МСХОАД

Мобилки конченные (ну по базе), даже док с ответами нормально написать не могут.

1. Логическое проектирование систем хранения данных. Сбор информации. Документация. Минимальные информационные требования. Источники для выявления правил данных.

Логическое проектирование систем хранения данных — это этап проектирования базы данных, на котором **разрабатывается абстрактная модель хранения информации**, **независимая от конкретной СУБД**, но уже структурированная в виде таблиц, сущностей, атрибутов и связей между ними.
Это **мост между бизнес-процессами и физической реализацией БД**, на котором информация из предметной области переводится в чёткие логические структуры.

Логическое проектирование отвечает на вопрос:
"Какие данные нужно хранить и как они связаны между собой?"

Что включает логическое проектирование:

1. Определение сущностей
- Что мы хотим описывать? (например: Клиенты, Заказы, Товары)

2. Определение атрибутов
- Какие характеристики у этих объектов? (например: имя клиента, дата заказа, цена товара)

3. Установление связей между сущностями
- Например: один клиент может делать много заказов (связь один-ко-многим).

4. Формализация бизнес-правил
- Например: "Клиент не может оформить заказ без e-mail", или "Товар не может стоить меньше нуля".

5. Нормализация данных
- Разбиение на таблицы, чтобы исключить дублирование и зависимость данных.

Зачем нужно логическое проектирование:
- Чтобы **понять структуру данных** до начала физической реализации.
- Чтобы избежать проблем в будущем: дублирования, ошибок в связях, уязвимостей.
- Чтобы **отделить логику от технической реализации** (мы пока не думаем о MySQL, PostgreSQL, MongoDB и т. д.).
- Чтобы согласовать модель с бизнес-требованиями.

Отличие от других этапов:

Этап	Описание
Концептуальное проектирование	Модель в терминах предметной области, на уровне бизнес-объектов.
Логическое проектирование	Преобразование концептуальной модели в таблицы, связи, атрибуты.
Физическое проектирование	Конкретная реализация в СУБД (типы данных, индексы, оптимизация и т. д.).

Сбор данных:

OLTP – данные, получаемые в результате повседневных транзакций. Хранилища данных – поддержка принимаемых решений.

Основные принципы: Предварительные вычисления. Частота обновления данных зависит от потребностей пользователей. Объединение данных из нескольких источников.

Документация:

Перечень границ проекта. Перечень отрицательного опыта пользователей. Запросы пользователей. Поддержка общедоступного хранилища документации с версионностью.

Минимальные информационные требования

Это **набор критически необходимых данных**, без которых система не сможет выполнять свои функции. Например:

- Для системы учета клиентов:
    - Уникальный ID клиента.
    - Имя.
    - Контактная информация.

Цель: избежать перегрузки системы ненужными данными, фокусироваться на минимально жизнеспособной информации (MVD – Minimum Viable Data).

Источники для выявления правил данных

1. **Существующие базы данных** (анализ полей, ограничений, связей).
2. **Бизнес-документы** (договоры, инструкции, стандарты).
3. **Бизнес-аналитики и специалисты предметной области**.
4. **Правила и нормативы** (внешние: GDPR, ISO; внутренние политики).
5. **Анализ логов и поведения пользователей**.
6. **Пользовательские интерфейсы** (формы, отчеты — что заполняется, как фильтруется).

2. Логическое проектирование систем хранения данных. Реляционная и нереляционная модели. Сущности атрибуты, отношения и бизнес-правила. Методологии моделирования. Нормализация и денормализация. План проекта использования данных.

Реляционная и нереляционная модели

Реляционная модель:

- Основана на **таблицах** (relations), где каждая таблица состоит из **строк (записей)** и **столбцов (атрибутов)**.

- Обеспечивает строгую **структуру данных**.
- Применяются **ключи** (первичный, внешний).
- Примеры СУБД: PostgreSQL, MySQL, Oracle.

Плюсы:
- Ясная структура.
- Поддержка транзакций (ACID).
- Надежность и консистентность данных.

Минусы:
- Не всегда удобна для хранения иерархий, документов, медиа.
- Меньшая гибкость при изменении схемы.

Нереляционная модель (NoSQL):
- Гибкие структуры: **документы, графы, ключ-значение, колоночные хранилища**.
- Примеры: MongoDB (документ), Redis (key-value), Cassandra (column-family), Neo4j (граф).

Плюсы:
- Масштабируемость.
- Высокая производительность.
- Гибкость в моделировании.

Минусы:
- Слабая согласованность данных.
- Нет единого стандарта.
- Ограниченные возможности запросов (зависит от типа хранилища).

Сущности, атрибуты, отношения и бизнес-правила

Сущность (Entity) — объект реального мира, о котором нужно хранить данные.
Примеры: Клиент, Заказ, Товар.

Атрибут (Attribute) — характеристика сущности.
Примеры: Имя клиента, Дата заказа, Цена товара.

Связи (Relations) — отношения между сущностями.
- **Один-к-одному (1:1)** — например, Клиент ↔ Паспорт.
- **Один-ко-многим (1:N)** — Клиент → Заказы.
- **Многие-ко-многим (M:N)** — Студенты ↔ Курсы (через таблицу записей).

Бизнес-правила — ограничения и правила, регулирующие данные:
- "Каждому заказу должен соответствовать клиент."
- "Цена товара не может быть отрицательной."
- "Клиент не может делать заказ без подтверждённой электронной почты."

Методологии моделирования
ER-модель (Entity-Relationship)
- Основа логического проектирования реляционных БД.
- Используются **ER-диаграммы**.

UML (Unified Modeling Language)
- Диаграммы классов, активности, состояний.
- Подходит для системного и объектного проектирования.

IDEF1X — методология структурного моделирования данных.

- Часто используется для БД в промышленности.

Star Schema и Snowflake Schema

- Применяются в хранилищах данных.
- Подходы к построению аналитических моделей (факты и измерения).

Нормализация и денормализация

Нормализация — процесс устранения избыточности и зависимости.

- **1NF** — атомарность.
- **2NF** — отсутствие частичной зависимости.
- **3NF** — отсутствие транзитивных зависимостей.
- **BCNF, 4NF, 5NF** — для продвинутых случаев.

Плюсы:

- Целостность.
- Лёгкость обновления.

Минусы:

- Сложность запросов.
- Большое количество JOIN-ов.

Денормализация — обратный процесс: добавление избыточности ради производительности.

Применяется в OLAP, NoSQL, при кэшировании.

План проекта использования данных

Цель: определить, как, кем и где будут использоваться данные.

Основные этапы:

1. **Каталог данных** — какие данные хранятся.
2. **Карты потоков данных** — как данные перемещаются между системами.
3. **План доступа** — кто имеет доступ к каким данным.
4. **Безопасность и соответствие** — защита персональных данных.
5. **Хранение и архивирование** — жизненный цикл данных.
6. **Резервное копирование и восстановление**.
7. **Метрики качества данных** — корректность, полнота, актуальность.

3. Физическое проектирование и реализация. Проблемы использования данных, связанные с размером данных. Влияние физических характеристик.

Физическое проектирование — это этап, на котором логическая модель преобразуется в **конкретную реализацию в СУБД** с учетом **аппаратных ресурсов**, **структур хранения**, **индексов**, **оптимизаций** и **реального объёма данных**.

Проблемы использования данных, связанные с размером данных

1. **Рост объема данных**:
    - Увеличивает время выполнения запросов.
    - Требует больше места на дисках.
    - Усложняет резервное копирование, индексацию, миграцию.
2. **Раздутие таблиц и медленные запросы**:
    - Без индексов сканируются все строки.
    - JOIN-ы и агрегации становятся дорогими по ресурсам.
3. **Узкие места**:
    - Диски: нехватка пространства, скорость чтения/записи.

- Память: кэш не умещает все данные.
- CPU: сложные запросы/агрегации перегружают процессор.

4. **Фрагментация**:
   - Физические файлы таблиц и индексов могут стать фрагментированными, что замедляет работу.
5. **Архивация и удаление**:
   - Без политики жизненного цикла данные копятся бесконечно, снижая производительность.

Влияние физических характеристик

1. **Типы носителей**:
   - HDD — дешевле, но медленнее.
   - SSD — дороже, но быстрее (подходит для индексов, горячих данных).
2. **Сетевые задержки** (в распределённых системах):
   - Медленные сети между узлами — причина долгих запросов.
3. **Количество CPU и оперативной памяти**:
   - Мало памяти — нет места для буферов и кэшей.
   - Недостаточный CPU — тормозит при агрегациях и сложных фильтрах.
4. **RAID и конфигурации хранения**:
   - RAID 10 — быстро и надёжно (для OLTP).
   - RAID 5 — экономно, но медленнее при записи.

4. Физическое проектирование и реализация. Основные топологии приложений. Схема физического проектирования. Обеспечение целостности данных. Расширенный доступ к данным.

Основные топологии приложений

1. **Мононолитная**:
   - Всё в одном: приложение, БД, веб-сервер.
   - Подходит для простых решений.
   - Проблема: масштабировать трудно.
2. **Клиент-сервер**:
   - Отдельные серверы БД и клиентские приложения.
   - Классическая архитектура.
3. **Трёхуровневая**:
   - UI ↔ Логика ↔ База данных.
   - Масштабируемая, управляемая.
4. **Микросервисная архитектура**:
   - Каждый сервис использует собственную БД (или общее хранилище).
   - Легче масштабировать, но сложнее администрировать.

Схема физического проектирования

Это фактическое описание того, **как логическая модель реализуется в конкретной СУБД**:

1. **Типы данных**:
   - VARCHAR, INT, DATETIME и т. д. — подобраны по объему и типу использования.
2. **Индексы**:
   - Первичные ключи.
   - Вторичные индексы (на часто фильтруемые поля).
   - Комбинированные (по нескольким столбцам).

3. **Табличные пространства (tablespaces)**:
    - Логическое распределение таблиц и индексов по физическим дискам.
4. **Партиционирование таблиц**:
    - Горизонтальное (по дате, по региону).
    - Улучшает доступ к «актуальным» данным.
5. **Материализованные представления**:
    - Для кеширования результатов сложных запросов.

Обеспечение целостности данных

1. **Ограничения целостности**:
    - PRIMARY KEY, FOREIGN KEY, UNIQUE, CHECK.
2. **Триггеры**:
    - Реакция на изменения — контроль логики на уровне БД.
3. **Транзакции и уровни изоляции**:
    - Обеспечивают согласованность при одновременном доступе:
        - Read Uncommitted
        - Read Committed
        - Repeatable Read
        - Serializable

Расширенный доступ к данным

1. **API-доступ** (REST, GraphQL).
2. **Федеративные запросы** — объединение данных из нескольких источников.
3. **ETL-пайплайны** — загрузка, очистка и перемещение данных между системами.
4. **Репликация** — чтение с копий, балансировка нагрузки.

5. Физическое проектирование и реализация. Обеспечение безопасности. Определение требований к аппаратным средствам. Определение параметров роста данных. Определение параметров архивирования.

Обеспечение безопасности

1. **Аутентификация и авторизация**:
    - Роли пользователей и уровни доступа (GRANT, REVOKE).
2. **Шифрование**:
    - **На уровне диска** (Transparent Data Encryption).
    - **На уровне поля** — для персональных данных (PII).
3. **Журналы аудита**:
    - Кто, когда и что делал с данными.
4. **Сетевые меры**:
    - VPN, firewall, ограничение доступа по IP.

Определение требований к аппаратным средствам

На основании:

- Количества пользователей.
- Ожидаемых объёмов данных.
- Частоты запросов.

Компоненты:

- **CPU** — обрабатывает запросы, агрегации.
- **RAM** — буферизация, кеширование.
- **Диски** — скорость чтения/записи, объём.
- **Сетевые интерфейсы** — для кластеров и бэкапов.

Определение параметров роста данных

Планируется:

- **Годовой прирост данных**.
- **Пиковые нагрузки**.

- **Резерв на рост в 2–3 раза**.

Инструменты:

- Исторический анализ (если уже есть система).
- Прогноз на основе бизнес-плана.

Определение параметров архивирования

1. **Архивирование неактивных данных**:
   - Например, заказы старше 3 лет → в отдельную таблицу или хранилище.
2. **Политики хранения**:
   - Хранить активные данные 2 года.
   - Архив — 5 лет.
   - Удаление — через 7 лет.
3. **Форматы архивов**:
   - CSV, JSON, Parquet (для хранения в хранилищах данных, типа S3, HDFS).
4. **Доступ к архиву**:
   - Только для админов/аудиторов.
   - С возможностью восстановления в основной слой.

6. Иерархии. Виды организации иерархических данных. Иерархический тип данных в SQL Server.

Что такое иерархические данные?

Иерархические данные — это данные, которые организованы в виде **древа**, где один элемент (родитель) может содержать несколько подчинённых (потомков).

Примеры:

- Структура компании: Директор → Отдел → Сотрудник.
- Категории товаров: Электроника → Смартфоны → iPhone.

Виды организации иерархических данных

1. **Adjacency List (Список смежности)**
   Каждый узел содержит ссылку на родителя (наиболее распространённый способ):

```
CREATE TABLE Categories (
   ID INT PRIMARY KEY,
   Name NVARCHAR(100),
   ParentID INT REFERENCES Categories(ID)
);
```

2. **Path Enumeration (Хранение пути)**
   Каждый узел хранит полный путь к себе:

Path: /Root/Category/Subcategory/Item

3. **Nested Sets (Вложенные множества)**
   Храним левую и правую границы:

```
lft | rgt
----|----
  1 | 10
  2 | 3
  4 | 9
```

4. **Closure Table (Таблица замыканий)**
   Отдельная таблица, хранящая **все пути** от предка к потомку.

Иерархический тип данных в SQL Server: HierarchyID

SQL Server предоставляет специальный тип данных: **HierarchyID** Это **компактное бинарное представление** пути от корня до узла.

```
CREATE TABLE Employees (
   EmployeeID INT PRIMARY KEY,
   Name NVARCHAR(100),
   OrgNode HIERARCHYID
);
```

Преимущества:

- Упрощает работу с деревьями.
- Поддерживает методы:
    - .GetAncestor(n) — получить родителя на n уровней выше.
    - .GetDescendant(child1, child2) — вставка между двумя потомками.
    - .IsDescendantOf() — проверка потомков.
    - .GetLevel() — уровень узла.

Пример использования:

```
SELECT Name
FROM Employees
WHERE OrgNode.IsDescendantOf(hierarchy::GetRoot()) = 1;
```

7. Иерархии. Хранение иерархических данных. Иерархические запросы в Oracle.

Хранение иерархических данных в Oracle
Чаще всего используется **adjacency list** (как и в SQL Server):

```
CREATE TABLE Departments (
   Dept_ID NUMBER PRIMARY KEY,
   Dept_Name VARCHAR2(100),
   Parent_Dept_ID NUMBER REFERENCES Departments(Dept_ID)
);
```

Иерархические запросы в Oracle: CONNECT BY
Oracle имеет собственный синтаксис для иерархических запросов:

```
SELECT Dept_Name
FROM Departments
START WITH Parent_Dept_ID IS NULL
CONNECT BY PRIOR Dept_ID = Parent_Dept_ID;
```

Start with – Корень иерархии
Connect by – родительская связь
Prior – определение направление: родитель-потомок или наоборот
LEVEL — встроенная псевдоколонка, показывающая уровень в иерархии

Возможность обхода в разных направлениях:

- Сверху вниз: CONNECT BY PRIOR id = parent_id
- Снизу вверх: CONNECT BY id = PRIOR parent_id

8. Обобщенные табличные выражения. Применение ОТВ. Рекурсивные ОТВ.

Что такое ОТВ (Common Table Expression / CTE)
ОТВ (CTE) — временный результат SQL-запроса, определённый в начале выражения и использующийся в основном запросе как таблица.
Синтаксис:

```
WITH CTE_NAME AS (
   SELECT ...
)
SELECT * FROM CTE_NAME;
```

Применение CTE
- Разбиение сложных запросов на части.
- Множественное использование одной выборки.
- Замена подзапросов.
- Упрощение агрегаций и оконных функций.
- **Рекурсивные запросы** (иерархии!).

Рекурсивные ОТВ
Позволяют выполнять **рекурсивный обход иерархий** (например, получение всех подчинённых сотрудника).

Обязательные компоненты:
- Базовый SELECT.
- UNION ALL.
- Рекурсивная часть, ссылающаяся на CTE.

9. Графы. Графовые базы данных. Ключевое слово MATCH.

Что такое граф?
Граф — это структура данных, состоящая из:
- **Вершин (nodes)** — объекты (люди, товары, страницы, устройства)
- **Рёбер (edges)** — связи между объектами (дружба, ссылка, покупка)

Пример:
(Alice) ──[друг]──> (Bob)

Графовые базы данных (Graph DB)
Это БД, оптимизированные для хранения и обработки **связанных данных**.
Поддерживают:
- Быстрый обход по связям (например, рекомендательные системы).
- Гибкие схемы (узлы и связи могут иметь произвольные свойства).
- Графовые запросы с помощью языка, ориентированного на реляции (SQL) или графы (Cypher, Gremlin).

Примеры графовых БД:

СУБД	Особенности
Neo4j	Популярная графовая БД с языком Cypher.
SQL Server 2017+	Поддержка **графовых таблиц** (узлы и связи), SQL + MATCH.
Oracle 21c+	Oracle PGX и встроенная графовая аналитика.
PostgreSQL	Расширения: Apache AGE, agensgraph.

Графовые таблицы в SQL Server

- **Node Table**: таблица вершин
- **Edge Table**: таблица рёбер, которая ссылается на две вершины

```
CREATE TABLE People (
   ID INT PRIMARY KEY,
   Name NVARCHAR(100)
) AS NODE;

CREATE TABLE Friends (
   ID INT PRIMARY KEY
) AS EDGE;
```

Ключевое слово MATCH

Оператор MATCH используется в SQL Server **для обхода графа**:

```
SELECT p1.Name, p2.Name
FROM People p1, Friends f, People p2
WHERE MATCH(p1-(f)->p2);
```

10. Расширенные группировки. Применение расширенных группировок в SQL Server и в Oracle.

Что такое расширенные группировки?

Расширенные группировки позволяют строить **сводные таблицы и агрегаты по подмножествам группировок**, включая **подсчёты по нескольким измерениям сразу**.

Поддержка:

СУБД	Расширения
SQL Server	GROUPING SETS, CUBE, ROLLUP, GROUPING()
Oracle	То же самое — с поддержкой ROLLUP, CUBE, GROUPING SETS

ROLLUP

Создает агрегации **по иерархии уровней** (суммы по подгруппам и общая сумма).
sql

```
SELECT Region, City, SUM(Sales)
FROM SalesData
GROUP BY ROLLUP(Region, City);
```

Результат:

- Группировка по (Region, City)
- Группировка по Region (без City)
- Общая сумма (NULL, NULL)

CUBE

Создает агрегации **по всем возможным комбинациям** полей.

```
SELECT Product, Region, SUM(Sales)
FROM SalesData
GROUP BY CUBE(Product, Region);
```

Результат:

- Product + Region
- Product (все регионы)
- Region (все продукты)
- Общая сумма

GROUPING SETS

Полный контроль — можно указать, **какие именно комбинации** нужны:

```
SELECT Region, Product, SUM(Sales)
FROM SalesData
GROUP BY GROUPING SETS (
   (Region),
   (Product),
   (Region, Product)
);
```

GROUPING() и GROUPING_ID()

Позволяют определить, где в результате стоит агрегат (NULL как реальное значение или как результат группировки).

```
SELECT
  Region,
  Product,
  SUM(Sales),
  GROUPING(Region) AS IsRegionTotal,
  GROUPING(Product) AS IsProductTotal
FROM SalesData
GROUP BY CUBE(Region, Product);
```

11. Аналитические функции. Виды аналитических функций. Синтаксис аналитических функций.

Аналитические функции (OLAP-функции) — это функции, которые **рассчитываются на основе окна (partition)** данных, **не сворачивая всю таблицу** в одну строку (в отличие от агрегатных функций).

Примеры аналитических задач:

- Найти зарплату каждого сотрудника и среднюю по отделу.
- Пронумеровать строки внутри групп.
- Посчитать долю строки в общей сумме.
- Рассчитать скользящее среднее.

Синтаксис аналитической функции:

```
ФУНКЦИЯ(параметры) OVER (
   PARTITION BY ...  -- разбиение на группы
   ORDER BY ...      -- порядок внутри группы
   ROWS ...          -- рамка окна (опционально)
)
```

Виды аналитических функций:

Категория	**Функции**
Агрегации	SUM(), AVG(), MIN(), MAX(), COUNT()
Ранжирование	ROW_NUMBER(), RANK(), DENSE_RANK(), NTILE(n)
Навигационные	LAG(), LEAD(), FIRST_VALUE(), LAST_VALUE()
Статистические	CUME_DIST(), PERCENT_RANK(), PERCENTILE_CONT()

Примеры
Пример 1: ROW_NUMBER()

```
SELECT
  EmployeeID,
  DepartmentID,
  Salary,
  ROW_NUMBER() OVER (PARTITION BY DepartmentID ORDER BY Salary DESC) AS
RankInDept
FROM Employees;
```

Назначает номер строки **внутри каждого отдела**, упорядочивая по зарплате.

Пример 2: LAG() / LEAD()

```
SELECT
  Year,
  Sales,
  LAG(Sales, 1) OVER (ORDER BY Year) AS PrevYearSales,
  LEAD(Sales, 1) OVER (ORDER BY Year) AS NextYearSales
FROM SalesData;
```

Показывает значение **предыдущего** и **следующего года** для каждой строки.

Пример 3: Сумма с накоплением (cumulative sum)

```
SELECT
  EmployeeID,
  Salary,
  SUM(Salary) OVER (ORDER BY EmployeeID ROWS BETWEEN UNBOUNDED
PRECEDING AND CURRENT ROW) AS CumulativeSalary
FROM Employees;
```

12. Преобразование данных в SQL. Применение оператора MERGE. Применение операторов PIVOT и UNPIVOT.

Преобразование данных — это изменение структуры, формата или значения данных для соответствия требованиям хранения, анализа или отображения. В SQL используются:

- Операторы: CAST(), CONVERT(), COALESCE(), ISNULL()
- Преобразование строк и чисел: CONVERT(INT, '123')
- Преобразование дат: CAST('2025-06-01' AS DATETIME)
- Условные конструкции: CASE, IIF

2. Оператор MERGE
Оператор MERGE используется для **слияния данных из одной таблицы в другую**. Он позволяет **объединить операции INSERT, UPDATE, DELETE** в одном выражении.
Синтаксис:

```
MERGE INTO TargetTable AS T
USING SourceTable AS S
ON T.ID = S.ID
WHEN MATCHED THEN
   UPDATE SET T.Name = S.Name
WHEN NOT MATCHED THEN
   INSERT (ID, Name) VALUES (S.ID, S.Name)
WHEN NOT MATCHED BY SOURCE THEN
   DELETE;
```

Применение:

- Синхронизация таблиц
- Обновление данных при импорте
- Вставка новых строк и удаление устаревших

3. Оператор PIVOT
PIVOT используется для **преобразования строк в столбцы**. Это удобно при создании сводных таблиц.
Синтаксис:

```
SELECT Product, [2019], [2020], [2021]
FROM (
  SELECT Product, Year, Sales
  FROM SalesData
) AS SourceTable
PIVOT (
  SUM(Sales) FOR Year IN ([2019], [2020], [2021])
) AS PivotTable;
```

Применение:

- Аналитические отчёты
- Сводные таблицы
- Сравнение значений по категориям

4. Оператор UNPIVOT
UNPIVOT преобразует **столбцы в строки** — обратная операция к PIVOT.
Синтаксис:

```
SELECT Product, Year, Sales
FROM (
  SELECT Product, [2019], [2020], [2021]
  FROM SalesData
) AS SourceTable
UNPIVOT (
  Sales FOR Year IN ([2019], [2020], [2021])
) AS Unpivoted;
```

Применение:

- Подготовка данных для анализа
- Преобразование таблиц с фиксированной структурой в нормализованный вид

13. Объектные типы данных в SQL Server. Применение объектных типов данных. Объектные типы данных в SQL Server.

Объектные типы данных позволяют определить **составные (комплексные) типы**, которые могут содержать несколько полей. Это элементы **объектно-ориентированного расширения T-SQL**.
В SQL Server не реализована полноценная система пользовательских типов, как в Oracle, но доступны:
\

1.1. Табличные типы (Table Types)
Определяются один раз, могут использоваться как параметр в процедурах и функциях.

```
CREATE TYPE dbo.MyTableType AS TABLE (
   ID INT,
   Name NVARCHAR(100)
);
```

1.2. CLR-объекты (через .NET сборки)
Можно создать тип на C# и загрузить в SQL Server.

2. Применение объектных типов данных

- Передача таблиц в хранимые процедуры
- Моделирование сложных структур (например, координаты, точки)
- Расширение стандартной функциональности (например, геометрия, работа с JSON, шифрование)

3. Ограничения

- Не поддерживаются наследование и полиморфизм
- Табличные типы нельзя использовать в качестве столбца обычной таблицы (только как параметр)

14. Сборки. Применение сборок в SQL Server. Виды программных конструкций, используемых в сборках. Регистрация сборок. Использование сборок.

Сборка (assembly) — это **библиотека .NET (DLL)**, зарегистрированная в SQL Server для использования в виде:

- Хранимых процедур (CREATE PROCEDURE)
- Пользовательских функций (CREATE FUNCTION)

- Триггеров (CREATE TRIGGER)
- Пользовательских типов (CREATE TYPE)
- Агрегатных функций (CREATE AGGREGATE)

2. Применение сборок
- Расширение функциональности T-SQL через C#, VB.NET
- Работа с системами шифрования, файлами, сетевыми протоколами
- Обработка сложных структур (например, JSON, XML, криптография)
- Повышение производительности вычислений, невозможных или неэффективных в T-SQL

3. Виды программных конструкций
В сборках можно реализовать:
- **Процедуры (SqlProcedure)**
- **Функции (SqlFunction)**
- **Триггеры (SqlTrigger)**
- **Агрегаты (SqlUserDefinedAggregate)**
- **Пользовательские типы (SqlUserDefinedType)**

4. Регистрация сборки

Этапы:
1. **Включение возможности CLR:**

```
sp_configure 'clr enabled', 1;
RECONFIGURE;
```

2. **Создание ассембли:**

```
CREATE ASSEMBLY MyAssembly
FROM 'C:\MyAssembly.dll'
WITH PERMISSION_SET = SAFE;
```

3. **Создание объектов:**

```
CREATE FUNCTION dbo.MyFunction(@x INT)
RETURNS INT
AS EXTERNAL NAME MyAssembly.[MyNamespace.MyClass].MyMethod;
```

5. Параметры PERMISSION_SET
- SAFE: максимальная безопасность, без доступа к внешним ресурсам
- EXTERNAL_ACCESS: доступ к файловой системе, сети и т.п.
- UNSAFE: полный доступ, но требует подписания сборки

6. Использование сборок
После регистрации и создания функций/процедур можно использовать их как обычные объекты T-SQL.
Пример вызова:

```
SELECT dbo.MyFunction(100);
```

7. Ограничения
- Требуется знание .NET
- Повышенный контроль безопасности (администрирование)
- Возможны проблемы при обновлениях версий .NET

15. Пространственные данные. ГИС-приложения. Импорт пространственных данных из ГИС-систем. Виды пространственных данных.

1. Пространственные данные
Пространственные данные (spatial data) представляют объекты, имеющие географическое или геометрическое положение, форму и размер на плоскости или на сфере. В СУБД такие данные используются для моделирования карт, границ, маршрутов, зданий и других геообъектов.

2. ГИС-приложения
ГИС (Географические информационные системы) — это программное обеспечение, предназначенное для:
- Сбора, хранения, анализа, визуализации пространственных данных.
- Примеры: **ArcGIS, QGIS, MapInfo**, а также **PostGIS, GeoServer**.

СУБД может служить **хранилищем пространственных данных**, предоставляя функции хранения, индексации и запроса пространственных объектов.

3. Импорт пространственных данных из ГИС-систем
Основные форматы:
- **Shapefile (.shp)**
- **GeoJSON**
- **KML (Google Earth)**
- **GML (XML-формат OGC)**
- **WKT/WKB** (Well-Known Text/Binary)

Импорт в СУБД:
- Используются ETL-инструменты: **FME, ogr2ogr (из GDAL), SSIS**
- В SQL Server: geometry::STGeomFromText(...), geography::STGeomFromText(...)

4. Виды пространственных данных
В SQL Server и Oracle они делятся на два типа:
a. Геометрические (geometry)
- Используются для **плоской (евклидовой) системы координат**
- Примеры: линии, полигоны на 2D-плоскости (например, план здания)

b. Географические (geography)
- Используются для **сферической модели Земли**
- Примеры: координаты GPS, границы стран

Поддерживаются следующие **объекты**:
- POINT
- LINESTRING
- POLYGON
- MULTIPOINT, MULTILINESTRING, MULTIPOLYGON
- GEOMETRYCOLLECTION

16. Пространственные данные. SRID. Методы и свойства пространственных данных. Индексирование пространственных данных.

1. SRID (Spatial Reference Identifier)
- **SRID** — идентификатор системы координат.
- Обеспечивает согласованное толкование координат.
- Например:
  - SRID = 4326: WGS 84 (широта/долгота)
  - SRID = 0: произвольная евклидова система (обычно для geometry)

Нельзя выполнять пространственные операции между объектами с разными SRID.

2. Методы и свойства пространственных данных (SQL Server)
Методы:

Категория	Примеры
Создание	STGeomFromText, STPointFromText
Информация	STArea(), STLength(), STNumPoints()
Геометрия	STUnion(), STIntersection(), STBuffer()
Проверка отношений	STIntersects(), STContains(), STEquals()

Пример:
```
DECLARE @g geometry = geometry::STGeomFromText('POLYGON((0 0, 0 2, 2 2, 2 0, 0 0))', 0);
SELECT @g.STArea(); -- возвращает площадь
```

3. Индексирование пространственных данных
Пространственные индексы — это **специализированные структуры (например, R-tree, Quad-tree)**, позволяющие ускорить поиск пространственных объектов.
SQL Server:

```
CREATE SPATIAL INDEX idx_geo
ON SpatialTable(GeoCol)
USING GEOMETRY_GRID
WITH (BOUNDING_BOX = (-180, -90, 180, 90));
```

Oracle:
```
CREATE INDEX spatial_idx
ON spatial_table(geometry)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

17. Объектные типы данных в Oracle. Атрибуты. Методы. Объектные таблицы. Объектные представления.

1. Объектные типы в Oracle
Объектные типы позволяют создать **составной тип данных с атрибутами и методами**.
Пример объявления типа:
sql

```
CREATE TYPE PersonType AS OBJECT (
  id     NUMBER,
  name   VARCHAR2(100),
  MEMBER FUNCTION get_name RETURN VARCHAR2
);
```

---

2. Атрибуты и методы
- **Атрибуты** — поля объекта (аналог переменных-членов).
- **Методы** — функции и процедуры, определённые внутри объекта.
- **Типы методов**:
  - MEMBER: работает с конкретным экземпляром
  - STATIC: работает на уровне класса

Тело метода:
sql

```
CREATE TYPE BODY PersonType AS
  MEMBER FUNCTION get_name RETURN VARCHAR2 IS
  BEGIN
    RETURN name;
```

```
  END;
END;
```

---

3. Объектные таблицы
Таблица, строки которой представляют **экземпляры объекта**.
sql

```
CREATE TABLE persons OF PersonType;
```

Позволяет обращаться к атрибутам: SELECT p.name FROM persons p;

---

4. Объектные представления
Объектное представление — это **view**, основанное на объектных типах, которое отображает строковые таблицы как объектные.
sql

```
CREATE VIEW person_view OF PersonType
WITH OBJECT IDENTIFIER (id)
AS SELECT id, name FROM person_table;
```

Используется при миграции с реляционной модели на объектную.

18. Объектные типы данных в Oracle. Индексирование объектных данных.

1. Индексирование объектных таблиц
Индексы можно создавать на:

- **Атрибутах объектов**
- **Методах (через функциональные индексы)**

Индекс по атрибуту:
sql

```
CREATE INDEX idx_person_name ON persons(name);
```

Индекс по вложенному объекту:
sql

```
CREATE TYPE AddressType AS OBJECT (
  street VARCHAR2(100),
  city   VARCHAR2(50)
);

CREATE TYPE PersonType AS OBJECT (
  id      NUMBER,
  name    VARCHAR2(100),
  address AddressType
);

CREATE TABLE persons OF PersonType;

-- Индекс на вложенный атрибут
CREATE INDEX idx_city ON persons(address.city);
```

---

2. Функциональные индексы (на метод)
sql

```
CREATE INDEX idx_get_name ON persons(p.get_name());
```

Позволяют индексировать результат вызова метода объекта.

---

3. Иерархия объектов и типы
Oracle поддерживает **наследование объектов**, но **индексировать можно только конкретный уровень иерархии**.

---

4. Ограничения
- Не поддерживаются составные индексы на вложенные объекты без специальных настроек
- Метод должен быть **детерминированным**, если он участвует в индексе

19. Коллекции в Oracle. Виды коллекций. Методы и исключения коллекций. Множественная обработка записей с выражением FORALL.

1. Коллекции в Oracle
Коллекции в Oracle — это структуры, хранящие наборы элементов одного типа. Используются в PL/SQL для хранения и обработки множественных значений.

---

2. Виды коллекций

Тип коллекции	Описание
Associative array	Ассоциативный массив (index-by table); индекс — числовой или строковый
Nested table	Массив без фиксированного размера, может быть сохранён в БД
VARRAY	Вектор фиксированной максимальной длины

Примеры объявления:
sql

```
-- Ассоциативный массив
TYPE emp_table IS TABLE OF emp%ROWTYPE INDEX BY PLS_INTEGER;

-- Nested table
TYPE name_table IS TABLE OF VARCHAR2(100);

-- VARRAY
TYPE grades_array IS VARRAY(5) OF NUMBER;
```

---

3. Методы коллекций

Метод	Описание
COUNT	Количество элементов
EXTEND	Увеличить размер (для nested table/VARRAY)
DELETE	Удаление элементов
EXISTS(i)	Проверка существования i-го элемента
FIRST / LAST	Первый / последний индекс
PRIOR / NEXT	Предыдущий / следующий индекс
TRIM(n)	Удаляет n последних элементов

---

4. Исключения при работе с коллекциями

Исключение	Причина
NO_DATA_FOUND	При обращении к отсутствующему индексу
SUBSCRIPT_BEYOND_COUNT	Обращение по индексу вне диапазона
SUBSCRIPT_OUTSIDE_LIMIT	Недопустимый индекс для VARRAY

---

5. FORALL — множественная обработка

FORALL используется для **массового выполнения DML-операций**, снижая количество контекстных переключений между SQL и PL/SQL.

Пример:

sql

```
TYPE id_table IS TABLE OF employees.employee_id%TYPE;
v_ids id_table := id_table(100, 101, 102);

FORALL i IN 1..v_ids.COUNT
  DELETE FROM employees WHERE employee_id = v_ids(i);
```

20. Коллекции в Oracle. Сравнение коллекций. MULTISET.

1. Сравнение коллекций

Коллекции можно сравнивать с помощью логических операторов **SET, SUBMULTISET, =, !=** и ключевого слова **MULTISET**.

---

2. Ключевое слово MULTISET

MULTISET используется для выполнения операций над множествами (nested table и VARRAY):

Оператор	Назначение
MULTISET UNION	Объединение (с дубликатами)
MULTISET UNION DISTINCT	Объединение без дубликатов
MULTISET INTERSECT	Пересечение
MULTISET EXCEPT	Разность
IS A SUBMULTISET OF	Проверка включения

Пример:

sql

```
DECLARE
  TYPE numlist IS TABLE OF NUMBER;
  a numlist := numlist(1, 2, 3);
  b numlist := numlist(2, 3);
BEGIN
  IF b IS A SUBMULTISET OF a THEN
    DBMS_OUTPUT.PUT_LINE('b входит в a');
  END IF;
END;
```

21. Коллекции в Oracle. Множественная обработка записей с выражением BULK COLLECT.

1. BULK COLLECT

BULK COLLECT — это механизм **массовой выборки** данных из SQL-запросов в коллекции.

Позволяет **уменьшить количество переключений контекста** между SQL-движком и PL/SQL, что значительно повышает производительность.

---

2. Пример использования BULK COLLECT

sql

```
DECLARE
```

```
  TYPE name_table IS TABLE OF employees.last_name%TYPE;
  v_names name_table;
BEGIN
  SELECT last_name
  BULK COLLECT INTO v_names
  FROM employees
  WHERE department_id = 50;
END;
```

Можно использовать и с курсорами:

```
sql

OPEN c;
FETCH c BULK COLLECT INTO v_collection;
CLOSE c;
```

---

3. BULK COLLECT с LIMIT

Для обработки больших объёмов данных можно использовать LIMIT:

```
sql

LOOP
  FETCH c BULK COLLECT INTO v_data LIMIT 100;
  EXIT WHEN v_data.COUNT = 0;
  -- Обработка данных
END LOOP;
```

22. Динамический SQL. Использование динамического SQL в SQL Server и в Oracle.

1. Общая суть динамического SQL

Динамический SQL — это SQL-код, сформированный и выполняемый **во время исполнения**.

Используется при:

- Формировании запросов со структурами, неизвестными на момент компиляции
- Выполнении запросов с переменным числом параметров
- Администрировании и генерации мета-запросов

---

2. В Oracle: EXECUTE IMMEDIATE

Синтаксис:

```
sql

EXECUTE IMMEDIATE 'SQL-запрос';
```

С переменными:

```
sql

DECLARE
  v_sql VARCHAR2(1000);
  v_result VARCHAR2(100);
BEGIN
  v_sql := 'SELECT last_name FROM employees WHERE employee_id = :id';
  EXECUTE IMMEDIATE v_sql INTO v_result USING 100;
END;
```

С DDL и DML:

```
sql

EXECUTE IMMEDIATE 'CREATE TABLE temp_table (id NUMBER)';
```

---

3. В SQL Server: EXEC и sp_executesql
EXEC:
sql

```
DECLARE @sql NVARCHAR(MAX)
SET @sql = 'SELECT * FROM Employees'
EXEC(@sql)
```

sp_executesql (с параметрами):
sql

```
DECLARE @sql NVARCHAR(MAX)
DECLARE @id INT = 100
SET @sql = 'SELECT * FROM Employees WHERE EmployeeID = @emp_id'
EXEC sp_executesql @sql, N'@emp_id INT', @id
```

23. Программные конструкции PL/SQL. Табличные функции.

1. Табличные функции в PL/SQL
Табличная функция — это функция, возвращающая **множество строк** в виде **таблицы** (тип коллекции, совместимый с SQL-запросами). Используется как источник строк в SQL, как если бы это была обычная таблица.

---

2. Определение табличной функции
Чтобы создать табличную функцию, необходимо:

1. Объявить тип строки (ROWTYPE или OBJECT).
2. Объявить тип таблицы (collection).
3. Создать функцию, возвращающую этот тип таблицы.

Пример:
sql

```
-- Тип строки
CREATE OR REPLACE TYPE emp_rec AS OBJECT (
  empno NUMBER,
  ename VARCHAR2(100)
);

-- Тип таблицы
CREATE OR REPLACE TYPE emp_tab AS TABLE OF emp_rec;

-- Функция
CREATE OR REPLACE FUNCTION get_emps (dept_id NUMBER)
  RETURN emp_tab PIPELINED IS
BEGIN
  FOR r IN (SELECT empno, ename FROM emp WHERE deptno = dept_id) LOOP
    PIPE ROW(emp_rec(r.empno, r.ename));
  END LOOP;
  RETURN;
END;
```

Использование в SQL:
sql

```
SELECT * FROM TABLE(get_emps(10));
```

---

3. Применение

- Создание представлений на основе логики PL/SQL

- Встраивание бизнес-логики в источники данных
- Работа с внешними источниками или преобразованиями данных

24. Программные конструкции PL/SQL. Конвейерные функции.

1. Что такое конвейерные функции (pipelined functions)?
Это особый вид табличных функций, возвращающих **результаты по одной строке**, сразу передавая их в SQL-движок — **без ожидания завершения выполнения функции**.
Позволяют обрабатывать большие объёмы данных **в потоковом режиме** с высокой производительностью.

---

2. Структура конвейерной функции
sql

```
CREATE OR REPLACE FUNCTION func_name(param ...)
  RETURN collection_type PIPELINED IS
BEGIN
  -- перебор и возврат по одному элементу
  FOR ... LOOP
    PIPE ROW(...);
  END LOOP;
  RETURN;
END;
```

---

3. Преимущества

Преимущество	**Объяснение**
Производительность	Строки передаются немедленно в SQL
Поддержка параллельных запросов	Используется Oracle Parallel Execution
Поддержка внешних данных	Можно вызывать из внешних таблиц (external tables)

---

4. Параллельные конвейерные функции
Могут быть реализованы с использованием интерфейса INTERFACE, PARTITION BY, PARALLEL_ENABLE для масштабируемой обработки.
sql

```
FUNCTION get_data(...)
  RETURN tab_type
  PIPELINED PARALLEL_ENABLE PARTITION BY HASH(...)
```

25. Программные конструкции PL/SQL. Пакеты. Пакетные переменные. Пакетные курсоры. Пакетные исключения.

1. Пакеты в PL/SQL
Пакет — это логическая единица программного кода, содержащая:
- Объявления типов, процедур, функций, курсоров, переменных (в спецификации)
- Реализацию (в теле пакета)

---

2. Структура пакета
sql

```
-- Спецификация
CREATE OR REPLACE PACKAGE emp_pkg IS
  PROCEDURE add_emp(...);
```

```
  FUNCTION get_count RETURN NUMBER;
  emp_counter NUMBER := 0;  -- Пакетная переменная
END;

-- Тело пакета
CREATE OR REPLACE PACKAGE BODY emp_pkg IS
  PROCEDURE add_emp(...) IS
  BEGIN
    emp_counter := emp_counter + 1;
  END;

  FUNCTION get_count RETURN NUMBER IS
  BEGIN
    RETURN emp_counter;
  END;
END;
```

---

3. Пакетные переменные

- **Сохраняют состояние между вызовами в рамках сессии**
- Используются для хранения счетчиков, кэшей, параметров

---

4. Пакетные курсоры

Можно объявлять как **явные курсоры**:

sql

```
PACKAGE emp_pkg IS
  CURSOR emp_cur IS SELECT * FROM emp;
END;
```

---

5. Пакетные исключения

Пользовательские исключения можно определить в пакете:

sql

```
PACKAGE emp_pkg IS
  e_invalid_salary EXCEPTION;
END;
```

---

6. Преимущества пакетов

Преимущество	**Объяснение**
Инкапсуляция	Сокрытие реализации
Повышение производительности	Загружается и кэшируется как единое целое
Повторное использование	Код можно использовать в разных процедурах

26. Технологии высокой доступности.

1. Определение

Высокая доступность (High Availability, HA) — это способность системы оставаться доступной даже в условиях сбоев аппаратуры, ПО, или инфраструктуры.

---

2. Цели высокой доступности

- **Минимизация простоев**
- **Обеспечение непрерывного доступа к данным**
- **Повышение отказоустойчивости и надежности**

---

3. Ключевые технологии высокой доступности

Технология	Назначение
Oracle Data Guard	Репликация базы данных в режиме реального времени
Oracle RAC (Real Application Clusters)	Несколько серверов используют одну базу данных
Always On Availability Groups (SQL Server)	Репликация и автоматическое переключение баз данных
Failover Clustering	Переключение на резервный узел при отказе
Replication (Oracle/SQL Server)	Репликация данных между экземплярами
Backup and Restore Strategies	Восстановление данных из резервных копий

---

4. Параметры оценки HA

Параметр	Описание
RTO (Recovery Time Objective)	Максимально допустимое время восстановления
RPO (Recovery Point Objective)	Максимально допустимая потеря данных
Uptime (%)	Время доступности в процентах, например: 99.999% (5 девяток)

---

5. Архитектурные подходы

- **Active-Active**: все узлы обрабатывают нагрузку
- **Active-Passive**: один узел активен, второй — резервный
- **Geo-Redundancy**: распределённые географически кластеры

27. Ретроспективные запросы. Настройка ретроспективных запросов.

Ретроспективные запросы (Flashback Queries) позволяют выполнять SQL-запросы к данным **в прошлом состоянии**, основываясь на сохранённых изменениях в UNDO-сегментах или журналах транзакций. Эта технология реализована в **Oracle Database**.

Виды ретроспективного доступа в Oracle

Тип	Описание
Flashback Query	Просмотр состояния таблицы на заданный момент времени
Flashback Table	Возврат таблицы к состоянию в прошлом
Flashback Version Query	Просмотр всех версий строки по SCN или времени
Flashback Transaction Query	Анализ и откат транзакций (DBA_FLASHBACK_TXN_REPORT)

Пример: Flashback Query
sql

```
SELECT * FROM employees AS OF TIMESTAMP (SYSTIMESTAMP - INTERVAL '5' MINUTE);
```

Требования

- Необходима настройка **UNDO_RETENTION** (время хранения информации в undo-сегментах)
- Таблицы не должны быть за пределами UNDO (например, с Nologging)

sql

```
ALTER SYSTEM SET UNDO_RETENTION = 900;
```

- Необходимы соответствующие **права (SELECT FLASHBACK ON object)**

SCN и TIMESTAMP

- SCN (System Change Number) — уникальный номер изменения данных
- TIMESTAMP — указание времени в человекочитаемом виде

sql

```
SELECT CURRENT_SCN FROM V$DATABASE;
```

28. Задания. Системные пакеты DBMS_JOB и DBMS_SCHEDULER в Oracle.

1. DBMS_JOB

Старый способ выполнения **периодических или отложенных заданий**.

Пример создания задания:

sql

```
DECLARE
  job_id NUMBER;
BEGIN
  DBMS_JOB.SUBMIT(job_id, 'my_procedure;', SYSDATE, 'SYSDATE + 1/24');
  COMMIT;
END;
```

- Задание запускается каждый час (1/24 суток)
- Запускается в фоновом процессе

---

2. DBMS_SCHEDULER

Современный механизм с расширенными возможностями.

Пример задания с DBMS_SCHEDULER:

sql

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB (
    job_name        => 'hourly_job',
    job_type        => 'PLSQL_BLOCK',
    job_action      => 'BEGIN my_procedure; END;',
    start_date      => SYSTIMESTAMP,
    repeat_interval => 'FREQ=HOURLY',
    enabled         => TRUE
  );
END;
```

Возможности DBMS_SCHEDULER:

Возможность	**Пример**
Расписание по календарю	FREQ=DAILY; BYHOUR=12; BYMINUTE=0
Работа с внешними программами	Тип EXECUTABLE
Отчеты о выполнении	Таблицы ALL_SCHEDULER_JOB_RUN_DETAILS
Привязка к ресурсам	Через job_class, windows

29. Резервное копирование и восстановление. Применение резервного копирования в SQL Server.

Типы резервного копирования в SQL Server

Тип	Описание
Полное (Full)	Полная копия всей базы данных
Дифференциальное	Только изменения после последнего Full
Журнал транзакций (Log)	Все транзакции с последнего Log backup
Copy-Only	Не влияет на цепочку бэкапов

Примеры
Полный backup:
sql

```
BACKUP DATABASE MyDB TO DISK = 'C:\backup\MyDB.bak';
```

Дифференциальный:
sql

```
BACKUP DATABASE MyDB TO DISK = 'C:\backup\MyDB_diff.bak' WITH DIFFERENTIAL;
```

Журнал транзакций:
sql

```
BACKUP LOG MyDB TO DISK = 'C:\backup\MyDB_log.trn';
```

Восстановление
Восстановление полного бэкапа:
sql

```
RESTORE DATABASE MyDB FROM DISK = 'C:\backup\MyDB.bak' WITH NORECOVERY;
```

Восстановление журнала:
sql

```
RESTORE LOG MyDB FROM DISK = 'C:\backup\MyDB_log.trn' WITH RECOVERY;
```

Планирование
В SQL Server можно использовать **SQL Server Agent Jobs** для регулярного выполнения backup’ов.

30. Резервное копирование и восстановление. Применение резервного копирования в Oracle.

1. Утилиты и подходы

Метод	Описание
RMAN (Recovery Manager)	Стандартный инструмент резервного копирования
User-Managed Backup	Использование SQL и OS-утилит (cp, tar)
Datapump Export/Import	Логическая копия объектов (expdp/impdp)
Flashback технологии	Возврат на прошлое состояние без восстановления

---

2. Примеры RMAN
Подключение:
bash

rman target /
Полное резервное копирование:
rman

BACKUP DATABASE PLUS ARCHIVELOG;
Дифференциальное:
rman

BACKUP INCREMENTAL LEVEL 1 DATABASE;
Копия управляющего файла:
rman

BACKUP CURRENT CONTROLFILE;

---

3. Восстановление с RMAN
rman

STARTUP MOUNT;
RESTORE DATABASE;
RECOVER DATABASE;
ALTER DATABASE OPEN;

---

4. Каталоги резервных копий

- Используется **Recovery Catalog** (внешняя база) или **Автоматический контроль метаданных (control file)**

---

5. Flash Recovery Area (FRA)
Специальная директория, где Oracle автоматически сохраняет backup, журналы, flashback-объекты.
sql

SHOW PARAMETER db_recovery_file_dest;

---

6. Автоматизация

- Планирование через DBMS_SCHEDULER или CRON
- RMAN scripts + shell scripts

31. Репликация. Участники репликации. Настройка репликации.

Определение
Репликация — это процесс копирования и распределения данных и объектов базы данных из одной базы данных в другую с последующей синхронизацией данных для обеспечения согласованности.

Основные участники репликации (на примере SQL Server):

Участник	Назначение
Издатель (Publisher)	Источник данных, в котором создаются публикации
Публикация (Publication)	Совокупность объектов (таблиц, представлений и т.п.) для репликации
Подписчик (Subscriber)	Получатель данных из публикации
Дистрибьютор (Distributor)	Сервер, отвечающий за доставку данных от издателя подписчикам

Настройка репликации (на примере SQL Server):

1. **Назначить Distributor**:
   - Локальный или удалённый сервер
   - Создание дистрибутивной БД
2. **Создать Publication**:
   - Определение объектов для репликации
   - Настройка фильтрации данных
3. **Создать Subscription**:
   - Указать подписчика и тип подписки (push/pull)
4. **Запустить агенты**:
   - Snapshot Agent, Log Reader Agent, Distribution Agent

32. Репликация. Топологии репликации. Виды репликации.

Топологии репликации:

Топология	Описание
Одноуровневая (Star)	Один Publisher, несколько Subscribers
Многоуровневая (Chain)	Subscriber может быть Publisher для других Subscribers
Меш (Mesh)	Каждый узел может быть Publisher и Subscriber одновременно

Виды репликации в SQL Server:

Вид	Описание
Snapshot	Передаёт полную копию данных в определённые моменты времени
Transactional	Реплицирует изменения (INSERT/UPDATE/DELETE) почти в реальном времени
Merge	Объединяет изменения на разных узлах, поддерживает двустороннюю синхронизацию

Сравнение видов:

Параметр	Snapshot	Transactional	Merge
Частота обновлений	По расписанию	Почти мгновенно	По расписанию или вручную
Направление	Односторонняя	Односторонняя	Двунаправленная
Конфликтность	Нет	Нет	Есть, требуется разрешение
Загрузка системы	Высокая при передаче	Низкая	Средняя

33. Настройка объектов базы данных. Кэширование. Журналирование. Параллельный доступ.

Кэширование (Buffer Cache)
Цель — уменьшение количества обращений к физическим устройствам хранения, ускорение доступа к данным.
Варианты кэширования:

- **Pinning объектов** — ручное закрепление таблиц в буфере
- **Materialized Views** — кэширование результатов запросов
- **Result Cache (Oracle)** — хранение результатов запросов

sql

```
ALTER TABLE employees CACHE;
```

Журналирование (Logging)
Цель — обеспечить отказоустойчивость и возможность восстановления данных.

Типы журналирования (Oracle):

Тип	Назначение
Redo Logging	Все изменения записываются в redo log
Archive Logging	Позволяет делать Point-in-time Recovery
NoLogging	Операции не записываются в журнал (быстро, но небезопасно)

sql

```
ALTER TABLE employees NOLOGGING;
```

Параллельный доступ (Concurrency Control)
Методы обеспечения:

1. **Блокировки (Locks)**:
   - Row-level, Table-level
   - Shared/Exclusive
2. **Транзакции**:
   - BEGIN / COMMIT / ROLLBACK
   - Изоляция: READ COMMITTED, SERIALIZABLE и т.д.
3. **Механизмы в СУБД**:
   - **Oracle**: Multi-Version Concurrency Control (MVCC)
   - **SQL Server**: Snapshot Isolation, Row Versioning

Пример параллелизма (Oracle):
sql

```
SELECT /*+ PARALLEL(employees, 4) */ * FROM employees;
```

Позволяет использовать 4 параллельных потока при чтении.

34. Основы бизнес-аналитики. Хранилища данных и киоски данных. Проектирование хранилищ данных.

1. Бизнес-аналитика (Business Intelligence, BI)
Совокупность процессов, технологий и инструментов для преобразования данных в информацию и знаний для поддержки принятия решений.

---

2. Хранилище данных (Data Warehouse)
Интегрированная, предметно-ориентированная, неизменяемая и исторически накопленная база данных, используемая для аналитической обработки.
Особенности:

- Поддержка принятия решений
- Интеграция из различных источников
- Поддержка временных срезов (time variant)
- Read-only операции (OLAP)

---

3. Киоски данных (Data Marts)
Подмножества хранилищ данных, ориентированные на определённую бизнес-функцию (например, продажи, финансы).

Хранилище данных	Киоск данных (Data Mart)
Централизованное	Децентрализованное (по отделам)
Более масштабное	Меньший объём
Источник для BI-систем	Часто источник — хранилище данных

---

4. Проектирование хранилищ данных
Этапы:

1. Сбор требований
2. Выбор архитектуры (Inmon vs Kimball)
3. Создание схемы:
    - Звезда (Star schema)
    - Снежинка (Snowflake schema)
4. Моделирование фактов и измерений
5. ETL-процессы: извлечение, трансформация, загрузка

Пример: Star Schema

- **Таблица фактов**: sales
- **Измерения**: product, time, region

35. Основы бизнес-аналитики. Кубы и их архитектура. Агрегирование. Уровень агрегирования. Физическое хранение куба.

1. Куб (OLAP Cube)

Многомерная структура данных, предназначенная для аналитической обработки (OLAP). Представляет данные по измерениям и фактам.

---

2. Архитектура OLAP-куба

Компонент	Назначение
Факты	Числовые показатели (например, выручка)
Измерения	Контекст фактов (время, товар, магазин)
Агрегации	Суммы, средние, минимумы и т.п.
Иерархии	Например: Год → Квартал → Месяц

---

3. Агрегирование и уровни

Агрегирование — предварительный расчёт сводных значений, чтобы ускорить запросы.

Уровень агрегирования — степень детализации:

Уровень	Пример агрегата
Высокий	Сумма продаж по году
Средний	Сумма по кварталам
Низкий	Детали по дням

---

4. Физическое хранение кубов

- **MOLAP (Multidimensional OLAP)** — хранение в специализированных структурах
- **ROLAP (Relational OLAP)** — хранение в реляционных таблицах
- **HOLAP (Hybrid OLAP)** — комбинированный подход

36. Терминология служб SSAS. Разработка и просмотр многомерного куба.

SSAS (SQL Server Analysis Services)

Подсистема Microsoft SQL Server для анализа данных (OLAP и Data Mining).

Ключевая терминология:

Термин	Описание
Cube	Многомерная модель данных
Measure	Показатель (например, сумма продаж)
Dimension	Измерение (время, география, продукт)
Hierarchy	Иерархия внутри измерения

Термин	Описание
Calculated Measure	Вычисляемое выражение
KPIs	Ключевые показатели эффективности
Partitions	Разделение данных в кубе
Aggregation Design	Настройка предварительных агрегатов

Разработка куба (пошагово в Visual Studio / SSDT):

1. Создание проекта SSAS
2. Импорт источников данных (Data Source)
3. Создание представлений (Data Source View)
4. Построение куба:
    - Выбор фактов и измерений
    - Определение иерархий
    - Настройка агрегатов
5. Обработка (Process)
6. Развёртывание (Deploy)
7. Просмотр с помощью Excel, Power BI или SQL Server Management Studio

37. NoSQL решения. MongoDB. Коллекции. Документы. Основные операции с документами и коллекциями.

ап **1. NoSQL**

Класс СУБД, не основанный на традиционной реляционной модели. Предназначен для горизонтального масштабирования, высокой доступности и работы с semi-structured и unstructured данными.

---

2. MongoDB

Документо-ориентированная NoSQL-база данных. Работает с форматами JSON/BSON.

---

3. Основные понятия MongoDB:

Элемент	Описание
База данных	Набор коллекций
Коллекция	Аналог таблицы в реляционных СУБД

Элемент	Описание
Документ	JSON-подобный объект (ключ-значение)
_id	Уникальный идентификатор документа

Пример документа:

json

```
{
  "_id": ObjectId("..."),
  "name": "Ivan",
  "age": 30,
  "address": {
    "city": "Moscow",
    "zip": "101000"
  }
}
```

---

4. Основные операции

Создание документа:

js

```
db.users.insertOne({ name: "Anna", age: 28 });
```

Поиск:

```
js


db.users.find({ age: { $gt: 25 } });
```

Обновление:

```
js


db.users.updateOne(
  { name: "Anna" },
  { $set: { age: 29 } }
);
```

Удаление:

```
js


db.users.deleteOne({ name: "Anna" });
```

Агрегация:

```
js


db.orders.aggregate([
  { $group: { _id: "$status", total: { $sum: "$amount" } } }
]);
```

38. MongoDB. Поиск в документах. Проекции. Ограничение выборки. Сортировка.

1. Поиск в документах

В MongoDB поиск осуществляется с помощью метода .find() с указанием фильтра.

Примеры фильтров:

js

```
db.users.find({ age: { $gt: 30 } });            // Возраст больше 30
db.users.find({ name: { $regex: "^A" } });      // Имя начинается с "A"
db.users.find({ "address.city": "Moscow" });    // Вложенное поле
```

---

2. Проекции

Позволяют ограничить поля, возвращаемые в результате запроса.

Синтаксис:

js

```
db.users.find({ age: { $gt: 30 } }, { name: 1, _id: 0 });
```

- 1 — включить поле
- 0 — исключить поле

---

3. Ограничение выборки

Ограничение количества возвращаемых документов с помощью .limit():

js

db.users.find().limit(5); // Вернуть 5 первых записей

Также можно использовать .skip() для пропуска определённого количества документов:

js

db.users.find().skip(10).limit(5);

---

4. Сортировка

Метод .sort() используется для упорядочивания результатов по одному или нескольким полям:

js

db.users.find().sort({ age: -1, name: 1 });

- 1 — по возрастанию
- -1 — по убыванию

39. MongoDB. Индексирование. Виды индексов. Планы запросов. Оценка планов. Хинты.

1. Индекс в MongoDB

Индекс — это структура данных, повышающая скорость выполнения операций поиска.

По умолчанию каждое поле _id индексируется.

---

2. Виды индексов

Тип индекса	Описание
Single field	По одному полю
Compound index	По нескольким полям, важен порядок
Multikey index	По массиву значений
Text index	Для полнотекстового поиска
Hashed index	Для шардирования
Sparse index	Индексирует только документы, где поле есть
Unique index	Значения поля уникальны
Partial index	Индексирует только документы, удовлетворяющие условию

Примеры создания:

js

```
db.users.createIndex({ age: 1 });                    // Индекс по возрасту
db.users.createIndex({ name: 1, age: -1 });          // Составной индекс
db.users.createIndex({ tags: 1 });                   // Multikey
db.articles.createIndex({ content: "text" });        // Текстовый индекс
```

---

3. План запроса (Query Plan)

MongoDB использует **query planner** для выбора оптимального способа выполнения запроса. Анализ можно провести через:

js

```
db.collection.find({ ... }).explain("executionStats");
```

Ключевые метрики:

- totalKeysExamined
- totalDocsExamined
- executionTimeMillis

---

4. Хинты (hints)

Хинт — указание MongoDB использовать конкретный индекс:

js

```
db.users.find({ age: { $gt: 25 } }).hint({ age: 1 });
```

Применяется для оптимизации или отладки.

40. MongoDB. Индексирование. Пространственные индексы. Методы поиска пространственных данных с учетом индекса.

1. Пространственные данные в MongoDB

MongoDB поддерживает геоданные в формате **GeoJSON**:

json

```
{
  "type": "Point",
  "coordinates": [longitude, latitude]
}
```

---

2. Пространственные индексы

MongoDB поддерживает два типа геопространственных индексов:

Тип индекса	Назначение
2d	Плоская проекция (для старых приложений)
2dsphere	Геосферическая модель (WGS84, для GeoJSON)

Создание индекса:

js

```
db.places.createIndex({ location: "2dsphere" });
```

---

3. Методы пространственного поиска

Поиск ближайших объектов ($near)

js

```
db.places.find({
  location: {
    $near: {
      $geometry: {
        type: "Point",
        coordinates: [37.6173, 55.7558]
      },
      $maxDistance: 5000  // в метрах
    }
  }
});
```

Поиск в пределах полигона ($geoWithin)

js

```
db.places.find({
  location: {
    $geoWithin: {
      $geometry: {
        type: "Polygon",
        coordinates: [ [ [...], [...], [...], [...] ] ]
      }
    }
  }
```

});

Поиск пересечений ($geoIntersects)

js

```
db.routes.find({
  geometry: {
    $geoIntersects: {
      $geometry: {
        type: "LineString",
        coordinates: [...]
      }
    }
  }
});
```

---

4. Оптимизация и анализ

- Использовать explain() для оценки геозапросов.
- Пространственные индексы значительно ускоряют поиск по координатам.
- 2dsphere индекс необходим для запросов с GeoJSON-данными.

41. MongoDB. Индексирование. Текстовый индекс. Различные виды текстовых индексов. Индексы с весами. Стемминг. Релевантность.

1. Текстовый индекс в MongoDB

Позволяет выполнять полнотекстовый поиск по строковым полям. Индексирует слова в документах с поддержкой стемминга и игнорированием стоп-слов.

---

2. Виды текстовых индексов

- **Стандартный текстовый индекс** — создаётся по одному или нескольким полям, индексирует слова для поиска.
- **Многоязычный индекс** — поддерживает стемминг и стоп-слова для разных языков.
- **Индексы с весами (weights)** — задают приоритет важности полей для релевантности.

---

3. Создание текстового индекса

js

```
db.articles.createIndex(
  { title: "text", content: "text" },
  { weights: { title: 10, content: 5 }, default_language: "russian" }
);
```

- Здесь поле title имеет больший вес, чем content.
- default_language определяет правила стемминга и стоп-слова.

---

4. Стемминг

Механизм, приводящий слова к их базовой форме (корню):

- "программирование" → "программ"
- "программировать" → "программ"

Стемминг позволяет находить документы по разным формам слова.

---

5. Релевантность

MongoDB вычисляет релевантность документов с помощью **score**, основанного на:

- Частоте встречаемых слов
- Весе поля
- Плотности совпадений

Результаты поиска можно сортировать по убыванию score:

js

```
db.articles.find(
  { $text: { $search: "поиск" } },
  { score: { $meta: "textScore" } }
).sort({ score: { $meta: "textScore" } });
```

42. MongoDB. Фреймворк агрегации. Назначение. Пайплайн. Стадии.

1. Назначение

Фреймворк агрегации позволяет выполнять сложные операции обработки и трансформации данных внутри базы — группировки, фильтрации, вычисления, сортировки, изменения структуры.

---

2. Пайплайн агрегации

Состоит из последовательности стадий (pipeline stages), каждая принимает на вход результаты предыдущей и возвращает результат для следующей.

---

3. Основные стадии:

Стадия	Описание
$match	Фильтрация документов (аналог WHERE)
$group	Группировка и агрегатные функции (SUM, AVG)
$project	Проекция и формирование новых полей
$sort	Сортировка
$limit	Ограничение количества результатов
$skip	Пропуск заданного количества документов
$unwind	Разворачивание массивов
$lookup	Соединение с другой коллекцией (аналог JOIN)

---

4. Пример пайплайна

js

```
db.orders.aggregate([
  { $match: { status: "completed" } },
```

```
  { $group: { _id: "$customerId", totalAmount: { $sum: "$amount" } } },
  { $sort: { totalAmount: -1 } },
  { $limit: 5 }
]);
```

43. MongoDB. Аутентификация. Роли. Пользователи. Привилегии. Встроенные роли.

1. Аутентификация

MongoDB поддерживает проверку подлинности пользователей, основанную на механизмах:

- **SCRAM-SHA-1/256 (стандартные)**
- **LDAP**
- **Kerberos**
- **X.509 сертификаты**

---

2. Пользователи

Пользователь создаётся в конкретной базе данных и может иметь роли с набором привилегий.

---

3. Роли

Роли группируют наборы привилегий. Роли бывают:

- **Встроенные (built-in) — предоставляются MongoDB по умолчанию.**
- **Пользовательские — создаются администратором.**

---

4. Привилегии

Определяют права пользователя на действия с базой данных, коллекциями и командами (например, чтение, запись, администрирование).

---

5. Примеры встроенных ролей

Роль	Назначение
read	**Чтение данных во всех коллекциях базы**
readWrite	**Чтение и запись в коллекциях базы**
dbAdmin	**Администрирование базы (создание индексов, stats)**
userAdmin	**Управление пользователями и ролями**
clusterAdmin	**Управление кластером MongoDB**
root	**Все права на сервере MongoDB**

---

6. Создание пользователя с ролью

js

```
db.createUser({
  user: "analyst",
  pwd: "securePass",
```

```
  roles: [ { role: "read", db: "sales" } ]

});
```

44. MongoDB. Устройство базы данных. Файлы. Механизмы чтения и записи. Ограничения целостности. Типы данных.

1. Устройство базы данных

- **Файлы данных** — MongoDB хранит данные в файлах, которые управляются движком хранения (Storage Engine).
- Наиболее популярный движок — **WiredTiger** (по умолчанию).
- Есть также **MMAPv1** (устаревший).

---

2. Файлы и storage engine

- Файлы данных содержат документы, индексы и метаданные.
- WiredTiger использует сжатие данных и поддерживает многоверсионную конкурентность (MVCC).

---

3. Механизмы чтения и записи

- **Запись:** операция осуществляется в память, затем асинхронно сбрасывается на диск (write-ahead log).
- **Чтение:** из памяти или с диска, оптимизировано с помощью кэширования.
- Механизмы обеспечивают высокую производительность и параллелизм.

---

4. Ограничения целостности

MongoDB — NoSQL СУБД, не реализует традиционные реляционные ограничения:

- Нет внешних ключей

- Нет транзакций на уровне SQL (до версии 4.0)
- Существуют **многоуровневые транзакции** с версии 4.0
- Ограничения реализуются на уровне приложения

---

5. Типы данных

MongoDB поддерживает разнообразные типы BSON:

Тип	**Описание**
String	Текстовые данные
Integer (int32/64)	Целые числа
Double	Числа с плавающей точкой
Boolean	Логические значения
Date	Дата и время
ObjectId	Уникальный идентификатор документа
Array	Массивы
Embedded Document	Вложенные документы
Binary data	Двоичные данные

Тип	Описание
Null	Значение NULL
Regular Expression	Регулярные выражения

45. Анализ данных в Python. Специализированные библиотеки.

Основные библиотеки для анализа данных в Python:

- **NumPy**
  Основная библиотека для работы с многомерными массивами и матрицами. Позволяет выполнять высокопроизводительные математические операции, линейную алгебру, генерацию случайных чисел.
- **Pandas**
  Предназначена для обработки и анализа данных в табличном формате (DataFrame). Обеспечивает удобные инструменты для фильтрации, группировки, агрегации и обработки пропусков.
- **Matplotlib**
  Базовая библиотека визуализации данных: создание графиков, диаграмм и других визуальных представлений.
- **Seaborn**
  Расширение Matplotlib с более удобным интерфейсом и современным стилем, особенно для статистической визуализации.
- **SciPy**
  Математическая библиотека с функциями для оптимизации, интегрирования, статистики, обработки сигналов и др.
- **Scikit-learn**
  Библиотека машинного обучения: классификация, регрессия, кластеризация, снижение размерности, обработка данных.
- **Statsmodels**
  Инструменты для проведения статистического моделирования, включая регрессионный анализ, временные ряды, проверку гипотез.

46. Анализ данных в Python. Основные понятия статистики. Проверка гипотез. Статистический вывод.

Основные понятия статистики:

- **Выборка и генеральная совокупность**
  Совокупность всех данных называется генеральной совокупностью. Выборка — подмножество данных для анализа.

- **Средние показатели:**
  - Среднее арифметическое (mean)
  - Медиана (median)
  - Мода (mode)
- **Дисперсия и стандартное отклонение** — меры разброса данных.
- **Распределение данных** — частотное распределение или плотность вероятности.

Проверка статистических гипотез:

- **Нулевая гипотеза (H0)** — исходное утверждение, которое проверяется (обычно отсутствие эффекта).
- **Альтернативная гипотеза (H1)** — утверждение, противоположное нулевой гипотезе.
- **Уровень значимости (α)** — вероятность ошибочно отвергнуть H0 (часто 0.05).
- **Статистика теста** — расчетное значение, позволяющее принять или отвергнуть H0.
- **p-значение** — вероятность получить наблюдаемые данные при условии, что H0 верна.

Процесс проверки гипотез:

1. Формулировка H0 и H1.
2. Выбор статистического теста (t-тест, χ^2-тест, ANOVA и др.).
3. Вычисление статистики теста и p-значения.
4. Сравнение p-значения с уровнем α.
5. Принятие решения: отвергнуть или не отвергать H0.

В Python:

- Для статистики и проверки гипотез используется библиотека **SciPy** (scipy.stats).

47. Анализ данных в Python. Сравнение средних значений.

Основные методы сравнения средних:

- **t-тест для независимых выборок**
  Проверяет, значимо ли отличаются средние двух независимых групп.
- **t-тест для зависимых выборок (парный t-тест)**
  Применяется, если наблюдения связаны (например, до и после лечения).
- **ANOVA (дисперсионный анализ)**
  Используется для сравнения средних более чем двух групп.

Реализация t-теста в Python:

python

```
from scipy import stats

# Два независимых массива данных
group1 = [23, 20, 22, 21, 24]
group2 = [30, 29, 31, 28, 32]
```

```
# t-тест для независимых выборок

t_stat, p_value = stats.ttest_ind(group1, group2)

print(f"t-statistic: {t_stat}, p-value: {p_value}")
```

Ключевые моменты:

- Проверка на нормальность распределения (например, тест Шапиро-Уилка).
- Проверка равенства дисперсий (тест Левена).
- Выбор правильного теста (с учётом данных).

48. Анализ данных в Python. Корреляция и регрессия.

Корреляция

- **Корреляция** — мера степени взаимосвязи между двумя переменными.
- Коэффициент корреляции принимает значения от -1 до +1:
  - +1 — идеальная положительная корреляция,
  - -1 — идеальная отрицательная,
  - 0 — отсутствует линейная связь.

Виды корреляции:

- **Пирсоновская корреляция (Pearson correlation)** — измеряет линейную зависимость.
- **Спирмена (Spearman's rank)** — для ранговых данных и нелинейных связей.
- **Кендалла (Kendall tau)** — для оценки ассоциации порядковых данных.

Реализация в Python (Pandas / SciPy):
python

```
import pandas as pd
from scipy.stats import pearsonr, spearmanr

data = pd.DataFrame({'x': [1, 2, 3, 4, 5], 'y': [2, 4, 5, 4, 5]})

# Пирсоновская корреляция
corr, p_val = pearsonr(data['x'], data['y'])
print(f'Pearson correlation: {corr}, p-value: {p_val}')

# Спирмена
corr_s, p_val_s = spearmanr(data['x'], data['y'])
print(f'Spearman correlation: {corr_s}, p-value: {p_val_s}')
```

Регрессия

- **Регрессия** — статистический метод для моделирования зависимости переменной (цели) от одной или нескольких независимых переменных.
- **Линейная регрессия** — простейший вид, где зависимость считается линейной.
- В Python для регрессии используется библиотека **scikit-learn** или **statsmodels**.

Пример линейной регрессии с scikit-learn:

python

```
from sklearn.linear_model import LinearRegression
import numpy as np

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1)
y = np.array([2, 4, 5, 4, 5])

model = LinearRegression()
model.fit(X, y)

print(f'Intercept: {model.intercept_}, Coefficient: {model.coef_[0]}')
```

49. Анализ данных в Python. Применение метода Bootstrap для алгоритма деревьев решений.

Метод Bootstrap

- **Bootstrap** — статистический метод оценки распределения статистики (например, среднего) с помощью повторных случайных выборок с возвращением из исходных данных.
- Позволяет оценить устойчивость модели и доверительные интервалы.

Bootstrap в деревьях решений

- В алгоритме **Random Forest** используется bootstrap-выборка для построения каждого дерева.
- Каждое дерево обучается на случайной выборке с повторениями из исходных данных, что повышает разнообразие деревьев и улучшает обобщающую способность.

Пример bootstrap с DecisionTreeClassifier:

python

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.utils import resample
import numpy as np

X = np.array([[1], [2], [3], [4], [5]])
y = np.array([0, 0, 1, 1, 1])

# Bootstrap-выборка
```

```
X_resampled, y_resampled = resample(X, y, replace=True, n_samples=len(X),
random_state=42)


model = DecisionTreeClassifier()

model.fit(X_resampled, y_resampled)
```

Итог:

- Bootstrap помогает оценить вариативность и устойчивость модели.
- В Random Forest это стандартный механизм.

50. Анализ данных в Python. Применение методов Airflow.

Apache Airflow

- Платформа для оркестрации и автоматизации рабочих процессов (workflow) — планировщик и менеджер DAG (Directed Acyclic Graph).
- Позволяет создавать, планировать и отслеживать цепочки задач обработки данных.

Применение в анализе данных:

- Автоматизация ETL-процессов (извлечение, трансформация, загрузка данных).
- Планирование и запуск моделей машинного обучения.
- Автоматизация отчетности и аналитики.
- Интеграция с различными источниками данных и инструментами.

Основные компоненты:

- **DAG** — описывает зависимости между задачами.
- **Операторы (Operators)** — отдельные задачи (PythonOperator, BashOperator, SQLOperator и др.).
- **Scheduler** — планировщик задач.
- **Executor** — выполняет задачи.

Пример простого DAG в Airflow:
python

```
from airflow import DAG
from airflow.operators.python import PythonOperator
from datetime import datetime

def print_hello():
   print("Hello Airflow!")

dag = DAG('hello_airflow', start_date=datetime(2023, 1, 1), schedule_interval='@daily')

task = PythonOperator(task_id='hello_task', python_callable=print_hello, dag=dag)
```

Итог:

- Airflow — мощный инструмент для организации повторяемых процессов анализа данных.
- Позволяет строить масштабируемые пайплайны с мониторингом и уведомлениями.