

Здесь содержатся расписанные билеты по БД.

Текущим цветом выделены разделы билетов

Синим цветом выделены куски “от себя”, но очень рекомендую их читать

Источники:

- статьи **yandex.cloud**, **otus**, **habr**, **sql-ex**, **microsoft**, **oracle**, **ibm** и некоторые другие компетентные ресурсы
- презентации курса **МФТИ** по БД
- презентации курса **ВШЭ** по распределенным хранилищам
- презентации курса **postgres-pro** <https://edu.postgrespro.ru/sqlprimer/>
- программа **ЗФТШ** по БД
- **yandex-gpt**
- личный опыт автора и его друга

СОДЕРЖАНИЕ:

Вопрос 1: База данных и модель данных. Иерархическая и сетевая модели данных: свойства, сходства и различия, достоинства и недостатки. Реляционная модель данных. Основные понятия реляционной модели данных.	2
Вопрос 2: Реляционная алгебра. Операции реляционной алгебры. Реляционные базы данных. SQL.	5
Вопрос 3: Разделы языка SQL: DDL, DML, DQL, DCL и TCL	7
Вопрос 4: Транзакции. TCL. Описать синтаксис. Точка останова. ACID свойства транзакций. Уровни изоляции транзакций. Проблемы поддержки изолированности.	8
Вопрос 5: Тройчная логика. Работа с NULL.	10
Вопрос 6: Проектирование баз данных. Основные задачи. Основные шаги. Этапы проектирования.	12
Вопрос 7: Нормальные формы. Потенциальные, первичные, внешние ключи. Аномалии данных в базе. Физическая модель данных	14
Вопрос 8: SCD. Типы SCD. Работа с несколькими версионными таблицами. Поиск разрывов и пересечений версионности. Сравнение таблиц: поиск всех изменений между 2 версиями одной и той же таблицы.	17
Вопрос 9: Хранимые процедуры и функции. Триггеры.	20
Вопрос 10: План запроса. Жизненный цикл запроса.	22
Вопрос 11: Методы соединения, основные черты, преимущества, недостатки.	23
Вопрос 12: Индексы. Определение, условия использования, способы сканирования.	25
Вопрос 13: Масштабируемость. Вертикальная и горизонтальная масштабируемость. Законы Амдала и Густавсона-Барсиса.	27
Вопрос 14: CAP-теорема. Современные СУБД с точки зрения CAP-теоремы. Классификация систем по признакам CAP. BASE свойства.	29
Вопрос 15: Классификация NoSQL СУБД. Описание характерных черт по каждому классу. Особенности работы с NoSQL.	31
Вопрос 16: Хранилища данных. Предпосылки к созданию, общая концепция. ETL vs ELT. Принципы создания, модели данных. Отчетность, OLAP кубы.	33

Вопрос 1: База данных и модель данных. Иерархическая и сетевая модели данных: свойства, сходства и различия, достоинства и недостатки.
Реляционная модель данных. Основные понятия реляционной модели данных.

- **База данных**

БАЗА ДАННЫХ

- совокупность данных, хранимых в соответствии со схемой данных, манипулирование которыми выполняют в соответствии с правилами средств моделирования данных
- совокупность данных, организованных в соответствии с концептуальной структурой, описывающей характеристики этих данных и взаимоотношения между ними, причём такое собрание данных, которое поддерживает одну или более областей применения

- **Модель данных**

- *Модель данных — это абстракция, которая прикладывается к конкретным данным и позволяет трактовать их как информацию.*
- *Модель данных описывает определённый набор родовых понятий и признаков, которыми обладают все конкретные системы управления базами данных и управляемые ими базы данных, если они используют эту модель.*
- *Словами из доки ЗФТШ:*

Модель данных — инструмент (ни о каких физических данных речи не идет), формальная теория представления и обработки данных в системе управления базами данных, которая включает, по меньшей мере, три аспекта:

- аспект структуры (организация хранения): методы описания типов и логических структур данных в базе данных;
- аспект манипуляции: методы манипулирования данными;
- аспект целостности: методы описания и поддержки целостности базы данных.

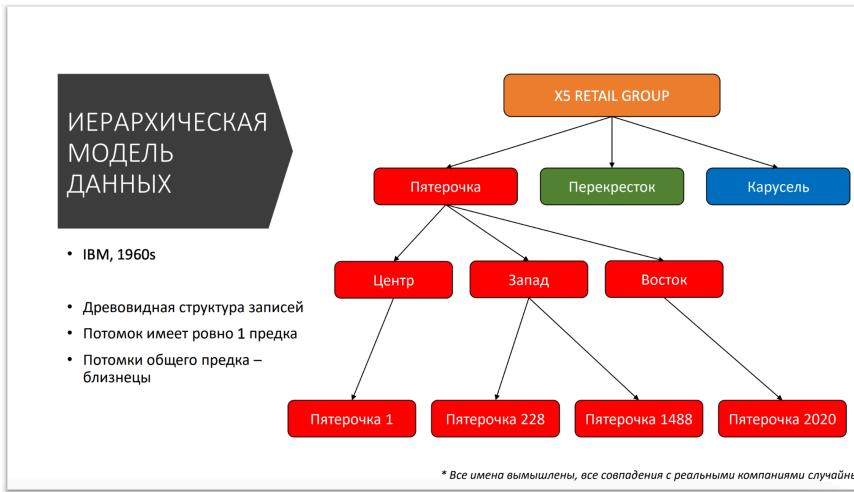
МОДЕЛЬ ДАННЫХ

Иерархическая модель данных

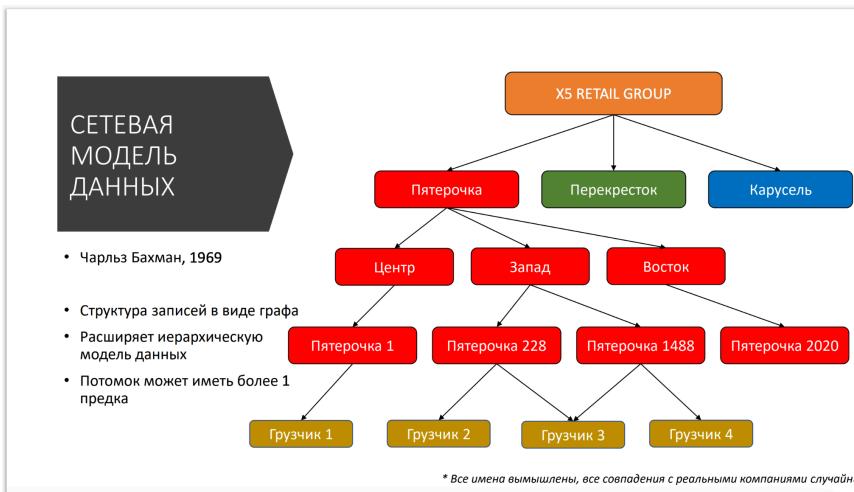
Сетевая модель данных

Реляционная модель данных

- **иерархическая модель данных**



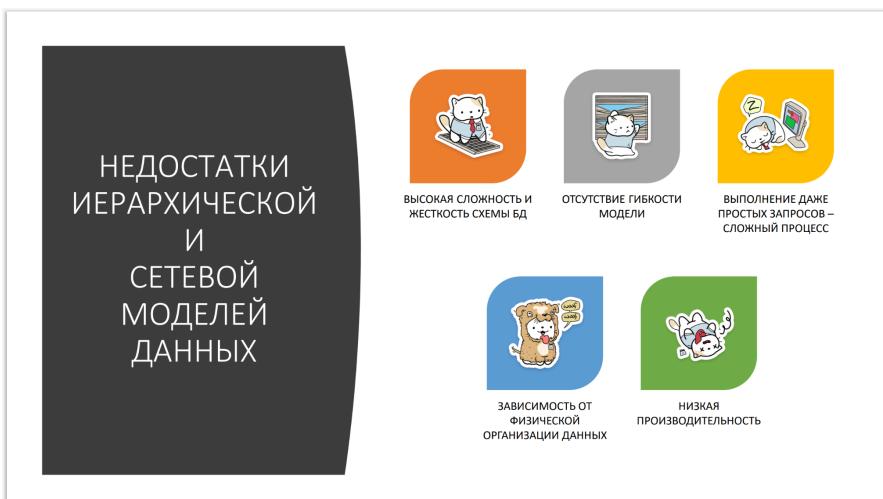
- сетевая модель данных



- сходства и различия моделей

- В целом сходства и различия моделей можно выявить, прочитав описание моделей
- Основное сходство - общая идеология с графовым подходом(дерево - частный случай графа)
- Основное различие в последнем свойстве про число предков

- достоинства и недостатки моделей



Иерархическая система:

Достоинства	Недостатки
Эффективное использование памяти ЭВМ	Незэффективность (по времени и сложности запросов)
Высокая скорость выполнения основных операций над данными	Медленный доступ к сегментам данных нижних уровней иерархии
Высокая скорость выполнения основных операций над данными	Четкая ориентация на определенные типы запросов
Простота при работе с небольшим объемом данных так как, иерархический принцип соподчиненности понятий является естественным для многих задач	Громоздкость для обработки информации с достаточно сложными логическими связями
	Также сложность понимания для обычного пользователя

Сетевая модель данных:

Достоинства	Недостатки
Возможность эффективной реализации по показателям затрат памяти и оперативности	Высокая сложность и жесткость схемы БД
В сравнении с иерархической моделью сетевая модель предоставляет большие возможности в смысле допустимости образования произвольных связей	Также сложность для понимания и выполнения обработки информации в БД обычным пользователем
В рамках сетевых СУБД легко реализуются иерархические <u>дата-логические</u> модели.	В сетевой модели данных ослаблен контроль целостности связей вследствие допустимости установления произвольных связей между записями

- реляционная модель данных

Реляционная модель:

Понятие реляционный (англ. relation – отношение) связано с разработками известного американского специалиста в области систем баз данных Е.Кодда.

Реляционная модель ориентирована на организацию данных в виде двумерных таблиц. Каждая реляционная таблица представляет собой двумерный массив и обладает следующими свойствами:

- каждый элемент таблицы – один элемент данных;
- все столбцы в таблице однородные, т.е. все элементы в столбце имеют одинаковый тип (числовой, символьный и т.д.) и длину;
- каждый столбец имеет уникальное имя (заголовки столбцов являются названиями полей в записях);
- одинаковые строки в таблице отсутствуют;
- порядок следования строк и столбцов может быть произвольным.

- основные понятия реляционной модели

ОСНОВНЫЕ ПОНЯТИЯ

- *Домен* – это множество допустимых значений
- *Атрибут* – это наименование домена
- *Кортеж* – это упорядоченный набор фиксированной длины
- *Отношение* – это математическая структура, которая формально определяет свойства различных объектов и их взаимосвязи
- *Арность отношения* – количество его элементов

Вопрос 2: Реляционная алгебра. Операции реляционной алгебры. Реляционные базы данных. SQL.

- **реляционная алгебра**

РЕЛЯЦИОННАЯ АЛГЕБРА

Семейство $\mathfrak{U} \subset 2^X$ подмножеств множества X (носитель алгебры) называется **алгеброй**, если оно удовлетворяет следующим свойствам:

- $\emptyset \in \mathfrak{U}$
- Если $A \in \mathfrak{U}$, то $X \setminus A \in \mathfrak{U}$
- Если $A, B \in \mathfrak{U}$, то $A \cup B \in \mathfrak{U}$.

Реляционная алгебра:

- Носитель – множество (всевозможных) отношений различных (конечных) порядков

- **операции реляционной алгебры**

ОПЕРАЦИИ РЕЛЯЦИОННОЙ АЛГЕБРЫ

Теоретико-множественные

Реляционные

ТЕОРЕТИКО-МНОЖЕСТВЕННЫЕ ОПЕРАЦИИ

Применимы к совместимым отношениям:

- Объединение
- Разность
- Пересечение

Здесь и далее декартово произведение == расширенное декартово произведение, если не оговорено обратное

РЕЛЯЦИОННЫЕ ОПЕРАЦИИ

- Ограничение
- Проекция
- Соединение
- Деление

Если будут силы - пролистай презу в этой части про сами операции

- **реляционные базы данных**

- В основе – реляционная модель данных
 - Таблица ≈ Отношение
 - Заголовок отношения ≈ Список наименований колонок таблицы
 - Кортеж ≈ Стока таблицы
 - Тело отношения ≈ Все строки таблицы
 - Средство манипуляции – реляционные системы управления базами данных (СУБД)
 - Способ манипуляции – специальный язык запросов
- **РСУБД == Р + СУБД == реляционная система управления базами данных**

ОСНОВНЫЕ СУЩЕСТВУЮЩИЕ РСУБД



- PostgreSQL
 - Open source and free
 - Наиболее соответствует стандарту SQL
- Oracle
 - Первыми создали коммерческое решение (1979 г.)
 - Много дополнительных плюшек
- MS SQL
 - Оригинальная идея получена от Sybase
 - Хорошая интеграция с Microsoft решениями
 - Разнообразие процедурных расширений
- MySQL
 - Тоже open source
 - Тоже производится Oracle

- **sql**

- SQL(structured query language)

STRUCTURED QUERY LANGUAGE (SQL)

- Предметно-ориентированный язык (Domain-specific language)
- Используется для работы с реляционными БД
- Управление большим количеством информации одним запросом
- Не нужно указывать, **как** получаем запись

- Для запросов важно понимать порядок выполнения SQL:

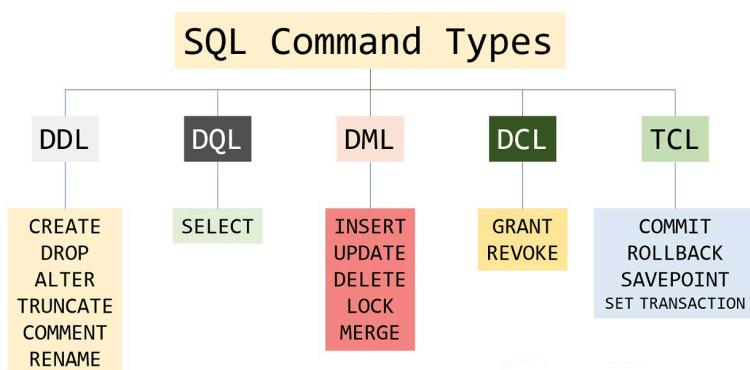
ПОРЯДОК ВЫПОЛНЕНИЯ ЗАПРОСА



- Еще полезно почитать: алиасы, глянуть примеры, посмотреть какие есть агрегирующие функции(ниже я добавил в 5 вопрос)
Еще чекни как работают подзапросы если будет время

Вопрос 3: Разделы языка SQL: DDL, DML, DQL, DCL и TCL

- **DDL** (data definition language)
 - Раздел SQL, отвечающий за описание данных
 - Это группа операторов определения данных. Другими словами, с помощью операторов, входящих в эту группу, мы определяем структуру базы данных и работаем с объектами этой базы, т.е. создаем, изменяем и удаляем их.
- **DML** (data manipulation language)
 - Раздел SQL, отвечающий за работу с данными, их изменение.
 - Это группа операторов для манипуляции данными. С помощью этих операторов мы можем добавлять, изменять, удалять и выгружать данные из базы, т.е. манипулировать ими.
 - **FYI:** сюда можно в принципе приписать и select если опустить упоминание DQL
- **DQL** (data query language)
 - Раздел SQL, отвечающий за запросы на выборку
 - Язык запросов данных предназначен для выполнения запроса к данным внутри схемы или объекта (т.е. таблицы, индекса, представления, функции и т.д.). С помощью запроса DQL мы можем получить данные из базы данных
- **DCL** (data control language)
 - Раздел SQL, отвечающий за контроль прав при работе с данными
 - Группа операторов определения доступа к данным. Иными словами, это операторы для управления разрешениями, с помощью них мы можем разрешать или запрещать выполнение определенных операций над объектами базы данных.
- **TCL** (transaction control language)
 - Раздел SQL, обеспечивающий управление транзакциями.
 - Группа операторов для управления транзакциями. Транзакция – это команда или блок команд (инструкций), которые успешно завершаются как единое целое, при этом в базе данных все внесенные изменения фиксируются на постоянной основе или отменяются, т.е. все изменения, внесенные любой командой, входящей в транзакцию, будут отменены.
- **Операторы:**



Вопрос 4: Транзакции. TCL. Описать синтаксис. Точка останова. ACID свойства транзакций. Уровни изоляции транзакций. Проблемы поддержки изолированности.

- **транзакции**

ТРАНЗАКЦИЯ

Транзакция – группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными, гарантированно переводящая БД из одного непротиворечивого состояния в другое.

- **TCL**

TCL (TRANSACTION CONTROL LANGUAGE)

- COMMIT
 - Применяет транзакцию, т.е. сохраняет изменения, произведенные в процессе выполнения транзакции
- ROLLBACK
 - Откатывает все изменения, произведенные в процессе выполнения транзакции
- SAVEPOINT
 - Создает так называемую точку останова

- **СИНТАКСИС**

TRANSACTION CONTROL LANGUAGE (TCL)

- BEGIN TRANSACTION transaction_mode [, ...]
 - ISOLATION LEVEL (SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED)
 - READ WRITE | READ ONLY
 - (NOT) DEFERRABLE
- BEGIN / START
 - COMMIT
 - ROLLBACK
- SAVEPOINT name
 - ROLLBACK TO SAVEPOINT name
 - RELEASE SAVEPOINT name

ПРИМЕР ИСПОЛЬЗОВАНИЯ

```
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
    INSERT INTO table1 VALUES (1);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (2);
    ROLLBACK TO SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (3);
    COMMIT;
```

- **точка останова**

- Точка останова – промежуточный участок в транзакции, на который можно откатиться в случае необходимости
 - Позволяет дробить транзакцию на части
 - Позволяет реализовать «вложенные» транзакции

- **ACID свойства**

(важно понимать, что чистого ACID на практике нет вообще нигде. Даже нераспределенный postgres на одном мастере - тоже не ACID)

СВОЙСТВА ТРАНЗАКЦИИ (ACID)

- **Atomicity** (Атомарность):
 - Выполнены либо все подоперации, либо никакие
- **Consistency** (Согласованность)
 - Каждая успешная транзакция фиксирует только допустимые результаты
- **Isolation** (Изолированность)
 - Параллельные транзакции не влияют на результаты друг друга
- **Durability** (Устойчивость)
 - Вне зависимости от сбоев системы результаты успешных транзакций сохраняются в системе



ATOMICITY (АТОМАРНОСТЬ)

- Никакая транзакция не будет зафиксирована в системе частично
- Выполнены либо все подоперации, либо никакие
- На практике одновременное и атомарное выполнение транзакций невозможно
- На практике «атомарность» реализуется с использованием «отката» (rollback)
- В процессе отката отменяются все уже произведенные операции

CONSISTENCY (СОГЛАСОВАННОСТЬ)

- Фиксируются только допустимые результаты операций
- Согласованность применяется не только в контексте БД.
Пример: банковские транзакции
- В ходе выполнения операции согласованность не требуется
- Вследствие атомарности промежуточная несогласованность остается скрытой

ISOLATION (ИЗОЛИРОВАННОСТЬ)

- Параллельные транзакции не оказывают влияния на друг друга
- В реальных БД полная изолированность не поддерживается
- Уровень изолированности – характеристика соответствия БД свойству изолированности

DURABILITY (УСТОЙЧИВОСТЬ)

- Вне зависимости от сбоев системы результаты успешных транзакций сохраняются в системе
- Если пользователь получил подтверждение об успешности транзакции, гарантируется сохранность результата

- уровни изолированности

УРОВНИ ИЗОЛИРОВАННОСТИ ТРАНЗАКЦИЙ

- **Read uncommitted** (чтение незафиксированных данных)
- **Read committed** (чтение фиксированных данных)
- **Repeatable read** (повторяемость чтения)
- **Serializable** (упорядочиваемость)

READ UNCOMMITTED

- Первый уровень изоляции
- Гарантирует отсутствие потерянных обновлений
- Итоговое значение – результат выполнения каждой транзакции
- Возможно считывание незафиксированных изменений
- Данные блокируются на время внесения изменений
- На время чтения данных блокировка отсутствует

READ COMMITTED

- Второй уровень изоляции
- Используется в большей части СУБД
- Защита от «грязного» чтения
- В процессе выполнения одна из транзакций успешно завершается, тогда остальные работают с измененными данными
- Реализация RPL на усмотрение разработчиков СУБД:
 - Блокирование читаемых и изменяемых данных
 - Сохранение нескольких версий параллельно изменяемых строк

REPEATABLE READ

- Третий уровень изоляции
- Читающая транзакция игнорирует изменения в данных, которые были ей ранее прочитаны
- Никакая транзакция не может изменить данные, читаемые текущей транзакцией, пока чтение не завершено
- Спасает от эффекта неповторяющегося чтения

SERIALIZABLE

- Четвертый (самый высокий) уровень изоляции
- Транзакции **полностью** изолированы друг от друга
- Параллельных транзакций как будто бы не существует вовсе
- Транзакции не подвержены эффекту «phantomного чтения»

Вопрос 5: Троичная логика. Работа с NULL.

- **троичная логика**

ТЕРНАРНАЯ (ТРОИЧНАЯ) ЛОГИКА

- Допустимые значения логического выражения:
 - TRUE
 - FALSE
 - UNKNOWN
- Как оперировать со значениями только из булевой логики
- Как оперировать со значениями типа UNKNOWN
- В SQL используется обозначение NULL



- **работа с NULL** - работа с неизвестным значением
- NULL в выражениях

СРАВНЕНИЕ С NULL

- $\text{NULL} = 1$ **NULL**
- $\text{NULL} <> 1$ **NULL**
- $\text{NULL} > 1$ **NULL**
- $\text{NULL} = \text{NULL}$ **NULL**



NULL В ВЫРАЖЕНИЯХ С AND/OR

- $(\text{NULL} = 1) \text{ OR } (1 = 0)$
 - NULL OR FALSE
- $(\text{NULL} = 1) \text{ OR } (1 = 1)$
 - NULL OR TRUE
- $(\text{NULL} = 1) \text{ AND } (1 = 0)$
 - NULL AND FALSE
- $(\text{NULL} = 1) \text{ AND } (1 = 1)$
 - NULL AND TRUE
- $\text{NOT } (\text{NULL} = \text{NULL})$
 - NOT NULL



NULL В ВЫРАЖЕНИЯХ С AND/OR

- $X \text{ AND TRUE}$ **X**
- $X \text{ AND FALSE}$ **FALSE**
- $X \text{ OR TRUE}$ **TRUE**
- $X \text{ OR FALSE}$ **X**



NULL В ВЫРАЖЕНИЯХ С AND/OR

- NULL OR FALSE **NULL**
- NULL OR TRUE **TRUE**
- NULL AND FALSE **FALSE**
- NULL AND TRUE **NULL**
- NOT NULL **NULL**



Т.е. если NULL как-то влияет на значение логического выражения, результат не определен

- NULL в запросах

- Из-за свойств NULL, работать напрямую с ним не получится, потому что выбирать данные по нему СУБД не будет
- для выбора есть предикат **IS NULL**
- в pg есть предикат **IS DISTINCT FROM**, который позволяет работать с NULL как с известным значением. То есть мы можем смотреть на то, разные ли значения в столбцах и там можно спокойно рассчитывать, что NULL и явное значение будут считаться различными по предикату
- в COALESCE возвращает первый не NULL. Если все NULL, то NULL
- в функциях агрегации NULL просто скипается

АГРЕГИРУЮЩИЕ ФУНКЦИИ

COUNT

- Определяет количество строк в результирующей таблице

MAX

- Определяет наибольшее из всех выбранных значений данного поля

MIN

- Определяет наименьшее из всех выбранных значений данного поля

SUM

- Определяет сумму всех выбранных значений данного поля

AVG

- Определяет среднее для всех выбранных значений данного поля

*Вопрос 6: Проектирование баз данных. Основные задачи. Основные шаги.
Этапы проектирования.*

- **проектирование баз данных**

.....

ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

– процесс создания детализированной модели данных* БД, а также необходимых ограничений целостности.

- Модель данных – абстрактная модель, которая:
 - Организует элементы данных
 - Описывает, как они взаимодействуют друг с другом
 - Описывает, как они взаимодействуют с объектами внешнего мира

- **основные задачи проектирования**

ОСНОВНЫЕ ЗАДАЧИ ПРОЕКТИРОВАНИЯ БД

	хранение всей необходимой информации		получение нужных данных по запросу
	сокращение избыточности и дублирования данных		обеспечение целостности базы данных

- **основные шаги проектирования**

ОСНОВНЫЕ ШАГИ ПРОЕКТИРОВАНИЯ

- Определение предметной области
- Выделение основных сущностей
- Определение взаимосвязей между сущностями
- Наложение логической структуры на данные
- Создание объектов в базе

- **этапы проектирования (подробные куски в 5 презентации)**

ОСНОВНЫЕ ЭТАПЫ ПРОЕКТИРОВАНИЯ

- Концептуальное (инфологическое) проектирование:
 - Итог: концептуальная модель в ER-нотации
- Логическое (даталогическое) проектирование:
 - Итог: логическая модель в ER-нотации
- Физическое проектирование
 - Итог: созданные в базе объекты с учетом их взаимосвязей

Ниже выкладка из нашего курса по распределенным хранилищам - просто для понимания со стороны



**Вопрос 7: Нормальные формы. Потенциальные, первичные, внешние ключи.
Аномалии данных в базе. Физическая модель данных**

- **нормальные формы** и их определения (НФ == нормальная форма)

НОРМАЛЬНАЯ ФОРМА

- **Нормальная форма** — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных
 - Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение
 - Приведение БД к нормальной форме – нормализация
 - Каждая следующая форма включает в себя ограничения предыдущих
 - **Первая нормальная форма (1NF)**
 - **Вторая нормальная форма (2NF)**
 - **Третья нормальная форма (3NF)**
 - Нормальная форма Бойса-Кодда (BCNF)
 - Четвертая нормальная форма (4NF)
 - Пятая нормальная форма / Нормальная форма проекции-соединения (5NF / PJNF)
 - Доменно-ключевая нормальная форма (DKNF)
 - Шестая нормальная форма (6NF)
-
- 1НФ: 1 ячейка – 1 значение
 - 2НФ: 1НФ + **все** неключевые атрибуты зависят от **всех** ключевых
 - Не существует неключевого атрибута, который зависел бы от какого-либо подмножества ключевых
 - 3НФ: 2НФ + **все** неключевые атрибуты зависят **только** от ключевых атрибутов:
 - Не существует неключевого атрибута, который бы зависел от какого-либо подмножества неключевых

- **ПОТЕНЦИАЛЬНЫЕ, ПЕРВИЧНЫЕ КЛЮЧИ**

- **Потенциальный ключ** – подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности:
 - Уникальность: нет и не может быть двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают
 - Минимальность: в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности
- **Первичный ключ** – это один из потенциальных ключей отношения, выбранный в качестве основного (*Primary key, PK*)

- **ВНЕШНИЕ КЛЮЧИ**

ВНЕШНИЙ КЛЮЧ

Пусть R_1 и R_2 – две переменные отношения, не обязательно различные. **Внешним ключом FK** в R_2 является подмножество атрибутов переменной R_2 такое, что выполняются следующие требования:

- В переменной отношения R_1 имеется потенциальный ключ PK такой, что PK и FK совпадают с точностью до переименования атрибутов
- В любой момент времени каждое значение FK в текущем значении R_2 идентично значению PK в некотором кортеже в текущем значении R_1 . Иными словами, в любой момент времени множество всех значений FK в R_2 является подмножеством значений PK в R_1 .

- аномалии в базе данных

АНОМАЛИИ ДАННЫХ

- Ситуация в таблице БД такая, что:
 - Существенно осложнена работа с БД
 - В БД присутствует противоречия
- Причина:
 - Излишнее дублирование данных в таблице

В презентации 3 вида аномалий данных:

- аномалии модификации - надо менять данные в нескольких местах сразу
- аномалии удаления - удаляя что то одно - удаляем сразу много лишнего и, возможно, нужного
- аномалии добавления - не можем добавить данные, не имея первичных ключей
- Еще возникающие проблемы изолированности вообще тоже считаются аномалиями в базе

ПРОБЛЕМЫ ПОДДЕРЖКИ ИЗОЛИРОВАННОСТИ

Потерянное обновление

- Изменение одного блока данных несколькими транзакциями

«Грязное» чтение

- Чтение данных, измененных впоследствии откатившейся транзакцией

Неповторяющееся чтение

- Повторное чтение измененных данных одной и той же транзакцией

Чтение «фантомов»

- Взаимосвязанные критерии изменения данных двумя транзакциями

- **физическая модель данных**
Физическая модель – описание реализации объектов логической модели на уровне конкретной базы данных с учетом всех ее особенностей
- рекомендую почитать эти слайды подробнее (4 презентация)

Вопрос 8: SCD. Типы SCD. Работа с несколькими версионными таблицами. Поиск разрывов и пересечений версионности. Сравнение таблиц: поиск всех изменений между 2 версиями одной и той же таблицы.

- **scd**

SLOWLY CHANGING DIMENSIONS (SCD)

- **Slowly changing dimensions (SCD)** – редко изменяющиеся измерения, то есть измерения, не ключевые атрибуты которых имеют тенденцию со временем изменяться
- Выделяют 5 основных типов (нумерация с 0)

- **типы scd**

SCD: ТИП 0

- После попадания в таблицу данные никогда не изменяются
- Практически никогда не используется (по понятным причинам)
- Не поддерживает версионность
- Является начальной «точкой отсчета» методологии SCD

SCD: ТИП 1

- Данные записываются поверх существующих значений
 - Старые значения нигде не сохраняются
 - Используется, если история не нужна
- Достоинства:
- Не добавляется избыточность
 - Очень простая структура
- Недостатки:
- Не хранит историю

SCD: ТИП 2

- Создание новой записи в таблице под каждую версию данных с добавлением полей даты начала и даты конца периода существования версии

EMPLOYEE_NM	POSITION_ID	DEPT_ID	VALID_FROM_DTTM	VALID_TO_DTTM
Николай	21	2	2010-08-11 00:00:00	2016-06-06 23:59:59
Николай	23	3	2015-06-07 00:00:00	5999-01-01 00:00:00
Денис	23	3	2010-08-11 00:00:00	2016-06-01 23:59:59
Борис	26	2	2010-08-11 00:00:00	5999-01-01 00:00:00
Пенки	25	2	2010-08-11 00:00:00	5999-01-01 00:00:00

SCD: ТИП 2

- В полях valid_from_dttm и valid_to_dttm обычно не используются значения NULL
- Вместо NULL используется некоторая константа, например, '5999-01-01 00:00:00' для valid_to_dttm, как в примере
- Такой подход упрощает написание условий:

```
WHERE day_dt BETWEEN valid_from_dttm AND valid_to_dttm вместо
WHERE day_dt >= valid_from_dttm
      AND (day_dt < valid_to_dttm
            OR valid_to_dttm IS NULL)
```

SCD: ТИП 2

- Достоинства:
- Хранит полную и неограниченную историю версий
 - Удобный и простой доступ к данным необходимого периода
- Недостатки:
- Провоцирует на избыточность или заведение дополнительных таблиц для хранения изменяемых атрибутов

SCD: ТИП 3

- В самой записи содержатся дополнительные поля для предыдущих значений атрибута.
- При получении новых данных, старые данные перезаписываются текущими значениями.

ID	UPDATE_DTTM	PREV_STATE	CURRENT_STATE
1	11.08.2010 12:58	0	1
2	11.08.2010 12:29	1	1

SCD: ТИП 3

- Достоинства:
 - Небольшой объем данных
 - Простой и быстрый доступ к истории
- Недостатки:
 - Ограниченнная история

SCD: ТИП 4

- История изменений содержится в отдельной таблице: основная таблица всегда перезаписывается текущими данными с перенесением старых данных в другую таблицу.
- Обычно этот тип используют для аудита изменений или создания архивных таблиц.

SCD: ТИП 4

Таблица с актуальными данными

EMPLOYEE_NM	POSITION_ID	DEPT_ID
Коля	21	2
Денис	23	3
Ворис	26	2
Пенни	25	2

Таблица с историей

EMPLOYEE_NM	POSITION_ID	DEPT_ID	HISTORY_DTM
Коля	21	1	11.08.2010 14:12:05
Денис	23	2	19.12.2012 09:54:57
Ворис	26	1	09.01.2018 22:22:22

SCD: ТИП 4

- Достоинства:
 - Быстрая работа с текущими версиями
- Недостатки:
 - Разделение единой сущности на разные таблицы

- работа с несколькими версионными таблицами

Работа с версионными таблицами включает в себя:

ТИПОВЫЕ ЗАДАЧИ SQL



- Поддержка версионности таблиц
- Соединение версионных таблицы
- Поиск разрывов в версионности
- Поиск пересечений в версионности
- Сравнение таблиц

ПОИСК РАЗРЫВОВ И ПЕРЕСЕЧЕНИЙ В ВЕРСИОННОСТИ



- Версионные таблицы хранят историю изменения атрибутов
- История изменения атрибутов представляет собой набор временных интервалов
- Зачастую требуется проверка истории на наличие разрывов или пересечений в версионности

СРАВНЕНИЕ ТАБЛИЦ

- Нередко возникает необходимость сравнения двух вариантов одной и той же таблицы и построения *таблицы с различиями* (*diff-таблицы*)

- ПОИСК РАЗРЫВОВ ВЕРСИОННОСТИ

ПОИСК РАЗРЫВОВ В ВЕРСИОННОСТИ

- Использование аналитических функций
- Разбить данные на группы по ключу таблицы:
 - по EMPLOYEE_NM
- Отсортировать данные в группах по дате начала действия интервала:
 - VALID_FROM_DTTM
- Проверить, превышает ли разница между датой начала действия текущего интервала и датой окончания действия предыдущего интервал в 1 секунду

- **поиск пересечений версионности**

ПОИСК ПЕРЕСЕЧЕНИЙ В ВЕРСИОННОСТИ

- Использование аналитических функций
- Разбить данные на группы по ключу таблицы:
 - EMPLOYEE_NM
- Отсортировать данные в группах по дате начала действия интервала:
 - VALID_FROM_DTTM
- Проверить, не начинается ли период действия текущей записи раньше, чем заканчивается предыдущий интервал

- **сравнение таблиц**

СРАВНЕНИЕ ТАБЛИЦ

- Таблица строк EMP, отсутствующих в EMP_OLD
 - результаты выполнения операций INSERT и UPDATE
- Таблица изменений неключевых полей
 - результаты работы операций UPDATE
- Таблица изменений в ключевых полях
 - результаты работы операций INSERT и DELETE
- Таблица всех изменений

- примеры, по которым составлены таблицы содержатся в лекции 6

Вопрос 9: Хранимые процедуры и функции. Триггеры.

- хранимые функции

ХРАНИМЫЕ ПРОЦЕДУРЫ И ФУНКЦИИ

– объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере

- Похожи на обычные процедуры языков высокого уровня:
 - входные параметры
 - выходные параметры
 - локальные переменные
 - числовые вычисления и операции над символьными данными
- Могут выполнять стандартные операции с базами данных (как DDL, так и DML)
- Возможны циклы и ветвления

ХРАНИМЫЕ ФУНКЦИИ

- позволяют повысить производительность
- расширяют возможности программирования
- поддерживают функции безопасности данных
- Вместо хранения часто запроса, достаточно ссылаться на соответствующую хранимую процедуру
- Рассматриваем на примере PostgreSQL

- хранимые процедуры

ХРАНИМЫЕ ПРОЦЕДУРЫ

В PostgreSQL до 11 версии были только хранимые функции, которые все называли хранимыми функциями. В 11 появились хранимые процедуры.

Функция	Процедура
Возвращает 1 или несколько значений	Не возвращает никаких значений
1 функция – 1 транзакция, в рамках которой её запустили	В процедуре можно создавать транзакции, используя TCL
Запускается с использованием SELECT	Запускается с использованием CALL

ПРЕИМУЩЕСТВА ХРАНИМЫХ ПРОЦЕДУР

- Скорость
- Сокрытие структуры данных
- Гибкое управление правами доступа
- Меньшая вероятность SQL injection
- Повторное использование SQL
- Простая отладка SQL

НЕДОСТАТКИ ХРАНИМЫХ ПРОЦЕДУР

- Размазывание бизнес-логики
- Скудность языка СУБД
- Непереносимость хранимых функций
- Отсутствие необходимых навыков у команды и высокая «стоимость» соответствующих специалистов

- триггеры

ТРИГГЕР

— хранимая процедура особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением (*INSERT*), удалением (*DELETE*) строки в заданной таблице, или изменением (*UPDATE*) данных в определённом столбце заданной таблицы реляционной базы данных.

- применяется для обеспечения целостности данных и реализации сложной бизнес-логики
- запускается сервером автоматически при попытке изменения данных в таблице, с которой он связан
- в случае обнаружения ошибки или нарушения целостности данных может произойти откат транзакции

- про все это очень подробно написано в презентации 8

Вопрос 10: План запроса. Жизненный цикл запроса.

- **план запроса**
 - a. **План запроса** (или план выполнения запроса) - это последовательность шагов, используемых для доступа к данным в системе управления реляционными базами данных SQL
 - b. **План выполнения запроса** — последовательность операций, необходимых для получения результата SQL-запроса в реляционной СУБД.
 - c. План в целом разделяется на две стадии:
 - Выборка результатов.
 - Сортировка и группировка, выполнение агрегаций.
 - d. Сортировка и группировка — это опциональная стадия, которая выполняется, если не найдено путей доступа для получения результата в запрошенном порядке.
 - e. Выборка результатов выполняется следующими способами:
 - вложенные циклы;
 - слияние.
- **жизненный цикл запроса**
https://sql-ex.ru/blogs/?/Anatomija_plan_a_zaprosa_v_PostgreSQL.html

Этапы ЖЦ

- a. Средство анализа просматривает инструкцию SELECT и разбивает её на логические единицы, такие как ключевые слова, выражения, операторы и идентификаторы.
- b. Строится дерево запроса с описанием логических шагов, необходимых для преобразования исходных данных в формат, требуемый результирующему набору.
- c. Оптимизатор запросов анализирует различные способы, с помощью которых можно обратиться к исходным таблицам. Затем он выбирает ряд шагов, которые возвращают результаты быстрее всего и используют меньше ресурсов.
- d. Дерево запроса обновляется для записи этого точного ряда шагов. Конечную, оптимизированную версию дерева запроса называют планом выполнения.
- e. Реляционный механизм начинает реализовывать план выполнения. В ходе обработки шагов, требующих данных из базовых таблиц, реляционный механизм запрашивает у подсистемы хранилища передачу данных из набора строк, указанных реляционным механизмом.
- f. Реляционный механизм преобразует данные, возвращённые подсистемой хранилища, в заданный для результирующего набора формат и возвращает результирующий набор клиенту.

Вопрос 11: Методы соединения, основные черты, преимущества, недостатки.

- методы соединения

СОЕДИНЕНИЕ ТАБЛИЦ

- **CROSS JOIN** (полное декартово произведение таблиц)
- **INNER JOIN** (исключает несовпадающие строки)
- **OUTER JOIN** (содержит несовпадающие строки):
 - **LEFT [OUTER] JOIN**
 - **RIGHT [OUTER] JOIN**
 - **FULL [OUTER] JOIN**

<https://tproger.ru/articles/sql-join>

- основные черты

CROSS JOIN

```
SELECT *  
FROM table_1  
CROSS JOIN table_2;  
  
SELECT column_name_list_1  
      , column_name_list_2  
     FROM table_1  
      , table_2;
```

На выходе: полное декартово произведение 2 таблиц

INNER JOIN

«Сцепление» строк 2 таблиц по заданному условию

```
SELECT column_name_comma_list  
      FROM table_1  
INNER JOIN table_2  
        ON table_1.column_name = table_2.column_name;
```

Только те строки, для которых условие соединения истинно

LEFT JOIN

«Сцепление» строк 2 таблиц по заданному условию

```
SELECT column_name_comma_list  
      FROM table_1  
LEFT JOIN table_2  
        ON table_1.column_name = table_2.column_name;
```

- В результате присутствуют **все** строки «левой» таблицы
- Те строки, которые не соединяются с «правой» таблицей, все равно попадают в результат
- Поля от «правой» таблицы в таких строках заполняются специальным значением NULL

RIGHT JOIN

«Сцепление» строк 2 таблиц по заданному условию

```
SELECT column_name_comma_list  
      FROM table_1  
RIGHT JOIN table_2  
        ON Table1.column_name = Table2.column_name;
```

- В результате присутствуют **все** строки «правой» таблицы
- Те строки, которые не соединяются с «левой» таблицей, все равно попадают в результат
- Поля от «левой» таблицы в таких строках заполняются специальным значением NULL

FULL JOIN

«Сцепление» строк 2 таблиц по заданному условию

```
SELECT column_name_comma_list  
      FROM table_1  
FULL JOIN table_2  
        ON table_1.column_name = table_2.column_name;
```

- В результирующей таблице присутствуют **все** строки «левой» и «правой» таблиц
- Иными словами, является комбинацией левого и правого соединения

SELF-JOIN

- Не является отдельным видом соединения
- Бывает полезен для некоторого типа задач

- Задача:
 - Дана таблица RELATIONS (PERSON_ID, PERSON_NM, FATHER_ID)
 - Необходимо сформировать таблицу с именами отцов и детей

- преимущества и недостатки

- отвечать на вопрос преимуществ и недостатков в сравнении между видами джоинов не имеет смысла, так как каждый вид соединения решает свою задачу.
- **Преимущества объединений:**

- Преимущество соединения в том, что оно выполняется быстрее.
- Время получения запроса с использованием соединений почти всегда будет быстрее, чем у подзапроса.
- Используя joins, вы можете минимизировать нагрузку на базу данных при вычислениях, т. Е. Вместо нескольких запросов использовать один запрос join. Это означает, что вы можете лучше использовать возможности базы данных для поиска, фильтрации, сортировки и т.д.

- **Недостатки объединений:**

- Недостатком использования соединений является то, что их не так легко прочитать, как подзапросы.
- Большее количество соединений в запросе означает, что серверу базы данных приходится выполнять больше работы, а значит, процесс извлечения данных занимает больше времени
- Поскольку существуют разные типы соединений, может возникнуть путаница в отношении того, какое соединение является подходящим типом соединения, которое следует использовать для получения правильного набора желаемых результатов.
- Соединений нельзя избежать при получении данных из нормализованной базы данных, но важно, чтобы соединения выполнялись правильно, поскольку неправильные соединения могут привести к серьезному снижению производительности и неточным результатам запроса.

- Есть интерес сравнивать джоины с подзапросами

- Есть прикольная статья
<https://www.geeksforgeeks.org/sql-join-vs-subquery/>
- Инфа есть в презентациях 3 и 5

Вопрос 12: Индексы. Определение, условия использования, способы сканирования.

- **Индексы**

Индексы - особые таблицы, используемые поисковыми системами для поиска данных. Их активное использование играет важнейшую роль в повышении производительность SQL-запросов.

Индекс - структура данных, позволяющая быстро определить положения требуемых данных в базе. Создается для столбца(-ов) в таблице.

Индексы хранятся в виде сбалансированных В-деревьев (B-tree).

Неплохая статейка:

<https://otus.ru/journal/vse-chto-neobhodimo-znat-pro-indeksy-ms-sql/>

- **Условия использования индексов**

Как используется индекс при поиске строк 

- Специальная подсистема СУБД (планировщик) проверяет, имеется ли для этой таблицы индекс, созданный на основе тех же столбцов, что указаны, например, в условии предложения WHERE.
- Если такой индекс существует, то планировщик оценивает целесообразность его использования в данном конкретном случае.
- Если его использование целесообразно, то сначала выполняется поиск необходимых значений в индексе, а затем, если такие значения в нем найдены, производится обращение к таблице с использованием указателей, которые хранятся в записях индекса.
- Таким образом, полный перебор строк в таблице может быть заменен поиском в упорядоченном индексе и переходом к строке таблицы по прямому указателю (ссылке).

ВАЖНО! Индексы требуют некоторых накладных расходов на их создание и поддержание в актуальном состоянии при выполнении обновлений данных в таблицах.

6

Важно, что даже если индекс создан, то не факт, что он точно будет использован при запросе (по крайней мере в postgres)

Еще полезно понимать, что индексам могут потребоваться перестроения при операции, меняющей данные, а это фактор, который говорит, что использовать индексы постоянно может быть плохо.

- **способы сканирования**

- индексное сканирование **INDEX SCAN**
 - Индексное сканирование — это метод доступа, при котором метод доступа возвращает идентификаторы версий строк по одному за раз.
 - Механизм индексирования получает очередной идентификатор, обращается к табличной странице, на которую он указывает, получает версию строки и, если она удовлетворяет правилам видимости, возвращает необходимый набор полей.
 - Процесс продолжается, пока у метода доступа не

закончатся подходящие под условия запроса идентификаторы.

- сканирование по битовой карте **BITMAP SCAN**
 - Сканирование по битовой карте — это способ доступа, который позволяет не тратить ресурсы на повторный просмотр табличных страниц.
 - Он происходит в два этапа:
 1. Сначала сканируется индекс (Bitmap Index Scan) и в локальной памяти процесса строится битовая карта. В этой карте отмечаются те строки, которые должны быть прочитаны.
 2. Когда индекс просканирован и битовая карта готова, начинается сканирование таблицы (Bitmap Heap Scan). При этом страницы читаются последовательно, и каждая страница просматривается ровно один раз.
- **неплохие статьи**
 - <https://habr.com/ru/companies/postgrespro/articles/578196/>
 - <https://russianblogs.com/article/28471058065/>
- **не относящееся сюда напрямую - сканирование запросов**
Это сканирование нужное, для понимания построения всего, что нужно для запроса и понимания всех механизмов запроса при его проведении. Это делается оператором **EXPLAIN**

Вопрос 13: Масштабируемость. Вертикальная и горизонтальная масштабируемость. Законы Амдала и Густавсона-Барсиса.

- **масштабируемость**

- **Масштабируемость базы данных** - это способность базы данных справляться с изменяющимися требованиями путем добавления / удаления ресурсов
- Есть варианты того, как мы можем масштабировать базы относительно видов - это вертикальная и горизонтальная масштабируемость.
- **Есть варианты относительно прикладных подходов:** шардирование(разбиение базы на куски по физическим серверам), репликация(создание слейвов), партиционирование(разбиение большой таблицы на логические куски)
https://web-creator.ru/articles/partitioning_replication_sharding

- **вертикальная масштабируемость**

Вертикальное масштабирование подразумевает увеличение производительности (процессора, памяти, диска) в рамках одного узла (хоста)

- **горизонтальная масштабируемость**

Горизонтальное масштабирование – рост производительности за счет добавлении ещё одного узла (хоста)

- **закон амдала**

- Закон Амдала — это формула, демонстрирующая потенциал ускорения вычислительной задачи, которого можно достичь при увеличении количества ресурсов системы. Обычно он используется в параллельных вычислениях, и может предсказать наличие реальных преимуществ от увеличения количества процессоров с учётом ограничений параллелизуемости программы.

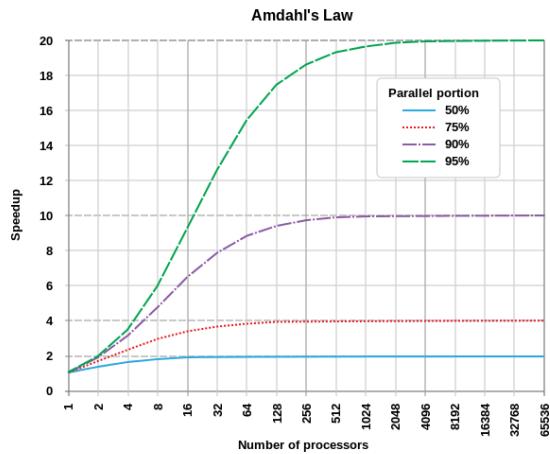
Закон Амдала (Amdahl's law)

-
- α – доля вычислений, которая должна выполняться последовательно,
 - N – число параллельных потоков выполнения,
 - S – полученное ускорение (speedup).

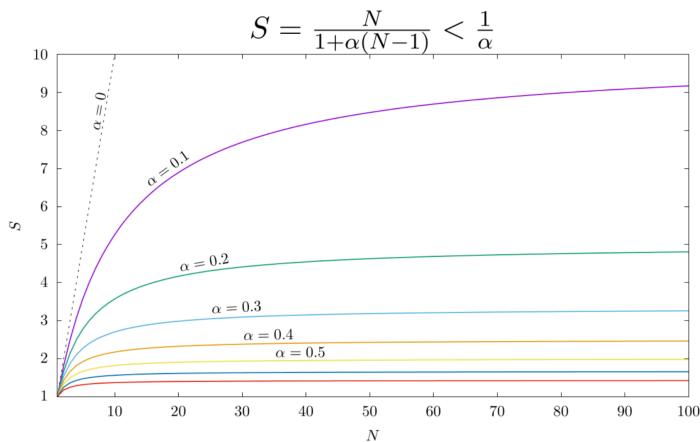
$$S = \frac{1}{\alpha + \frac{1-\alpha}{N}} = \frac{N}{1 + \alpha(N - 1)} \leq \frac{1}{\alpha}$$

Вывод: если разделяемая работа составляет 80%, вы не получите более чем пятикратное увеличение производительности за счёт параллелизации.

- Закон амдала иллюстрирует ограничение роста производительности вычислительной системы с увеличением количества вычислителей.



Закон Амдала: поведение



<https://habr.com/ru/articles/491946/>

- закон Густавсона-Барсиса

- Закон Густавсона-Барсиса оценивает максимально достижимое ускорение параллельной программы в зависимости от количества процессоров и доли последовательных расчетов.
- Ускорение масштабирования показывает, насколько эффективно могут быть организованы параллельные вычисления при увеличении сложности решаемых задач.
- Закон Густафсона отличается от закона Амдала тем, что учитывает увеличение объема выполненной задачи за постоянный промежуток времени.
- Ускорение масштабирования определяется как отношение объема вычислений, выполненных с использованием многопоточности, к объему вычислений, выполненных последовательно за один и тот же промежуток времени.

**Вопрос 14: CAP-теорема. Современные СУБД с точки зрения CAP-теоремы.
Классификация систем по признакам CAP. BASE свойства.**

- **cap-теорема**

В **CAP** говорится, что в распределенной системе возможно выбрать только 2 из 3-х свойств:

- C (consistency) — согласованность. Каждое чтение даст вам самую последнюю запись.
- A (availability) — доступность. Каждый узел (не упавший) всегда успешно выполняет запросы (на чтение и запись).
- P (partition tolerance) — устойчивость к распределению. Даже если между узлами нет связи, они продолжают работать независимо друг от друга.

- **современные СУБД с точки зрения CAP теоремы**

- Postgresql - PC

Postgresql

Postgresql действительно допускает множество различных конфигураций системы, поэтому их очень сложно описать. Давайте просто возьмем классическую Master-Slave репликацию с реализацией через Slony.

- Система работает в соответствии с ACID (существует пара проблем с двухфазным коммитом, но это вне рамок статьи).
- В случае разрыва связи, Slony попытается переключиться на новый Master, и у нас есть новый мастер с его согласованностью.
- Когда система функционирует в нормальном режиме, Slony делает все, чтобы достичь strong consistency. На самом деле, ACID — причина большой задержки в этой системе.
- Классификация системы — PC / EC (A).

- MongoDB - PA

MongoDB

Давайте узнаем что-то новое о MongoDB:

- Это ACID в ограниченном смысле на уровне документа.
- В случае распределенной системы — it's all about that BASE.
- В случае отсутствия разделений сети, система гарантирует, что чтение и запись будут согласованными.
- Если Master узел упадёт или потеряет связь с остальной системой, некоторые данные не будут реплицированы. Система выберет нового мастера, чтобы оставаться доступной для чтения и записи. (Новый мастер и старый мастер несогласованы).
- Система рассматривается как PA / EC (A), так как большинство узлов остаются CAP-available в случае разрыва. Помните, что в CAP MongoDB обычно рассматривается как CP. Создатель PAXELC, Даниэль Дж. Абади, говорит, что существует гораздо больше проблем с согласованностью, чем с доступностью, поэтому PA.

- Cassandra - PA

Cassandra

- Предназначена для «скоростного» взаимодействия (low-latency interactions).
- ACID на уровне записи.
- В случае распределенной системы — it's all about that BASE.
- Если возникает разрыв связи, остальные узлы продолжают функционировать.
- В случае нормального функционирования — система использует уровни согласованности для уменьшения задержки.
- Система рассматривается как PA / EL (A).

- **классификация систем по признакам CAP**

При классификации выбирать приходится между CP, AP и P.

Большинство всех систем - просто P

Есть крутая альтернатива CAP - это PACELC.

- **BASE свойства** (аббр)

Это как acid на уровне cap систем(распределенных систем)

BASE — это своеобразный контраст ACID, который говорит нам, что истинная согласованность не может быть достигнута в реальном мире и не может быть смоделирована в высокомасштабируемых системах.

Что стоит за BASE:

- **Basic Availability.** Система отвечает на любой запрос, но этот ответ может быть содержать ошибку или несогласованные данные.
- **Soft-state.** Состояние системы может меняться со временем из-за изменений конечной согласованности.
- **Eventual consistency** (конечная согласованность). Система, в конечном итоге, станет согласованной. Она будет продолжать принимать данные и не будет проверять каждую транзакцию на согласованность.
- **очень хорошая статья**
<https://habr.com/ru/articles/328792/>

Вопрос 15: Классификация NoSQL СУБД. Описание характерных черт по каждому классу. Особенности работы с NoSQL.

- **классификация nosql СУБД**

Существует 4 основных типа NoSQL СУБД:

- Хранилище «**ключ-значение**». В нём есть большая хеш-таблица, содержащая ключи и значения. Примеры: Riak, Amazon DynamoDB.
- **Документо-ориентированное хранилище**. Хранит документы, состоящие из легированных элементов. Пример: CouchDB.
- **Колоночное хранилище**. В каждом блоке хранятся данные только из одной колонки. Примеры: HBase, Cassandra.
- Хранилище на основе графов(**графовые**). Сетевая база данных, которая использует узлы и рёбра для отображения и хранения данных. Пример: Neo4J.

- **Статейка:**

- <https://tproger.ru/translations/types-of-nosql-db>

- На самом деле их вроде можно кастомно классифицировать более детально: колоночные, графовые, строковые(реляционные), поисковые, документо-ориентированные, аналитические, ключ-значение, new-sql и др.

Пара известных примеров:

- документо-ор.: **MongoDB**
- поисковые: **Elasticsearch**
- ключ-значение(**Key-Value**): **Redis**
- **new-sql**: **YDB**(внутренняя разработка Яндекса)

- **описание характерных черт по каждому классу**

- Ключ-значение:

- Такие базы данных как правило используют хеш-таблицу, в которой находится уникальный ключ и указатель на конкретный объект данных. Существует понятие блока (bucket) — логической группы ключей, которые не группируют данные физически. В разных блоках могут быть идентичные ключи.
 - Ключ может быть синтетическим или автогенерированным, а значение может быть представлено строкой, JSON, блобом (BLOB, Binary Large Object, большой двоичный объект) и т.д.

- Документо-ор.:

- Данные, представленные парами ключ-значение, сжимаются как хранилище документов схожим с хранилищем «ключ-значение» образом, с той лишь разницей, что хранимые значения (документы) имеют определённую структуру и кодировку данных. XML, JSON и BSON — некоторые из стандартных распространённых кодировок.

- Колоночные:

- В колоночных NoSQL базах данные хранятся в ячейках, сгруппированных в колонки, а не в строки данных.

Колонки логически группируются в колоночные семейства. Колоночные семейства могут состоять из практически неограниченного количества колонок, которые могут создаваться во время работы программы или во время определения схемы. Чтение и запись происходит с использованием колонок, а не строк.

- Графовые:
 - Графовые структуры используются вместе с рёбрами, узлами и свойствами, что обеспечивает безындексную смежность. При использовании графового хранилища данные могут быть легко преобразованы из одной модели в другую.
 - Такие базы данных используют ребра и узлы для представления данных.
 - Узлы связаны между собой определенными отношениями, представленными ребрами между ними.
 - У узлов и отношений есть некоторые свойства.
- **особенности работы с nosql**
 - лучше всего пролистать статью:
<https://cloud.yandex.ru/ru/blog/posts/2022/10/nosql>
 - но на всякий случай краткая выжимка особенностей:
 - отсутствие единого стандарта
 - смягченные требования к транзакциям
 - кастомное управление
 - гибкость модели
 - доступность данных
 - лучшая масштабируемость
 - высокая производительность
 - экономия ресурсов

Вопрос 16: Хранилища данных. Предпосылки к созданию, общая концепция. ETL vs ELT. Принципы создания, модели данных. Отчетность, OLAP кубы.

- **хранилища данных**

Хранилище данных — это разновидность системы управления данными, которая обеспечивает поддержку бизнес-аналитики.
(DWH == Data Warehouse)

- **предпосылки к созданию**

- a. Нужно было решать задачи, которые решает хранилище
- b. Хранилища данных обеспечивают для компаний обширные преимущества, так как дают возможность анализировать большие объемы разнообразных данных, извлекать из них значительную ценность, а также хранить записи за прошедшие периоды.

Эволюция хранилища данных от анализа данных к ИИ и машинному обучению

Первые хранилища данных появились в конце 1980-х гг., и их задачей было обеспечить обмен данными между операционными БД (БД для поддержки бизнеса) и системами поддержки принятия решения (СППР). Первым хранилищам данных требовалось много копий. Большинство компаний использовали несколько СППР для различных потребностей. Хотя эти СППР обычно использовали одни и те же данные, процессы сбора, очистки и интеграции выполнялись для каждой из них по отдельности.

- **общая концепция**

- a. Обычно хранилище данных включает в себя следующие компоненты:

- реляционную базу данных для хранения данных и управления ими;
- решение для извлечения, загрузки и преобразования данных, которое служит для подготовки данных к анализу;
- средства статистического анализа, отчетности и глубинного анализа данных;
- инструменты анализа для визуализации данных и их представления для корпоративных пользователей.

- b. Характеристики:

- **Субъектно-ориентированность.** Хранилища можно использовать для анализа данных, которые относятся к одной теме или функциональной области (например, продажи).
- **Единообразие.** Хранилища данных обеспечивают целостность данных различных типов, полученных из разных источников.
- **Неизменность.** Элементы данных, помещенные в хранилище данных, не подвергаются изменениям.
- **Изменения во времени.** Анализ данных, помещенных в хранилище данных, предназначен для выявления изменений в закономерностях, возникающих со временем.

- c. статьи:

- <https://www.oracle.com/cis/database/what-is-a-data-warehouse/>
- <https://cloud.yandex.ru/ru/blog/posts/2022/06/data-warehouse>

- **ETL vs ELT**

- a. **ETL** (Extract, Transform, Load) — это трёхэтапный процесс управления данными, в дословном переводе значит «извлечение, преобразование, загрузка». Сначала извлекается информация из структурированных и неструктурных источников, затем преобразовывается в нужный формат и загружается в место назначения.
- b. **ELT** — это аббревиатура от extraction (извлечения), loading

(загрузки) и transformation (преобразования). Это процесс интеграции данных, при котором из различных источников извлекается сырья информация (в её исходных форматах), загружается непосредственно в центральный репозиторий, будь то облачное хранилище данных, озеро данных или data lakehouse, где происходит преобразование данных в подходящие форматы для дальнейшего анализа и отчётности.

- c. Основное различие между ними заключается в том, ГДЕ и КОГДА выполняются преобразование и загрузка данных. При **ETL** данные преобразуются на временном этапе подготовки до того, как попадут в целевой репозиторий (например, в корпоративное хранилище данных), в то время как **ELT** позволяет преобразовывать данные после их загрузки в целевую систему (облачные хранилища данных или озёра данных).

- d. [статья:](#)

- <https://habr.com/ru/articles/695546/>

- **принципы создания**

- a. Трехуровневая архитектура:

- нижний уровень — базы данных, которые объединяют данные из различных источников;
 - средний уровень — сервисы и приложения, которые преобразуют данные в специальную структуру для анализа и сложных запросов;
 - верхний уровень — инструменты для создания отчетов, визуализации и анализа данных.

- b. Модели проектирования:

- модель Инмона: данные из источников поступают в хранилище после процесса ETL;
 - модель Кимбалла: после процесса ETL данные загружаются в витрины данных, а объединение витрин создает концептуальное хранилище данных.

- c. Учет специфики данных, взаимосвязей внутри групп данных, связей между ними, типов преобразования данных, частоты обновления, взаимосвязи между объектами хранилища, процессов передачи, резервного копирования, восстановления.

- **модели данных**

Различают кстати не только модели, но и подходы, а именно облачные и нет.

Существует несколько моделей хранилищ данных. Вот некоторые из них:

- a. Реляционные базы данных (базы данных SQL) — обеспечивают построчное хранение данных в таблицах, что подразумевает строгую структуру данных.
- b. Базы данных NoSQL — могут быть документоориентированными, графовыми или key-value-хранилищами.
- c. Key-value-хранилища — хранят данные в виде хеш-таблицы, где у каждой записи есть только один индекс.

- d. Графовые базы данных — хранят все данные в виде узлов и связей между ними.
- e. Базы данных OLAP — хранят данные в колонкоориентированной модели, что позволяет эффективно сжимать данные и строить над ними различные агрегаты.

- **отчетность**

Тут просто наверное стоит сказать, что отчетность - это часть задач, решаемых DWH и берется она из агрегаций данных, которые хранилище собирает для аналитики.

Можно добавить пример, например из IBM:

<https://www.ibm.com/docs/ru/engineering-lifecycle-management-suite/lifecycle-management/6.0.2?topic=overview-data-warehouse>

- **OLAP кубы**

- a. OLAP (OnLine Analytical Processing) — оперативная аналитическая обработка данных или анализ данных в реальном времени.

Как работают OLAP-системы

OLAP состоит из нескольких компонентов:

- Хранилище данных.
- Средства ETL.
- Сервер.
- Аналитические инструменты.

Хранение данных

Данные появляются вне аналитической системы и имеют самую разную форму и представление. А для оперативной обработки структура данных должна быть оптимизирована под ее особые требования. Решение этой проблемы — специальные хранилища, где данные заранее импортированы из различных источников, а затем очищены, преобразованы в нужный формат и упорядочены по заданному принципу — в многомерные или классические реляционные базы данных.

Многомерные хранилища

Такая система называется **MOLAP**. Для хранения строится OLAP-куб — многомерный массив данных, упорядоченный по измерениям или категориям. С помощью последних создаются информативные сводные таблицы. В центре куба расположена двумерная таблица фактов, которые характеризуют взаимодействие элементов из разных измерений. MOLAP — самый быстрый вид аналитических систем: сервер напрямую извлекает из куба меры, которые соответствуют поступающим запросам.



- b. <https://cloud.yandex.ru/ru/docs/glossary/olap>
- c. <https://habr.com/ru/companies/vk/articles/703508/>