будут автоматически удаляться , если используется простая модель восстановления. Этот процесс удаления происходит для всех завершенных транзакций, когда устанавливается контрольная точка.

SQL Server поддерживает восстановление данных на следующих уровнях:

* База данных (полное восстановление базы данных)
* Файл данных (восстановление файла)
* Страница данных (восстановление страницы)

**Метаданные** — информация о другой информации, или данные, относящиеся к дополнительной информации о содержимом или объекте. Метаданные раскрывают сведения о признаках и свойствах, характеризующих какие-либо сущности, позволяющие автоматически искать и управлять ими в больших информационных потоках. Описание того, какие типы объектов хранятся в базе данных, какие у них есть поля (аттрибуты, элементы), описание зависимостей между объектами (формализованные структурированные описания системы).

Тремя наиболее используемыми классами метаданных являются:

* ***Внутренние метаданные***, описывающие структуру или составные части вещи, то, чем вещь является. Например, формат и размер файла.
* ***Административные метаданные***, требующиеся для процессов обработки информации, назначение вещи. Например, информация об авторе, редакторе, дата публикации и т. п.
* ***Описательные метаданные***, которые описывают природу вещи, её признаки. Например, набор связанных с информацией категорий, ссылки на другие вещи, связанные с данной.

Метаданные содержат, как правило, кроме чисто системных, еще и вспомогательные, пользовательские параметры: заголовок таблицы, заголовки атрибутов, единицы измерения и другие. Метаданные применяются обычно для построения интерфейса, однако их структурная роль не менее важна. Метаданные представляют собой пул информации о данных, начиная с технических деталей, таких как размер записи базы данных, и заканчивая списком бизнес-целей, для которых предназначены различные фрагменты данных. Описания направляют конечных пользователей к нужным бизнес-данным и затем помогают им понять, что значат данные и насколько они являются оперативными.

## 9.Безопасность и Аудит. Ключевые понятия и участники системы безопасности. Модели управления доступом

Audit – функциональность системы безопасности, которая может отслеживать практически любое действие с сервером или базой данных (выполняемое пользователями) и записывать эти действия в файловую систему или системный журнал Windows.

SQL Server обеспечивает защиту данных от неавторизированного доступа и от фальсификации. Основными функциями безопасности SQL Server являются:

1.проверка подлинности (аутентификация) – процедура проверки соответствия некоего лица и его учетной записи в компьютерной системе

2) авторизация — это предоставление лицу прав на какие-то действия в системе.

Дополнительными функциями безопасности SQL Server являются:

1) шифрование, 2) контекстное переключение, 3) олицетворение, 4) встроенные средства управления ключами.

В основе системы безопасности SQL Server лежат три понятия: 1) участники системы безопасности (Principals); 2) защищаемые объекты (Securables), 3) система разрешений (Permissions).

**Участники системы безопасности**

Область влияния участника зависит:

* от области его определения:
  + Операционная система
  + СУБД
  + База данных
* от количества пользователей области:
  + Индивидуальный (неделимый) участник, например, логин в ОС
  + Коллективный участник - группа пользователей ОС

**Защищаемые объекты:**

Защищаемые объекты – это ресурсы, доступ к которым регулируется системой авторизации.

К областям защищаемых объектов относятся:

* Сервер - управление учетными записями подключения (login)
* База данных - управление:
  + учетными записями пользователей (user), ролями (role)
* Схема - управление разрешениями на объекты БД: функции (function), процедуры (stored procedure),таблицы (table),представления (view)

**Система разрешений:**

У субъекта системы есть только один путь получения доступа к объектам - иметь назначенные непосредственно или опосредовано разрешения.

* При непосредственном управлении разрешениями они назначаются субъекту явно
* При опосредованном разрешения назначаются через членство в группах, ролях или наследуются от объектов, лежащих выше по цепочке иерархии.

Управление разрешениями производится путем выполнения инструкций языка DCL:

* GRANT (разрешить)
* DENY (запретить)
* REVOKE (отменить)

**Пользователи и группы пользователей:**

Работа с пользователем: Создать пользователя, Удалить пользователя, Добавить пользователя в группу

**Роли:**

Роль - группа привилегий. Роль можно назначить другой роли (многоуровневость). Замыкающую роль создать нельзя.

В Oracle, в отличие от Postgres, под каждого пользователя создаётся своя схема, в которую он может писать, то есть в БД нельзя просто сходить-почитать.

Права можно выдавать для всей группы и у всей группы забирать. Для этого создаётся роль, представляющая группу, а затем членство в этой группе выдаётся ролям индивидуальных пользователей. Для настройки групповой роли сначала нужно создать саму роль:

**Модели управления доступом:**

* MAC — мандатная модель управления доступом
  + Доступ субъекта к объекту определяется его уровнем доступа: уровень доступа субъекта должен быть не ниже уровня секретности объекта. Объекты наследуют метки пользователей, создавших их. Используется в ОС, файловых системах.
  + Достоинства: простата построения общей схемы доступа, простата администрирования Недостатки: проблема разграничения пользователей одного уровня, пользователь не может назначать доступ к объекту
* DAC — прямое управление доступом
  + Доступ субъекта к объекту определяется наличием субъекта в списке доступа (ACL) объекта.
  + Достоинства: простата реализации, гибкость (пользователь может описать доступ к своим ресурсам) Недостатки: излишняя детализированность (приводит к запутанности), сложность администрирования, пользователь может допустить ошибку при назначении прав
* RBAC — ролевая модель управления доступом
  + Доступ субъекта к объекту определяется наличием у субъекта роли, содержащей полномочия, соответствующие запрашиваемому доступу.
  + Формирование ролей: Ролевое разграничение доступа позволяет реализовать гибкие, изменяющиеся динамически в процессе функционирования компьютерной системы правила; Как правило, данный подход применяется в системах защиты СУБД; Ролевой подход часто используется в системах, для пользователей которых чётко определён круг их должностных полномочий и обязанностей;
  + Достоинства: повышение операционной эффективности (снизить потребность в смене документов и паролей, позволяет организациям быстро добавлять и изменять роли, сокращение вероятности ошибки при назначении прав), повышение соответствия (удовлетворение нормативным и законодательным требованиям конфиденциальности), прозрачность и контроль, снижение затрат, снижение риска нарушений и утечки данных Недостатки: одно бизнес-правило «размазывается» среди множества ролей и становится неочевидным, что усложняет понимание такого правила и его поддержку, начинается взрывной рост числа ролей, что значительно усложняет управление ими
* АВАС — атрибутивная модель управления доступом
  + Доступ субъекта к объекту определяется динамически на основании анализа политик, учитывающих значения атрибутов субъекта, объекта и окружения. Характерна для модели звёздочки.
  + Политика на основе атрибутов: Делает управление доступом более эффективным, уменьшая сложность нормативных требований; Одна и та же политика (набор корпоративных правил) может использоваться в разных системах. Это помогает управлять согласованностью доступа к ресурсам в пределах одной компании; Централизованное управление доступом предполагает единственный авторитетный источник для правил доступа;
  + Достоинства: нет ограничений на сложность бизнес-правил, бизнес-правило не «размазывается» по системе, что делает его понимание и поддержку достаточно простыми, не происходит взрывного роста числа условий, что упрощает их сопровождение, легко автоматизируется (распределение ролей). Недостатки: отсутствие формальных определений моделей и сложность администрирования, практически неразрешимость задач вычисления значений разнородных атрибутов в процессе конструирования и выполнения политик

## 10MPP системы. Распределенное и колоночное хранение. Распределенные вычисления, модель MapReduce. Обеспечение отказоустойчивости.

Существует два вида масштабирования серверов БД.

1. Вертикальное масштабирование.
2. Горизонтальное масштабирование.

При вертикальном масштабировании увеличиваются ресурсы (улучшается железо) одного сервера (например, добавляются плашки оперативной памяти). Отсюда возникает проблема вертикального масштабирования — есть пределы этого улучшения (например, количество слотов оперативной памяти).

При горизонтальном масштабировании увеличивается количество серверов (создается кластер). Пределы этого увеличения тоже существуют: при достижении определенного числа серверов в кластере время диспетчеризации будет превышать время работы сервера (аналогия с 4 лабораторной работой по АА).

Одним из примеров горизонтального масштабирования является *MPP* (in English — Massive Parallel Processing) (in Russian — Массивно-параллельные вычисления).

*MPP —* структура, в которой вместо одного перегруженного сервера используется несколько слегка нагруженных.

**Vertica**

Vertica представляет собой колоночную СУБД с поддержкой MPP.

Основные свойства:

* Отсутствует единая точка отказа (если откажет любой узел, все остальные продолжат работать).
* Каждый узел является независимым и самостоятельным.
* Отсутствует единая для всей системы точка подключения (подключаться можно к любому).
* Узлы инфраструктуры дублируются.
* Данные на узлах кластера автоматически копируются.

Как Vertica хранит данные?

Логические единицы хранения информация — таблицы, схемы, представления.

Физическая единица хранения информации — проекция.

**Обеспечение отказоустойчивости**

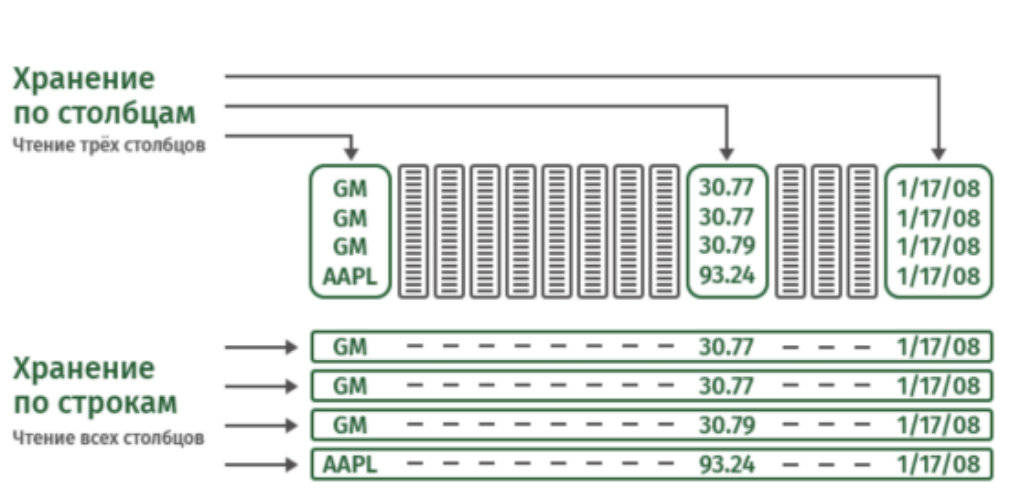
*Репликация* — это передача изменений в информации, произведенных в одном узле системы баз данных, на другие узлы

*Key Safety* — механизм отказоустойчивости, с помощью которого можно настраивать количество репликаций для каждой проекции (по сути *Key Safety —* некоторый параметр, целое число). Проекции, полученные с помощью репликации называются *дружественными* (in English *— Buddy Projection*).

1. ***Key Safery = 0 —*** репликации нет.
2. *Key Safety* = 1 — репликация на один соседний узел.
3. *Key Safety* = 2 — репликация на 2 соседних узла.

**В чем выгода колоночного хранения?**

Если мы читаем строки, то, например, для выполнения команды SELECT 1,11,15 FROM table1 нам придется читать всю таблицу. Это огромный объем информации. В данном случае колоночный подход выгоднее. Он позволяет считать только три нужных нам столбца, экономя память и время.



MapReduce — это способ агрегации больших хранилищ данных. Стадия Map выполняется на множестве серверных узлов распределенной обработки. Она обычно исполняет некую задачу на каждом распределенном серверном узле для выборки данных из узлов данных и может дополнительно преобразовывать или предварительно обрабатывать данные, когда они находятся на распределенной серверном узле. Стадия Reduce выполняется на одном или более серверных узлов конечной обработки и консолидирует все результаты от стадий Map в один конечный набор результатов, используя иножество различных алгоритмов объединения. Одно из основных преимуществ MapReduce — он обеспечивает горизонтальное масштабирование (scale out) вместо вертикального (scale up). Иначе говоря, для масштабирования вы просто добавляете обычные серверные узлы, а не приобретаете более эффективное оборудование для одного основного серверного узла. Горизонтальное масштабирование, в целом, является более дешевым и гибким выбором, поскольку используется недорогое стандартное аппаратное обеспечение, тогда как вертикальное масштабирование обычно гораздо дороже, потому что стоимость аппаратного обеспечения растет по экспоненте по мере увеличения его сложности.

## 11.In-Memory базы данных. Преимущества и недостатки. Примеры использования.

***In-memory Database*** — база данных в оперативной памяти.

Преимущества:

* Высокая производительность.
* Структуры данных в памяти.
* Поддержка высокоуровневыми языками.
* Репликация (передача изменений в информации, произведенных в одном узле системы баз данных, на другие узлы).

Недостатки:

* Надежность (уязвимость к отказам, так как оперативная память — энергозависимая память). Существуют ряд методов, которые позволяют снизить этот риск: снэпшоты, логи транзакций, NVRAM, NVDIMM
* При частом обновлении данных в БД неэффективно, так как постоянно будет происходить двойное обращение к памяти (сначала к *In-memory Database,* потом к основной БД).

***Redis (in English — REmote DIctionary Server) —*** сервер БД, хранилище структур данных в оперативной памяти в формате “ключ-значение”.

Типы данных, поддерживаемые *Redis:*

* Строки.
* Списки.
* Хеш-таблицы.
* Множества.
* Упорядоченные множества.

**Примеры использования**

* Кэширование.
* Управление сессиями.
* Таблицы лидеров в режиме реального времени.
* Очереди.
* Чат и обмен сообщениями.
* Ограничение интенсивности.

## 12. Инструкции языка описания данных, инструкции языка обработки данных, инструкции безопасности, инструкции управления транзакциями.

**Инструкции языка описания данных**

* CREATE создает объект БД (саму базу, таблицу, представление, пользователя и т. д.)
* АLTER изменяет объект
* DROP удаляет объект
* ENABLE TRIGGER включает триггер
* DISABLE TRIGGER отключает триггер
* TRUNCATE TABLE удаляет все строки в таблице, не записывая в журнал удаление отдельных строк
* UPDATE STATISTICS обновляет статистику оптимизации запросов для таблицы или индексированного представления

**Инструкции языка обработки данных**

* SELECT считывает данные, удовлетворяющие заданным условиям
* INSERT добавляет новые данные
* UPDATE изменяет существующие данные
* DELETE удаляет данные
* MERGE выполняет операции вставки, обновления или удаления для целевой таблицы на основе результатов соединения с исходной таблицей
* BULK INSERT выполняет импорт файла данных в таблицу или представление базы данных в формате, указанном пользователем
* READTEXT считывает значения text, ntext или image из столбцов типа text, ntext или image начиная с указанной позиции
* WRITETEXT обновляет и заменяет все поле text, ntext или image
* UPDATETEXT обновляет часть поля text, ntext или image

**Инструкции безопасности**

* GRANT предоставляет пользователю разрешения на определенные операции с объектом
* REVOKE отзывает ранее выданные разрешения
* DENY задает запрет, имеющий приоритет над разрешением
* АDD SIGNATURE добавляет цифровую подпись для хранимой процедуры, функции, сборки или триггера
* OPEN MASTER KEY открывает главный ключ в текущей базе данных
* CLOSE MASTER KEY закрывает главный ключ в текущей базе данных
* OPEN SYMMETRIC KEY расшифровывает симметричный ключ и делает его доступным для использования
* CLOSE SYMMETRIC KEY закрывает симметричный ключ или все симметричные ключи, открытые в текущем сеансе
* EXECUTE AS контекст выполнения сеанса переключается на заданное имя входа и имя пользователя
* REVERT переключает контекст выполнения в контекст участника, вызывавшего последнюю инструкцию EXECUTE AS
* SETUSER позволяет члену предопределенной роли сервера sysadmin или члену предопределенной роли базы данных db\_owner олицетворять другого пользователя

**Инструкции управления транзакциями**

* BEGIN DISTRIBUTED TRANSACTION запускает распределенную транзакцию, управляемую координатором распределенных транзакций
* BEGIN TRANSACTION отмечает начальную точку явной локальной транзакции
* COMMIT TRANSACTION отмечает успешное завершение явной или неявной транзакции
* COMMIT WORK действует так же, как и инструкция COMMIT TRANSACTION
* ROLLBACK TRANSACTION откатывает явные или неявные транзакции до начала или до точки сохранения транзакции
* ROLLBACK WORK действует так же, как и инструкция ROLLBACK TRANSACTION
* SAVE TRANSACTION устанавливает точку сохранения внутри транзакции

## 13. Объекты базы данных: функции, процедуры, триггеры и курсоры

**Функции**

По поведению различают **детерминированные** и **недетерминированные** функции.

* Функция является **детерминированной**, если при одном и том же заданном входном значении она всегда возвращает один и тот же результат.
* Функция является **недетерминированной**, если она может возвращать различные значения при одном и том же заданном входном значении.

Все пользовательские функции всегда недетерминированные

По типу возвращаемого значения функции делятся:

* скалярные
* подставляемые табличные
* многооператорные

**Скалярная функция:**

Скалярные пользовательские функции обычно используются в списке столбцов инструкции SELECT и в предложении WHERE.

Синтаксис:

CREATE FUNCTION [имя-схемы.]имя-фукнции([список-объявлений-параметров])

RETURNS скалярный-тип-данных

[ WITH список-опций-функций]

[AS]

BEGIN

тело-функции

RETURN скалярное-выражение

END [;]

**Подставляемая табличная функция**

Тело подставляемой табличной функции фактически состоит из единственной инструкции SELECT.

Синтаксис создания:

CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )

RETURNS TABLE

[ WITH список-опций-функций ]

[ AS ]

RETURN [ ( ] выражение-выборки [ ) ]

END [ ; ]

Синтаксис вызова:

SELECT \*

FROM [имя-схемы.]имя-функции

**Многооператорная табличная функция**

В многооператорной табличной функции команды T-SQL располагаются внутри блока BEGIN-END. Т.к. блок может содержать несколько инструкций SELECT, в предложении RETURNS необходимо явно определить таблицу, которая будет возвращаться. Поскольку оператор RETURN в многооператорной табличной функции всегда возвращает таблицу, заданную в предложении RETURNS, он должен выполняться без аргументов.

Синтаксис:

CREATE FUNCTION [ имя-схемы. ] имя-функции ( [ список-объявлений-параметров ] )

RETURNS @имя-возвращаемой-переменной

TABLE определение-таблицы

[ WITH список-опций-функций ]

[ AS ]

BEGIN

RETURN

END [ ; ]

"Таблица"(табличная переменная), которая используется в функции по сути не является таблицей, ей потом нельзя после функции пользоваться как таблицей, но можно создать таблицу на основе таблицы из функции. После ретерна функции нельзя обновлять таблицу.

**Хранимые процедуры**

Хранимая процедура – именованный объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. У них могут быть входные и выходные параметры и локальные переменные. В хранимых процедурах могут выполняться операторы DDL, DML, TCL, DCL.

Процедуры можно создавать для:

* постоянного использования,
* для временного использования в одном сеансе (локальная временная процедура),
* для временного использования во всех сеансах (глобальная временная процедура)

Создание:

CREATE PROCEDURE [ имя-схемы. ] имя-процедуры [ список-объявлений-параметров ]

[ WITH список-опций-процедуры ]

[ FOR REPLICATION ]

AS

тело-процедуры

[ ;]

Если имя схемы не указано при создании процедуры, то автоматически назначается схема по умолчанию для пользователя, который создает процедуру. Список объявлений параметров является необязательным. Параметрами процедуры могут быть любые типы данных, за исключением table.

При выполнении процедуры в первый раз она компилируется, при этом определяется оптимальный план получения данных. При последующих вызовах процедуры может быть повторно использован уже созданный план, если он еще находится в кэше планов компонента Database Engine.

Одна процедура может вызывать другую. Уровень вложенности процедур может достигать 32.

Вызов процедуры:

CALL имя-процедуры([параметры])

**Триггеры**

**Триггер** - это хранимая процедура особого типа, которая выполняет одну или несколько инструкций T-SQL в ответ на событие из области действия сервера или базы данных.

По **типу событий** делятся на 2 класса:

* DDL
* DML

**Триггер DDL**

**Триггер DDL** может активироваться, если выполняется такая инструкция, как ALTER SERVER CONFIGURATION, или если происходит удаление таблицы с использованием команды DROP TABLE.

Все события (кроме ALTER\_SERVER\_CONFIGURATION) разделены на две группы:

* события уровня базы данных (DDL\_DATABASE\_LEVEL\_EVENTS)
* события уровня сервера(DDL\_SERVER\_LEVEL\_EVENTS)

Триггеры DDL не ограничены областью схемы.

Пример

CREATE TRIGGER safety

ON DATABASE

FOR DROP\_TABLE, ALTER\_TABLE AS

PRINT 'You must disable Trigger "safety" to drop or alter tables!'

ROLLBACK;

**Триггер DML**

Для поддержания согласованности и точности данных используются декларативные и процедурные методы. Триггеры применяются в следующих случаях:

* если использование методов декларативной целостности данных не отвечает функциональным потребностям приложения;
* если необходимо каскадное изменение через связанные таблицы в базе данных;
* если база данных денормализована и требуется способ автоматизированного обновления избыточных данных в нескольких таблицах;
* если необходимо сверить значение в одной таблице с неидентичным значением в другой таблице;
* если требуется вывод пользовательских сообщений и сложная обработка ошибок.

Классы DML-триггеров:

* **INSTEAD OF**. Триггеры этого класса выполняются в обход действий, вызывавших их срабатывание, заменяя эти действия.
* **AFTER/BEFORE.** Триггеры этого класса исполняются после или до действия, вызвавшего срабатывание триггера. Они считаются классом триггеров по умолчанию.

Синтаксис DML-триггера

CREATE TRIGGER имя\_триггера

ON имя\_таблицы\_или\_представления

[ WITH ENCRYPTION ]

класс\_триггера тип(ы)\_триггера

[ WITH APPEND ]

[ NOT FOR REPLICATION ]

AS sql\_инструкции

Различия между DML и DDL триггерами:

| **Характеристика** | **INSTEAD OF** | **AFTER/BEFORE** |
| --- | --- | --- |
| Объект, к которому можно привязать триггер | Таблица или представление. Триггеры, привязанные к представлению, расширяют список типов обновления, которые может поддерживать представление | Таблица. |
| AFTER-триггеры срабатывают при модификации данных в таблице в ответ на модификацию представления |  |  |
| Допустимое число триггеров | В таблице или представлении допускается не больше одного |  |
| триггера в расчете на одно действие. Можно определять представления для других представлений, каждое со своим собственным INSTEAD OF-триггером | К таблице можно привязать несколько AFTER- триггеров |  |
| Порядок исполнения | Поскольку в таблице или представлении допускается не больше |  |
| одного такого триггера в расчете на одно событие, порядок не имеет смысла | Можно определять триггеры, срабатывающие первым и |  |
| последним. Для этого служит системная хранимая процедура |  |  |
| sp\_settriggerorder. Порядок срабатывания других триггеров, |  |  |
| привязанных к таблице, случаен |  |  |

**Курсоры**

Операции в реляционной базе данных выполняются над множеством строк. Набор строк, возвращаемый инструкцией SELECT, содержит все строки, которые удовлетворяют условиям, указанным в предложении WHERE. Такой полный набор строк, возвращаемых инструкцией, называется результирующим набором. Приложения, особенно интерактивные, не всегда эффективно работают с результирующим набором как с единым целым. Им нужен механизм, позволяющий обрабатывать одну строку или небольшое их число за один раз.

По области видимости имени курсора различают:

* Локальные курсоры (LOCAL). Область курсора локальна по отношению к пакету, хранимой процедуре или триггеру, в которых этот курсор был создан. Курсор неявно освобождается после завершения выполнения пакета, хранимой процедуры или триггера, за исключением случая, когда курсор был передан параметру OUTPUT. В этом случае курсор освобождается при освобождении всех ссылающихся на него переменных или при выходе из области видимости.
* Глобальные курсоры (GLOBAL). Область курсора является глобальной по отношению к соединению. Имя курсора может использоваться любой хранимой процедурой или пакетом, которые выполняются соединением. Курсор неявно освобождается только в случае разрыва соединения. Глобальность курсора позволяет создавать его в рамках одной хранимой процедуры, а вызывать его из совершенно другой процедуры, причем передавать его в эту процедуру необязательно

По типу курсора различают:

* Статические курсоры (STATIC). Создается временная копия данных для использования курсором. Все запросы к курсору обращаются к указанной временной таблице в базе данных tempdb, поэтому изменения базовых таблиц не влияют на данные, возвращаемые выборками для данного курсора, а сам курсор не позволяет производить изменения.
* Динамические курсоры (DYNAMIC). Отображают все изменения данных, сделанные в строках результирующего набора при просмотре этого курсора. Значения данных, порядок, а также членство строк в каждой выборке могут меняться. Параметр выборки ABSOLUTE динамическими курсорами не поддерживается.
* Курсоры, управляемые набором ключей (KEYSET). Членство или порядок строк в курсоре не изменяются после его открытия. Набор ключей, однозначно определяющих строки, встроен в таблицу в базе данных tempdb с именем keyset.
* Быстрые последовательные курсоры (FAST\_FORWARD). Параметр FAST\_FORWARD указывает курсор FORWARD\_ONLY, READ\_ONLY, для которого включена оптимизация производительности. Параметр FAST\_FORWARD не может указываться вместе с параметрами SCROLL или FOR\_UPDATE.

По способу перемещения по курсору различают:

* Последовательные курсоры (FORWARD\_ONLY). Курсор может просматриваться только от первой строки к последней. Поддерживается только параметр выборки FETCH NEXT. Если параметр FORWARD\_ONLY указан без ключевых слов STATIC, KEYSET или DYNAMIC, то курсор работает как DYNAMIC. Если не указан ни один из параметров FORWARD\_ONLY или SCROLL, а также не указано ни одно из ключевых слов STATIC, KEYSET или DYNAMIC, то по умолчанию задается параметр FORWARD\_ONLY. Курсоры STATIC, KEYSET и DYNAMIC имеют значение по умолчанию SCROLL.
* Курсоры прокрутки (SCROLL). Перемещение осуществляется по группе записей как вперед, так и назад. В этом случае доступны все параметры выборки (FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE). Параметр SCROLL не может указываться вместе с параметром для FAST\_FORWARD.

По способу распараллеливания курсоров различают:

* READ\_ONLY. Содержимое курсора можно только считывать.
* SCROLL\_LOCKS. При редактировании данной записи вами никто другой вносить в нее изменения не может. Такую блокировку прокрутки иногда еще называют «пессимистической» блокировкой.
* OPTIMISTIC. Означает отсутствие каких бы то ни было блокировок. «Оптимистическая» блокировка предполагает, что даже во время выполнения вами редактирования данных, другие пользователи смогут к ним обращаться.

Синтаксис

DECLARE имя-курсора CURSOR

[ область-видимости-имени-курсора ]

[ возможность-перемещения-по-курсору ]

[ типы-курсоров ]

[ опции-распараллеливания-курсоров ]

[ выявление-ситуаций-с-преобразованием-типа-курсора ]

FOR инструкция\_select

[ опция-FOR-UPDATE ]

**Прокрутка курсора:**

Основой всех операций прокрутки курсора является ключевое слово FETCH. В качестве аргументов оператора FETCH могут выступать:

* NEXT – возвращает строку результата сразу же за текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH NEXT выполняет первую выборку в отношении курсора, она возвращает первую строку в результирующем наборе. NEXT является параметром по умолчанию выборки из курсора.
* PRIOR – возвращает строку результата, находящуюся непосредственно перед текущей строкой и перемещает указатель текущей строки на возвращенную строку. Если инструкция FETCH PRIOR выполняет первую выборку из курсора, не возвращается никакая строка и положение курсора остается перед первой строкой.
* FIRST – возвращает первую строку в курсоре и делает ее текущей
* LAST – возвращает последнюю строку в курсоре, и делает ее текущей.

## 14.Оптимизация запроса: индексы, партиционирование, сегментирование

**Индекс** – это объект базы данных, обеспечивающий дополнительные способы быстрого поиска и извлечения данных. Индекс может создаваться на одном или нескольких столбцах.

Это означает, что индексы бывают **простыми** и **составными**. Если в таблице нет индекса, то поиск нужных строк выполняется простым сканированием по всей таблице. При наличии индекса время поиска нужных строк можно существенно уменьшить.

**Недостатки**:

* дополнительное место на диске и в оперативной памяти,
* замедляются операции вставки, обновления и удаления записей.

Представление индекса в виде сбалансированного дерева означает, что стоимость поиска любой строки остается относительно постоянной, независимо от того, где находится эта строка.

**Сбалансированное дерево** состоит из:

* корневого узла (root node), содержащего одну страницу,
* нескольких промежуточных уровней (intermediate levels), содержащих дополнительные страницы,
* листового уровня (leaf level).

## 

## Существует два типа индексов:

## • Кластерные индексы

## • Некластерные индексы, которые включают: – индексы на основе кучи; – индексы на основе кластерных таблиц.

## Кластерные индексы В кластерном индексе таблица представляет собой часть индекса, или индекс представляет собой часть таблицы в зависимости от вашей точки зрения. Листовой узел кластерного индекса – это страница таблицы с данными. Поскольку сами данные таблицы являются частью индекса, 53 для таблицы может быть создан только один кластерный индекс. Кластерный индекс является уникальным индексом по определению.

## Некластерные индексы на основе кучи В листьях некластерного индекса на основе кучи хранятся указатели на строки данных. Указатель строится на основе идентификатора файла (ID), номера страницы и номера строки на странице. Весь указатель целиком называется идентификатором строки (RID).

## Некластерные индексы, основанные на кластерных таблицах В листьях некластерного индекса, основанного на кластерных таблицах, хранятся указатели на корневые узлы кластерных индексов. Поиск в таком индексе состоит из двух этапов: • поиск в некластерном индексе; • поиск в кластерном индексе.

## Партиционирование

Партиционирование – это метод разделения больших (исходя из количества записей, а не столбцов) таблиц на много маленьких.

партиционирование по дате. партиционирование по диапазону идентификаторов

партиционирование по чему-нибудь другому – например, по первой букве имени пользователя.

Достоинства: – легко понять; – никакой поддержки — есть строго определенный набор партиций и нам никогда не придется добавлять новые;

Недостатки: – количество строк в партициях будет стабильно расти; – в некоторых партициях будет существенно больше строк, чем в других (больше людей с никами, начинающимися на «t\*», чем на «y\*»); – поиск по id потребует сканирования всех партиций;

**Сегментирование.**

Для распределения данных по кластерам используется их сегментация(Segmentation), а точнее сегментация проекций(Projection) в которых они находятся. Задача разработчика — подобрать такой список полей и/или такую функцию(например, хэш-функцию), благодаря которым данные равномерно распределятся по нодам кластера. HP Vertica рекомендует исполь55 зовать встроенные функции HASH и MODULARHASH для этих целей.

## 15. План запроса. Этапы выполнения запроса

**5 стадий выполнения запроса:**

1. Прикладная программа устанавливает подключение к серверу PostgreSQL. Эта программа передаёт запрос на сервер и ждёт от него результатов.
2. На этапе разбора запроса сервер выполняет синтаксическую проверку запроса, переданного прикладной программой, и создаёт дерево запроса. Дерево разбора состоит из узлов двух типов:
   * Атомы - лексические элементы следующих типов: ключевые слова; имена атрибутов или отношений; константы; скобки; операторы; и др.
   * Синтаксические категории - имена семейств, представляющих часть запроса. Заключаются в угловые скобки.
3. Система правил принимает дерево запроса, созданное на стадии разбора, и ищет в системных каталогах правила для применения к этому дереву. Обнаружив подходящие правила, она выполняет преобразования, заданные в теле правил. Одно из применений системы правил заключается в реализации представлений. Когда выполняется запрос к представлению (т. е. виртуальной таблице), система правил преобразует запрос пользователя в запрос, обращающийся не к представлению, а к базовым таблицам из определения представления.
4. Планировщик/оптимизатор принимает дерево запроса (возможно, переписанное) и создаёт план запроса, который будет передан исполнителю. н выбирает план, сначала рассматривая все возможные варианты получения одного и того же результата. Например, если для обрабатываемого отношения создан индекс, прочитать отношение можно двумя способами. Во-первых, можно выполнить простое последовательное сканирование, а во-вторых, можно использовать индекс. Затем оценивается стоимость каждого варианта и выбирается самый дешёвый. Затем выбранный вариант разворачивается в полноценный план, который сможет использовать исполнитель.
5. Исполнитель рекурсивно проходит по дереву плана и получает строки тем способом, который указан в плане. Он сканирует отношения, обращаясь к системе хранения, выполняет сортировку и соединения, вычисляет условия фильтра и, наконец, возвращает полученные строки.

**План запроса**

**План выполнения** показывает, как будут сканироваться таблицы, затрагиваемые оператором — просто последовательно, по индексу и т. д. — а если запрос связывает несколько таблиц, какой алгоритм соединения будет выбран для объединения считанных из них строк.

Наибольший интерес представляет ожидаемая стоимость выполнения оператора, которая показывает, сколько, по мнению планировщика, будет выполняться этот оператор. Фактически выводятся два числа: стоимость запуска до выдачи первой строки и общая стоимость выдачи всех строк. Для большинства запросов важна общая стоимость, но в таких контекстах, как подзапрос в EXISTS, планировщик будет минимизировать стоимость запуска, а не общую стоимость (так как исполнение запроса всё равно завершится сразу после получения одной строки). Кроме того, если количество возвращаемых строк ограничивается предложением LIMIT, планировщик интерполирует стоимость между двумя этими числами, выбирая наиболее выгодный план.

Синтаксис вызова плана запроса:

EXPLAIN [( параметр [, …] )] … оператор

оператор - любой оператор план выполнения которого интересует.

Структура плана запроса представляет собой дерево узлов плана. Узлы на нижнем уровне дерева — это узлы сканирования, которые возвращают необработанные данные таблицы. Разным типам доступа к таблице соответствуют разные узлы: последовательное сканирование, сканирование индекса и сканирование битовой карты. Источниками строк могут быть не только таблицы, но и например, предложения VALUES и функции, возвращающие множества во FROM, и они представляются отдельными типами узлов сканирования. Если запрос требует объединения, агрегатных вычислений, сортировки или других операций с исходными строками, над узлами сканирования появляются узлы, обозначающие эти операции. И так как обычно операции могут выполняться разными способами, на этом уровне тоже могут быть узлы разных типов.

Стоимость может измеряться в произвольных единицах, определяемых параметрами планировщика. Традиционно единицей стоимости считается операция чтения страницы с диска; Стоимость узла верхнего уровня включает стоимость всех его потомков.

Эта стоимость отражает только те факторы, которые учитывает планировщик. В частности, она не зависит от времени, необходимого для передачи результирующих строк клиенту, хотя оно может составлять значительную часть общего времени выполнения запроса. Тем не менее планировщик игнорирует эту величину, так как он не сможет изменить её, выбрав другой план.