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

**База данных** – это самодокументированное собрание интегрированных записей
Самодокументированное – содержит описание собственных структур
Интегрированных – содержит файлы данных, метаданные, индексы
**Осн требования в БД** (1. Неизбыточность данны**. 2.** Совместное использование данных **3.** Эффективность доступа к БД **4.** Целостность данных **5.** Безопасность данных **6**. Восстановление данных после 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 данными во ОП 3. Управление транзакциями 4. Журнализация 5.Поддержка языков БД.
Классификация СУБД:
По модели данных: 1. Дореляционные. 2 Реляционные 3. Постреляционные
По архитектуре организации хранения данных: локальные и распределенные
По способу доступа к БД: 1. Файл-серверные 2. Клиент-серверные. 3. Встраиваемые 4. Сервисно-ориентированные

## 2. Семантическое моделирование 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

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

**Сущность** — это реальный или представляемый объект, информация о котором должна сохраняться и быть доступна.

**Связь** — это ассоциация, устанавливаемая между сущностями.

**Потенциальный ключ** – непустое подмножество множества атрибутов, кот обладает свойствами уникальности и неизбыточности.

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

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

**Структурная часть** — описывает, из каких объектов состоит реляционная модель. Основными понятиями структурной части реляционной модели являются

- тип данных
- домен – уточнение типа данных
- атрибут отношения — это пара вида <имя_атрибута, имя_домена >.
- схема отношения — мн-во упорядоченных пар <имя_атрибута, имя_домена>
- кортеж — это мн-во упорядоченных пар <имя_атрибута, значение_атрибута>, которое содержит одно вхождение каждого имени атрибута, принадлежащего схеме отношения.
- 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.

Свойство отношения

- в отношении нет одинаковых кортежей
- кортежи и атрибуты не упорядочены
- каждый картеж содержит ровно й значение для каждого атрибута
- все значения атрибутов атомарны

**Потенциальный ключ** – непустое подмножество множества атрибутов, кот обладает свойствами уникальности и неизбыточности.

**Целостная часть** —два базовых требования целостности: целостность сущностей и ссылочная целостность (или целостность внешних ключей).

**Целостность сущности** – каждый кортеж отношения должен отличаться от любого другого кортежа этого отношения

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

**Ссылочная целостность** – нельзя добавить запись содержащую внешним ключ, кот не существует

Внешний ключ в отношении P2 – непустое подмножество множества атрибутов FK этого отношения, такое что

- сущ отношение P1 с потенциальным ключом CK
- каждое значение внешнего ключа FK в текущем значении P2 обязательно совпадает со значением CK некоторого кортежа в тек знач отношения P1

**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.

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

P – отношение, X и Y произвольные подмножества множества атрибутов. X->Y каждое значение X связано в точности в одним значением Y

Аномалии обновления:

INSERT – при вставке 1 стр относящейся к конкретной сущности надо добавить информацию о др сущности

DELETE – не можем удалить информацию от одной сущности не удалив информацию о др сущности

UPFATE – при изменении названия города надо переписывать все строчки (излишнее хранение данных) **Аксиомы Армстронга:**

1. Правило рефлексивности: (B ⊆A) ⇒(A -> B).
2. Правило дополнения: (A -> B) ⇒AC -> BC
3. Правило транзитивности: (A -> B) и (B -> C) ⇒(A -> C).
4. Правило самоопределения:A -> A.
5. Правило декомпозиции: (A -> BC) ⇒(A -> B) и (A -> C)
6. Правило объединения: (A -> B) и (A -> C) ⇒(A -> BC).
7. Правило композиции: (A -> B) и (C -> D) ⇒(AC -> BD).
8. Правило Дарона: (A->B)и(C->D)⇒(A(C–B)->BD).

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

**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

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

Неприводимое множество

- 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

Процесс нормализации преследует цели:

- избежать избыточности хранения данных;
- устранить аномалии обновления отношений.

1НФ – все атрибуты простые

2НФ – 1НФ + каждый неключевой атрибут неприводимо зависит от первичного ключа

3НФ – 2НФ + каждый неключевой атрибут нетранзитивно зависит от первичного ключа

Нормальная форма Бойса-Кодда

1. Переменная-отношение имеет два (или более) потенциальных ключа.
2. Эти потенциальные ключи являются составными.
3. Два или более потенциальных ключей перекрываются (т. е. имеют по крайней мере один общий атрибут).

3НФ + каждый детерминант является потенциальным ключом.

4НФ + все нетривиальные многозначные зависимости явл ФЗ от ПК

## 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.

АСИД:

- **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 завершения, происходит фиксация транзакции, и становится невозможным откат до состояния до начала транзакции.

Управлением параллельным выполнением транзакций:

- **Пессимистическое** Блокировки и семафоры в системы. PostgreSql
- **Оптимистическое** технология версии строк. Конфликты.

Неблагоприятные эффекты, вызванные параллельным выполнением транзакций:

1. **проблема последнего изменения** 2 транзакции выполняются одновременно, и они изменяют один и тот же набор данных. Решение: последовательное внесение изменений;
2. **проблема "грязного" чтения**. Чтение данных во время того, как другой пользователь работает с ними и данные находятся в несогласованном состоянии. Решение: считывание данных после окончания всех изменений – ожидание всех писателей и затем запуск читателей;
3. **проблема неповторяемого чтения** Пытаемся считать один и тот же набор данных несколько раз и получаем разный результат, так как др пользователь изменил их. Решение: ожидание всех читателей и потом запуск всех писателей.
4. **проблема чтения фантомов** (Phantom) Чтение данных во время того как другая транзакция их удаляет, получается работа с умершим объектом.

Уровни изоляции транзакции:

**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.

Чем выше уровень изоляции транзакций тем меньше уровень параллелизма

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

**Блокировка** — это механизм, с помощью которого синхронизирутся одновременный доступ нескольких пользователей к одному фрагменту данных.

**Свойства**

- Гранулярность (размер блокировки)
- Режим (тип блокировки) оперируем действием, которое привело к блокировке
- Продолжительность, время, которое ресурс был заблокирован

**Гранулярность** – размер объекта, который блокируется в случае какой либо работы. (Блокировка нескольких строк, При обновлении по некоторому ключу, могут быть заблокированы строки внутри индекса, Блокировка страницы, Блок из 8 страниц, Таблица, Файл базы данных, метаданные)

**Эскалация** (lock escalation) - укрупнение блокировки с целью минимизации времени блокировки.

**Режимы блокировок**

- Совмещаемая блокировка (SELECT) чтобы вместе с читателями не было писателей, 2 уровень изоляции транзакций, запрещение неповторяемого чтения.
- Блокировка обновления (UPDATE)
- Монопольная блокировка (X) инструкции INSERT, UPDATE или DELETE
- Блокировка с намерением, update в большом обьеме данных, где много времени уходит на поиск
- Блокировка массового обновления (BU) Используется, если выполняется массовое копирование данных в таблицу и указана подсказка TABLOCK.
- Диапазон ключей Защищает диапазон строк, считываемый запросом при использовании уровня изоляции сериализуемой транзакции. Запрещает другим транзакциям вставлять строки, что помогает запросам сериализуемой транзакции уточнять, были ли запросы запущены повторно

**Продолжительность блокировки** определяется тремя факторами:

• типом запроса;
• уровнем изоляции транзакции;
• наличием или отсутствием подсказок блокировки

Поиск **взаимных блокировок** осуществляется путём построения и анализа графа тупика (замкнутая цепь запросов)

После обнаружения взаимоблокировки надо выбрать жертву блокировки, процесс, который будет убит. В качестве жертвы взаимоблокировки выбирается сеанс, выполняющий ту транзакцию, откат которой потребует меньше всего затрат.

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

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

**Модель восстановления** управляет процессом регистрации транзакций, определяет, требуется ли для журнала транзакций резервное копирование, а также определяет, какие типы операций восстановления доступны.

3 модели восстановления:

- Модель полного восстановления (FULL). По умолчанию. Регистрация всех изменения, которые произошли в системе. Создается полный журнал и его реплики.
- Модель восстановления с неполным протоколированием (BULK_LOGGED). Неполное протоколирование большинства массовых операций (копирование, т е операции, которые не требуют полной записи, а достаточно записать их факт).
- Простая модель восстановления (SIMPLE). Только запись определенных событий. Нужно часто производить резервное копирование, чтобы не потерять данные. Подходит для маленьких БД.

2 основных тип записи Log-файла:

- код выполненной логической операции,
- исходный и результирующий образ измененных данных.

**Физическая архитектура журнала транзакций**

Журнал транзакций состоит из одного или нескольких физических файлов. Каждый физический файл журнала разбивается на несколько виртуальных файлов журнала (VLF). Размером и количеством виртуальных журналов пользователь не управляет.

Усечение очистка данных, которые уже не нужны. Усекается все до начала виртуального журнала, в котором находится MinLSN.

MinLSN – номер минимальной инструкции, который необходимо хранить для того чтобы БД полностью была восстановлена после какой то проблемы. Если случился сбой, то накатив кусочек от начала MinLSN до последней контрольной точки можно получить ровно ту БД, которая была до момента сбоя.

Белым подсвечен активный журнал

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

Основными функциями безопасности SQL Server являются:

- проверка подлинности (аутентификация) – процедура проверки соответствия некоего лица и его учетной записи в компьютерной системе.
- авторизация – это предоставление лицу прав на какие-то действия в системе.

Дополнительными функциями безопасности SQL Server являются:

• шифрование,
• контекстное переключение,
• олицетворение,
• встроенные средства управления ключами.

В основе системы безопасности SQL Server лежат три понятия:

- участники системы безопасности (Principals) - это сущности, которые могут запрашивать ресурсы SQL Server;
- защищаемые объекты (Securables) – это ресурсы, доступ к которым регулируется системой авторизации. Некоторые защищаемые объекты могут храниться внутри других, создавая иерархии «областей», которые сами могут защищаться. К областям защищаемых объектов относятся (a) сервер, (b) база данных и (c) схема.
- система разрешений (Permissions). При непосредственном управлении разрешениями они назначаются субъекту явно, а при опосредованном разрешения назначаются через членство в группах, ролях или наследуются от объектов, лежащих выше по цепочке иерархии.

Аудит – функциональность системы безопасности, которая может отслеживать практически любое действие с сервером/БД и записывать эти действия в файловую систему

Система управления на основе политик позволяет администратору баз данных (АБД) создавать политики для управления объектами и экземплярами базы данных. Эти политики дают АБД возможность устанавливать правила создания и изменения объектов и их свойств.

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.
- DAC — прямое управление доступом. Доступ субъекта к объекту определяется наличием субъекта в списке доступа (ACL) объекта.
- RBAC — ролевая модель управления доступом. Доступ субъекта к объекту определяется наличием у субъекта роли, содержащей полномочия, соответствующие запрашиваемому доступу.
- ABAC — атрибутивная модель управления доступом. Доступ субъекта к объекту определяется динамически на основании анализа политик, учитывающих значения атрибутов субъекта, объекта и окружения

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

Логику работы с данными можно разделить на три основных языка:

1**. DDL (Data Definition Language).** Служит для описания структуры БД:

- Создание объекта (**create**)
- Изменение объекта (**alter**)
- Удаление объекта (**drop**)

2. **DML (Data Manipulation Language).** Служит для выполнения операция с данными:

- Выбор информации (**select**)

| | пишется | обрабатывается |
|---|---|---|
| select | 1 | 5 |
| From | 2 | 1 |
| Where | 3 | 2 |
| Group by | 4 | 3 |
| Having | 5 | 4 |
| Order by | 6 | 6 |

- Добавление данных (**insert**)
- Обновление данных (**update**)
- Удаление данных (**delete**)
  Delete from tab_name -> проставляет крестики
  Vacuum full -> зачищает крестики
  Truncate table tab_name -> удаляет по настоящему таблицу

3. **DCL (Data Control Language).** Служит для осуществления администраторских действий

• Выдача прав доступа к объекту (grant)
• Удаление прав доступа на объект (revoke)

По признаку определения границ транзакций различают:

- **Автоматические транзакции**. Каждая инструкция выполняется как отдельная транзакция.
- **Неявные транзакции**. 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

Физическая реализация **JOIN**:

- Nested loop join
- Hash join

- Merge Join

Виды JOIN-ов (по типу соединений) :

- INNER JOIN
- LEFT/ RIGHT JOIN
- FULL JOIN

Common Table Expression (CTE) или **обобщенное табличное выражение** (ОТВ) – это временные результирующие наборы (т.е. результаты выполнения SQL запроса), которые не сохраняются в базе данных в виде объектов, но к ним можно обращаться. Основной целью ОТВ является написание рекурсивных запросов, можно сказать для этого они, и были созданы;

```
WITH RECURSIVE t(n) AS (
   VALUES (1)
  UNION ALL
   SELECT n+1 FROM t WHERE n < 100
)
SELECT sum(n) FROM t;
```

**Оконные функции**:

```
function_name() OVER (
   [PARTITION BY partition_expression]
   [ORDER BY sort_expression [ASC | DESC] [NULLS {FIRST | LAST}]]
)
```

- Ранжирующие
    - Row_number() последовательная нумерация
    - Rank() нумерация уникальных значений с пробелами в ранжировании
    - Dense_rank() то же что и rank() но без пробелов
- Агрегатные (sum, min, max, avg, count)
- Функции произвольного доступа
    - LEAD(): Получает доступ к данным из последующих строк.
    - LAG(): Получает доступ к данным из предыдущих строк.

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

По поведению:- 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.

**По типу возвращаемого значения**:

- Скалярная функция;
- Подставляемая табличная функция;
- Многооператорная табличная функция

Для БД все что не кортеж – скаляр (массив, json … - скаляр)

**Хранимая процедура** – именованный объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере.

**Триггер** - это хранимая процедура особого типа, которая выполняет одну или несколько инструкций в ответ на событие.

По типу события триггеры деляться на два класса:

- DDL-триггеры (create,  alter, drop)
- DML-триггеры (INSERT, UPDATE и DELETE)

В SQL Server существуют два класса триггеров:

- **INSTEAD OF**. Триггеры этого класса выполняются в обход действий, вызывавших их срабатывание, заменяя эти действия.
- **AFTER/BEFORE**. Триггеры этого класса исполняются после действия, вызвавшего срабатывание триггера.

**Курсор** – не объект БД. Объем данных и итератор, итерируемая проекция с которой можем работать построчно. При построчной обработке мы лишаемся всех благ параллельной обработки в БД.

Курсоры можно классифицировать:

- По области видимости (локальная и глобальная);
- По типу;
    - Статические курсоры (STATIC).
    - Динамические курсоры (DYNAMIC).
    - Курсоры, управляемые набором ключей (KEYSET).
    - Быстрые последовательные курсоры (FAST_FORWARD).  H
- По способу перемещения по курсору;
    - Последовательные курсоры (FORWARD_ONLY). Только вперед
    - Курсоры прокрутки (SCROLL). Вперед и назад
- По способу распараллеливания курсора
    - 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.

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

Индекс – это объект базы данных, обеспечивающий дополнительные способы быстрого поиска и извлечения данных.

Плюсы:

- При наличии индекса время поиска нужных строк можно существенно уменьшить.

Минусы:

- Дополнительное место на диске и в оперативной памяти,
- Замедляются операции вставки, обновления и удаления записей.

Существует два типа индексов:

- **Кластерные индексы**; Листовой узел – это страница таблицы с данными. Для таблицы может быть создан только один кластерный индекс.
- **Некластерные индексы**, которые включают:
  - на основе кучи; В листьях ссылки на файлы с данными
  - на основе кластерных индексов. В листьях – другие кластерные индексы. Родственные связи между обьектами

Могут создаваться неявно. Например при создании PRIMARY KEY. Hash join создает индексы.

**Партиционирование** – это метод разделения больших (исходя из количества записей, а не столбцов) таблиц на много маленьких. И желательно, чтобы это происходило прозрачным для приложения способом.

Разделение на патриции возможно по:

- по дате
- по диапазону идентификаторов
- по чему-нибудь другому – например, по первой букве имени пользователя

Патриции надо создавать самостоятельно.

Каждая партиция – отдельная таблица. Хорошо раб когда партиции приблизительно одного размера.

**Сегментирование базы данных** — подход, предполагающий разделение баз данных, отдельных её объектов или индексов поисковых систем на независимые сегменты, каждый из которых управляется отдельным экземпляром сервера базы данных, размещаемым, как правило, на отдельном вычислительном узле.

Распределение в основном идет по ключам, т е по атрибутам по котором будем производить join.

2 вида масштабирования

- Вертикальное – улучшение сервера, есть предел
- Горизонтальное – добавление нового сервера

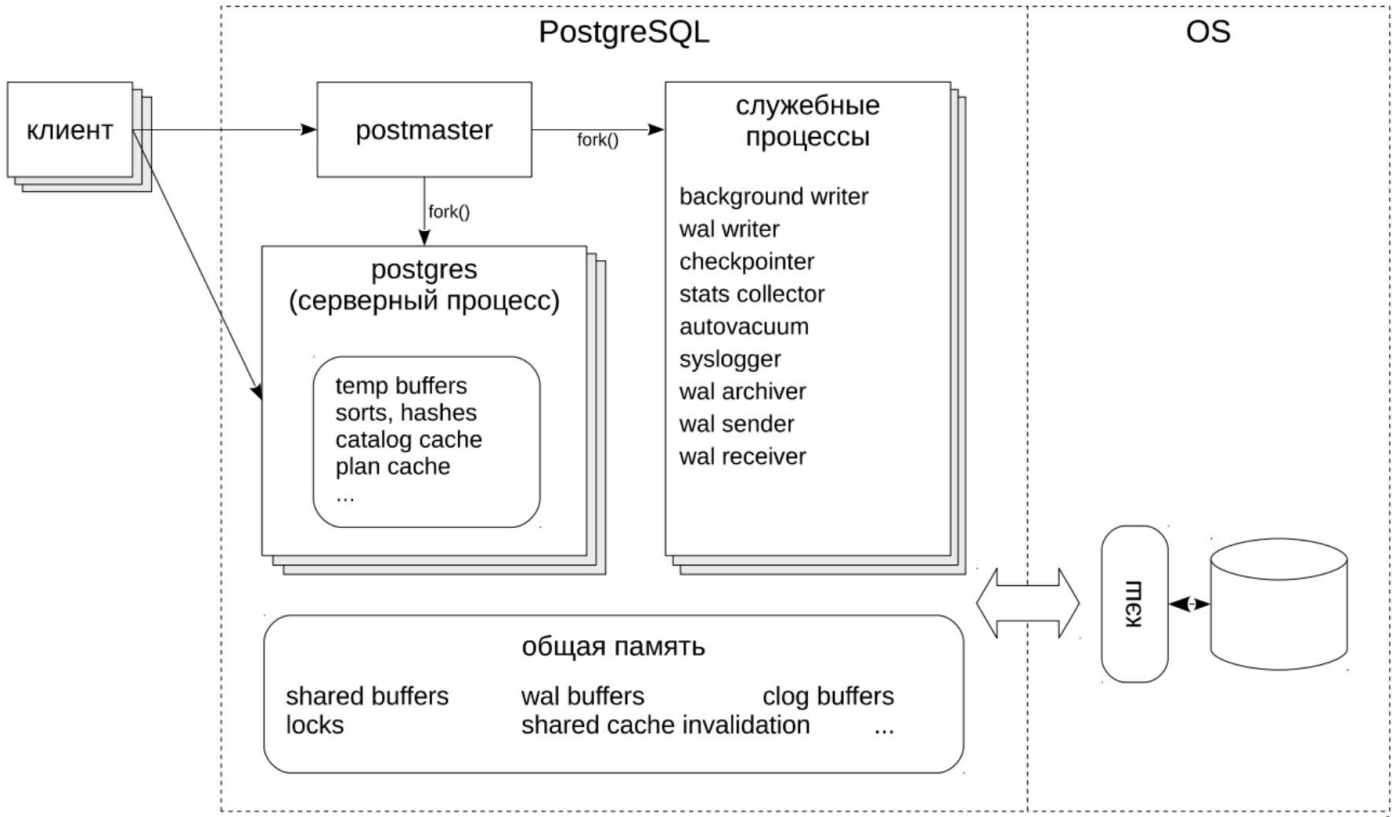
Сегментация по ключам, партиционированние по данным.

**Когда запрос работает долго**

1. Сегментация
2. Партиционированние
3. Индексирование

## 15. План запроса. Этапы выполнения запроса

Из чего состоит БД:




На каждое соединение создается по серверному процессу.

У экземпляра СУБД имеется общая для всех серверных процессов память. Большую ее часть занимает буферный кэш (shared buffers), необходимый для ускорения работы с данными на диске. В общей памяти находится информация о блокировках.

Служебный процесс обслуживает серверный: фоновые записи в лог, отслеживание checkoint, автоматическое vacuum.

**Стадии 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.
3. Система правил принимает дерево запроса, созданное на стадии разбора, и ищет в **системных каталогах правила** для применения к этому дереву.
4. **Планировщик**/оптимизатор принимает дерево запроса (возможно, переписанное) и создаёт план запроса, который будет передан исполнителю. Он выбирает план, сначала рассматривая все возможные варианты получения одного и того же результата.
5. Исполнитель рекурсивно проходит по дереву плана и **получает строки** тем способом, который указан в плане.

Наибольший интерес в выводимой информации представляет **ожидаемая стоимость выполнения оператора,** которая показывает, сколько, по мнению планировщика, будет выполняться этот оператор (это значение измеряется в единицах стоимости, которые не имеют точного определения, но обычно это об**ращение к странице на диске**).

EXPLAIN — показать план выполнения оператора

Структура плана запроса представляет собой дерево узлов плана

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

Массивно-параллельные системы обработки (massively parallel processing, MPP) – пример горизонтального масштабирования. — структура, в которой вместо одного перегруженного сервера используется несколько слегка нагруженных.

**Vertica** представляет собой колоночную СУБД с поддержкой MPP.

**Как хранятся данные в Vertica:**

Логические единицы хранения информации — это схемы, таблицы и представления. Физические единицы — это проекции. Проекции бывают нескольких типов:

- Суперпроекции - содержит все колонки нашей таблицы,
- запрос-ориентированные проекции - Если нужно ускорить какой-то из регулярных процессов, мы можем создать специальную запрос-ориентированную проекцию, которая будет содержать, 3 столбца из 10.
- агрегированные проекции (Aggregate Projections).

**Логическое хранение данных:** В *Vertica* есть две области хранения данных:

- **WOS** (Write Optimized Store) (Область, оптимизированная для записи).
- **ROS** (Read Optimized Store) (Область, оптимизированная для чтения).

**Tuple Mover** - механизм, который обеспечивает перетекание данных из первой области во вторую.

При использовании операции COPY, INSERT, UPDATE мы автоматически попадаем в WOS — область, где данные не оптимизированы для чтения и сортируются только при запросе, хранятся без сжатия или индексирования.

По истечении заданного в настройках времени данные из WOS перетекают в ROS — оптимизированную, ориентированную на чтение структуру дискового хранилища. В ROS хранится основная часть данных, здесь она сортируется и сжимается.

**Отказоустойчивость**

*Key Safety* — механизм отказоустойчивости, с помощью которого можно настраивать количество репликаций для каждой проекции. Копии проекций хранятся на соседних нодах и в случае отказа ноды берут работу на себя.

**Архитектура GreenPlum (параллельный PostgreSQL)**

Архитектура хранения данных аналогична *Vertica.* Но используется архитектура *master-slave.* Существует сервер-мастер, который по сути является маршрутизатором и только лишь координирует взимодействием между клиентом и серверами-сегментами. Данные хранятся только на серверах-сегментах.

Суть **модели MapReduce** состоит в разделении информационного массива на части, параллельной обработке каждой части на отдельном узле и финального объединения всех результатов.

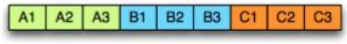
**Parquet -** это бинарный, колоночно-ориентированный формат хранения данных. Идеален для big data (неизменяемых).

Ускоряет работу когда в запросе не нужны все колонки сразу.

| A | B | C |
|---|---|---|
| A1 | B1 | C1 |
| A2 | B2 | C2 |
| A3 | B3 | C3 |

Тогда в текстовом файле, скажем, csv мы бы хранили данные на диске примерно так:


В случае с Parquet:



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

OLTP системы - min обьем БД - min время INSERT, UPDATE, DELETE
OLAP системы – денормализован - min время SELECT запросов
**Храни́лище да́нных** —база данных, предназначенная для подготовки отчётов и бизнес-анализа с целью поддержки принятия решений в организации.
В основу **пространственной модели** закладывается четкое разделение данных на измерения (Dimension) и факты (Fact). **Факт** —точечное атомарное событие (нет продолжительности) **Измерение** (dimension) — имеет версионность
**OLAP-куб** – используется для создания отчета по данным
В ячейке куба – файл, по осям – измерение.
Пользователи выполняют запросы чаще всего в разрезах. Имея предподготовленную сущность проще и быстрее получать такие разрезы.
**Архитектура хранилища**
Источники – OLTP и файлы
Staging – слой через который загружаются данные внутрь хранилища.
ODS – хранит историчность, факт изменения
DDS – правильное разложение данных. Перегруппировываем сущности, собираем общую информацию о них.

2 подхода построения DDS слоя
Инмону – хранилище должно находится в 3НФ, снижает аномалии
Кимболл – хранилище должно быть денормализовано (снежинки звездочки)
Гибридный метод Сначала используется Инмон, затем Кимвал

ETL и ELT — это два способа интеграции данных в одно хранилище. Основное различие между ними заключается в том, ГДЕ и КОГДА выполняются преобразование и загрузка данных.
ETL — *extraction*, *transformation* и *loading*.
ELT — *extraction*, *loading* и *transformation*

Как работает ETL:

1. Загрузка данных из источников
2. Очистка данных от ошибок (базовые проверки)
3. Мэппинг данных (преобразование данных в формат как в хранилище)
4. Агрегация данных (объединение, т к в OLTP-система может содержать несколько сумм для одного и того же набора элементов)
5. Выгрузка в целевую систему

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

Необходимость сократить время обработки операции. Что можно оптимизировать? – Чтение с диска.
In-Memory DataBase - базы данных в оперативной памяти.
В терминах ACID базы данных жертвуем Durability (надежность).

Существуют ряд методов, которые позволяют снизить этот риск:

- Снэпшоты на диск
- Логи транзакций
- Использование NVRAM или NVDIMM - модулей памяти, которые сохраняют свое состояние при пропадании питания

**Redis** – быстрое хранилище в памяти с для структур данных «ключ-значение». (хеш-таблица)
Несколько фактов о ключах:
Плохая идея - хранить слишком длинные ключи.
Хорошая идея — придерживаться схемы при построении ключей: «object-type:id:field»

Типы данных Redis

- Строки (strings). Базовый тип данных Redis. Строки в Redis бинарнобезопасны, могут использоваться так же как числа, ограничены размером 512 Мб.
- Списки (lists). Классические списки строк, упорядоченные в порядке вставки, которая возможна как со стороны головы, так и со стороны хвоста списка.
- Множества (sets). Множества строк в математическом понимании: не упорядочены, поддерживают операции вставки, проверки вхождения элемента, пересечения и разницы множеств.
- Хеш-таблицы (hashes). Классические хеш-таблицы или ассоциативные массивы.
- Упорядоченные множества (sorted sets). Упорядоченное множество отличается от обычного тем, что его элементы упорядочены по особому параметру «score».

+ Не тратится время на обращение к диску и считывание с него данных
+ Есть разные структуры данных
+ Отсутствие сложных операций, количество инструкций меньше, чем в стандартном SQL
+ Репликация и сохранность – есть логи и снэпшоты

Примеры использования In-Memory DataBase
• Кэширование. Высокопроизводительный кэш в памяти, который уменьшает задержку доступа, увеличивает пропускную способность и уменьшает нагрузку на реляционную БД
• Управление сессиями. За счет того можно регулировать время жизни переменных в Redis
• Таблицы лидеров в режиме реального времени. Когда обращение к бд занимает 20 сек а обновлять таблицу надо каждые 10 сек
• Ограничение интенсивности. За счет работы с очередями
• Очереди
• Чат и обмен сообщениями. Redis поддерживает стандарт «издатель-подписчик»

## 16. Гибкие методологии проектирования. Data Vault, Anchor Modeling

Модель DataWarehouse (DWH) – хранилища данных
Снежинка
DataVault 2.0
Якорная модель

Говорим про DDS слой, те данные, которые уже затянули с помощью ETL в хранилище данных.
Критерии сравнения:

- Количество человеко-дней, потраченных инженером на доработку той или иной сущности
- Количество человеко-дней, потраченных инженером, который должен добавлять новые объекты с хранилище, т е проработка нового источника
- Способность пользователей разбираться в структуре DDS слоя без сопроводительной документации

**Снежинка**
- Невозможность параллельной работы нескольких инженеров, так как таблица 1
- При внесении и добавлении данных в хранилище требуется множество согласований
+ Нет используется сложных логических структур – не нужна сопроводительная документация
- Много Join

**DataVault**
Данные в Data Vault представлены в виде трех сущностей:

- Хаб (hub) – бизнес-сущность, например клиент, продукт, заказ. Хаб-таблица содержит несколько полей: натуральные ключи сущности, суррогатный ключ, ссылка на источник записи и время добавления записи.
- Линк (связь, link) – представление отношений между Хабами. Линки строятся в виде таблиц «многие ко многим» и содержат ссылки на суррогатные ключи связанных Хабов.
- Сателлит (satellite) - изменяемые атрибуты сущностей, контекстные данные для хабтаблицы, связанные с Хабами и Связями по принципу «один ко многим».

+ При изменении сущности просто будем добавлять Сателлиты
+ При изменении сущностей сохраняем только ключ в хаб, а в снежинке мы бы дублировали все в таблице
- Нужна доп документация (мы ничего не знаем о сателите если он нигде не описан)
- Большое количество join (больше чем в снежинке)

**Якорная модель**
Якорь – сущность, событие, файт
+ Можно разрабатывать сущности независимо друг от друга
+ 6НФ – самый минимальный объем DDS слоя который можно достичь, избавление аномалий
- нужна документация
- еще больше join