Список тем и вопросов

- Что такое данные и информация?
- Котреж и атрибут
- Дублирование и избыточность
- Сущность
- Что такое модели данных?
- Что такое базы данных?
- Что такое домен, отношение, схема отношения, степень отношения, кардинальность отношения?
- Какие есть свойства отношений?
- Какие бывают базы данных?
- Уровни абстракции в БД (Уровни архитектуры ANSI/SPARC)
- Транзакции в БД и ACID
- Типы данных в БД
- Типы атрибутов
- Супер, потенциальный, первичный и внешний ключ и связи в БД
- Представления
- Реляционная алгебра
- Проектирование БД
- Порядок команд в SQL
- Индексы в реляционных БД
- Как работает JOIN под капотом
- Аномалии в БД
- Нормализация данных для СУБД (первая нормальная форма, вторая нормальная форма, третья нормальная форма, четвертая нормальная форма)

Теория, ответы на вопросы

Что такое данные и информация?

Данные - это отдельные факты или значения без какого-либо контекста, которые сами по себе не имеют смысла (Собственная интерпретация + интернет)
Данные - это поддающиеся многократной интерпретации представления информации в формализованном виде, пригодном для передачи, интерпретации и обработки

```
Примеры данных:
Числа (10), слова ("рубежка") или даты (2024-11-01)
```

Информация - это данные, которые были обработаны и обрели смысл в каком-то определенном контексте

```
10 – количество баллов, которые получат студенты прочитавшие эту методичку
"рубежка" – это мероприятие (рубежный контроль), которое скоро пройдет на
лекции у 3 потока
2024-11-01 – это дата рубежного контроля
```

Кортеж и атрибут

Кортеж - одна строка в таблице
Атрибут - один столбец в таблице. Заголовок столбца определяет его имя

Дублирование и избыточность:

Дублирование — это прямое повторение одних и тех же данных в разных местах (например, хранение имени студента в нескольких таблицах), что может привести к несоответствиям при обновлении

Избыточность — более широкое понятие, которое включает дублирование, но также охватывает излишние данные, которые не обязательны для уникальной идентификации объекта

Сущность:

Сущность — это множество экземпляров (реальных или абстрактных) однотипных объектов предметной области
Сущность называется *сильной*, если её экземпляры могут существовать независимо. *Слабые* сущности могут существовать только при наличии одного или нескольких экземпляров сильной сущности

Что такое модели данных?

Модели данных - это способы организации и представления данных. Она описывает, как данные связаны между собой и как их можно использовать

Основные типы моделей данных:

1. **Реляционная** - данные организованны в таблицы с рядами и колонками (кортежи и атрибуты); связи между данными создаются с помощью общих полей

- *Плюсы:*

    - *Простота и стандартизация*
    - *Целостность и согласованность данных*
    - *Удобство работы с SQL*
- *Минусы:*
    - *Ограничения в гибкости данных*
    - *Проблемы масштабирования*
    - *Избыточность данных*
    - *Нагрузка на производительность*

2. **Нереляционная** - данные хранятся в гибких структурах (документы, граф, пары ключ-значение) и не требуют строгих схем. Подходит для работы с неструктурированными и динамическим данными.

```
Примеры таких типов моделей данных:
– Иерархическая – данные в виде дерева, где каждый элемент имеет одного родителя и может иметь потомков
        – **Плюсы**:
            – Удобна для восприятия человеком, так как соответствует реальным иерархиям
            – Простая структура, легко понимать и использовать для хранения иерархических данных
            – Быстро при транзакциях
        – **Минусы**:
            – Дублирование данных, что приводит к избыточности
            – Сложности в поддержании целостности данных и обновлении при изменении иерархии
            – Нельзя сделать связь "многие-ко-многим"
– Сетевая – данные связаны как сеть, где один элемент может иметь множество связей с другими элементами
        – **Плюсы сетевой модели данных:**
            – Гибкость связей – поддерживает многие-ко-многим, что позволяет создавать сложные связи между данными
            – Эффективность доступа – позволяет быстро находить и связывать данные, благодаря оптимизации связей и указателей
            – Экономия на дублировании данных – связи создаются напрямую, что снижает избыточность данных по сравнению с иерархической моделью
        – **Минусы сетевой модели данных:**
            – Сложность управления – управление и администрирование такой модели требуют сложной логики и специализированного программного обеспечения
            – Трудность обновлений – изменения структуры могут потребовать значительных изменений в приложениях
            – Ограниченная поддержка – разработка и поддержка сетевой модели были частично вытеснены реляционной моделью, что ограничивает её применение
– Объектно-ориентированная – данные хранятся в виде объектов, как и в программировании
```

```
        - **Плюсы**:
            - Совместимость с ООП
            - Поддержка сложных данных
            - Гибкость структуры данных
            - Повышенная инкапсуляция
        - **Минусы**:
            - Ограниченная поддержка запросов
            - Сложность управления данными
            - Низкая производительность
- Документо-ориентированная - хранит данные в виде документов. Каждый документ может содержать структурированную или неструктурированную инфомрмацию, а также структура документа может гибкой
        - **Плюсы**:
            - Гибкость структуры данных
            - Хорошо подходит для неструктурированных данных
            - Высокая производительность при работе с отдельными документами
            - Масштабируемость

        - **Минусы**:
            - Ограниченные возможности сложных запросов
            - Избыточность данных
            - Меньшая целостность данных
            - Трудности с транзакциями
```

Что такое базы данных?

База данных - это организованное хранилище данных, управляемое системой управления базами данных (*СУБД*), для удобного хранения, поиска и обработки информации (Собственная интерпретация + интернет)
База данных — это набор постоянно хранимых данных, используемых прикладными системами предприятия (Маятин)

Что такое домен, отношение, схема отношения, степень отношения, кардинальность отношения?

Домен - это множество допустимых значений атрибута
Отношение - это множество упорядоченных кортежей, где каждое значение берется из соответствующего домена
Схема отношения — это строка заголовков
Степень отношения — это количество его атрибутов
Кардинальность отношения — это количество его кортежей

Какие есть свойства отношений?

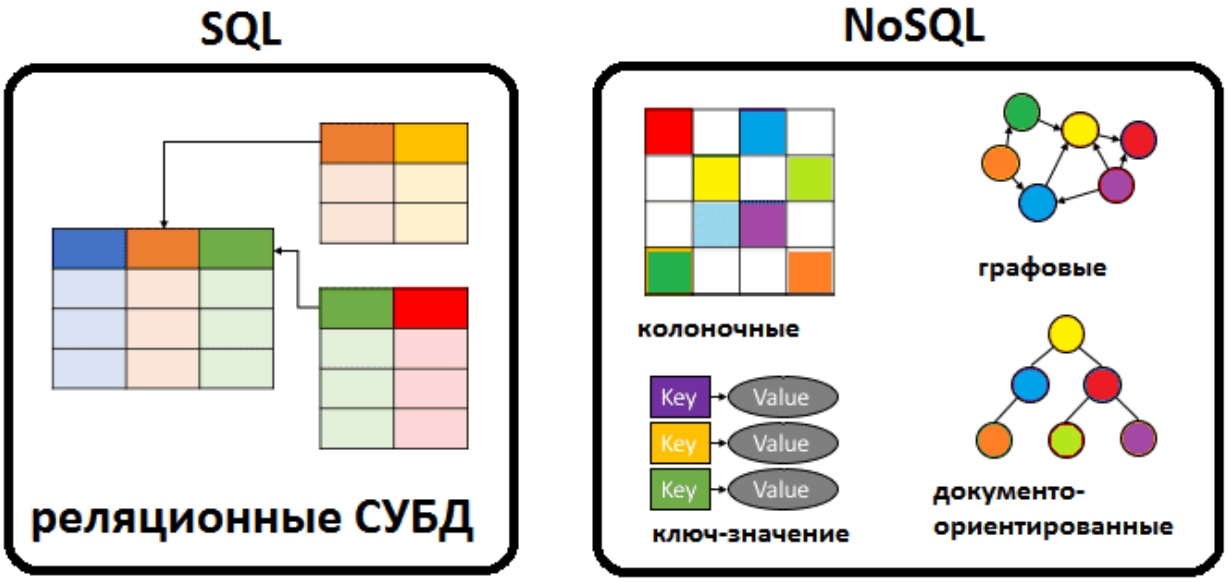
- ***Уникальность имени таблицы*** - каждая таблица имеет уникальное имя в БД, что позволяет однозначно ее идентифицировать

- ***Уникальность кортежей и атрибутов***- каждый кортеж и атрибут в таблице уникален, что исключает возможность дублирования данных
- ***Неупорядоченность кортежей и атрибутов*** - порядок атрибутов и кортежей не имеет значения, так как данные идентифицируются по именам атрибутов и извлекаются независимо от порядка
- ***Атомарность значений*** - каждая ячейка в таблице содержит одно неделимое значение
- ***Однородность доменов*** - значения атрибутов берутся из одного и того же домена (типа данных), чтобы сохранить единообразие
- ***Целостность данных*** - значения атрибутов должны соответствовать определенным правилам и ограничениям, например, `NOT NULL` для обязательных полей или ограничение уникальности для первичного ключа

Какие бывают базы данных?

Базы данных бывают следующих основных типов:

- **Реляционные (SQL)** - данные хранятся в таблицах с чёткими связями между ними. Пример: PostgreSQL, MySQL
- **Нереляционные (NoSQL)** - данные хранятся в гибких структурах (документы, ключ-значение, графы). Пример: Redis, MongoDB, ClickHouse
  Предпосылки к появлению NoSQL решений:
- **Увеличение объёма хранимых данных** — необходимость масштабируемых решений.
- **Слабо структурированные данные** — гибкость для хранения данных без строгой схемы.
- **Высокая взаимосвязность данных** — поддержка графов и сложных связей.
- **Архитектура информационных систем** — распределённые и облачные системы с горизонтальным масштабированием



Уровни абстракции в БД (Уровни архитектуры ANSI/SPARC)

Уровни абстракции в базах данных - это концепция, которая делит данные на уровни для упрощения управления и повышения гибкости, основные уровни:

1. **Внутренний уровень** - отвечает за физическое хранение данных на диске и оптимизацию работы с ними. Скрыт от пользователя, влияет на производительность и безопасность
2. **Концептуальный уровень** - логическая структура данных (сущности, атрибуты, связи), с которой работают разработчики при проектировании базы
3. **Внешний уровень** - представление данных для пользователей, показывающее нужную информацию и скрывающее технические детали хранения

Транзакции в БД и ACID

Транзакции - это последовательность, набор операций, которые выполняются как единое целое. Если хотя бы одна операция из набора не выполнится, все изменения отменяются. Это гарантирует целостность данных. Результатом каждой транзакции может быть либо `commit`, либо `rollback` (Собственная интерпретация + интернет)
Транзакция — это последовательность действий с базой данных, в которой либо все действия выполняются успешно, либо не выполняется ни одно из них. Результатом каждой транзакции может быть либо `commit`, либо `rollback` (Маятин)

```
Пример транзакции:
1) Начало транзакции
2) Изменение данных (например, добавление записи о новом заказе)
3) Если все операции прошли успешно – подтверждение (commit)
4) Если произошла ошибка – откат транзакции (rollback)
```

ACID - это набор свойств, которые должны обеспечиваться транзакциями в базе данных для гарантии надежности и целостности данных:

1. **Атомарность (Atomicity)** - транзакция выполняется полностью или не выполняется вовсе. Если одна часть операции не удается, все изменения откатываются
2. **Согласованность (Consistency)** - после завершения транзакции данные должны оставаться в согласованном состоянии, соблюдая все правила и ограничения базы данных
3. **Изолированность (Isolation)** - параллельные транзакции не должны влиять друг на друга. Результат одной транзакции виден другим только после её завершения
4. **Долговечность (Durability)** - после завершения транзакции все изменения сохраняются и остаются неизменными даже при сбоях системы

4 проблемы конкурирующих транзакций:

1. **Потерянное обновление**: Несколько транзакций изменяют одно и то же значение, но сохраняется лишь последнее.
    - *Пример*: Человек переводит деньги, и одновременно другой переводит ему. Итоговая сумма может быть неверной
2. **Грязное чтение**: Транзакция читает данные, изменённые, но не зафиксированные другой транзакцией, которая затем может отмениться.
    - *Пример*: Начисленные проценты будут ошибочны, если перевод отменяется
3. **Неповторяемое чтение**: Данные изменяются другой транзакцией после первого чтения.
    - *Пример*: Скидка пересчитывается неверно, если другая транзакция меняет цену товара
4. **Фантомное чтение**: Новые записи добавлены другой транзакцией между чтениями.
    - *Пример*: Стоимость доставки рассчитывается неверно, если добавляются новые покупки

4 уровня изоляции, они решают проблему конкурирующих транзакций:

1. **Незавершённое чтение**: Изменять данные должна только одна транзакция
2. **Завершённое чтение**: Если транзакция собирается менять данные, то она блокирует их монопольно
3. **Воспроизводимое чтение**: Если транзакция планирует читать данные, то она блокирует их монопольно
4. **Сериализуемость**: Если транзакция обращается к данным, то никакая другая транзакция не может добавлять или изменять строки, которые могут быть считаны при выполнении этой транзакции

Типы данных в БД

Так как мы используем *Postgress*, то поговорим о типах данных в этой СУБД:

1. **Числовые типы:**
    - **INTEGER, BIGINT, SMALLINT** - целые числа разной длины
    - **SERIAL, BIGSERIAL** - автоинкрементные целые числа
    - **FLOAT, REAL, DOUBLE PRECISION** - числа с плавающей точкой, небольшая точность
    - **NUMERIC, DECIMAL** - числа с плавающей точкой с фиксированной точностью
2. **Строковые типы:**
    - **CHAR(n), VARCHAR(n)** - строки фиксированной или переменной длины, где `n` задает максимальное количество символов

- **TEXT** - строка неограниченной длины

3. **Типы данных для даты и времени:**
   - **DATE** - только дата (год, месяц, день)
   - **TIME** - только время (часы, минуты, секунды)
   - **TIMESTAMP** - дата и время вместе, без часового пояса
   - **TIMESTAMPTZ** - дата и время с учётом часового пояса
   - **INTERVAL** - промежуток времени
4. **Логический тип:**
   - **BOOLEAN** - логический тип данных, который хранит значение True, False или Null
5. **UUID:**
   - **UUID** - универсальный уникальный идентификатор, используется для создания уникальных значений, как наше номер ИСУ
6. **Типы для работы с массивами:**
   - **ARRAY** - массивы значений любого типа
   (Понятно, что это не все типы данных, а самые основные)

Типы атрибутов

- ***Простые атрибуты*** - не делятся на подчасти
  Пример: ФИО - не делится, так как одна строка цельная
- ***Составные атрибуты*** - состоят из нескольких атрибутов
  Пример: Адрес - можно разделить на Улицу, Номер дома, Город и т.д.
- ***Идентифицирующие атрибуты (Обязательные атрибуты)*** - однозначно идентифицируют сущность
  Пример: Номер ИСУ - уникален для каждого студента
- ***Необязательные атрибуты*** — могут не иметь значения для некоторых записей и могут быть NULL
  Пример: Отчество - может просто не быть у некоторых студентов
- ***Однозначные атрибуты*** - хранят только одно значение
  Пример: Дата рождения - у каждого студента только одна дата рождения
- ***Многозначные атрибуты*** - могут содержать несколько значений
  Пример: Телефонные номера - у студента может быть несколько телефонных

Супер, потенциальный, первичный и внешний ключ и связи в БД

Суперключ — это атрибут или комбинация атрибутов, которые однозначно идентифицируют каждую запись (кортеж) в таблице

```
Пример:
Пусть у нас есть таблица "students" с атрибутами: ISU, full_name, email.
```

```
– Суперключом может быть комбинация (ISU, email), так как это уникально
идентифицирует кортеж в таблице.
```

Потенциальный ключ — это суперключ, который не содержит подмножества, также являющегося суперключом этого отношения

```
Пример:
В той же таблице "students" атрибут ISU является потенциальным ключом, так
как он сам по себе уникально определяет каждого студента и не содержит
избыточных атрибутов.
```

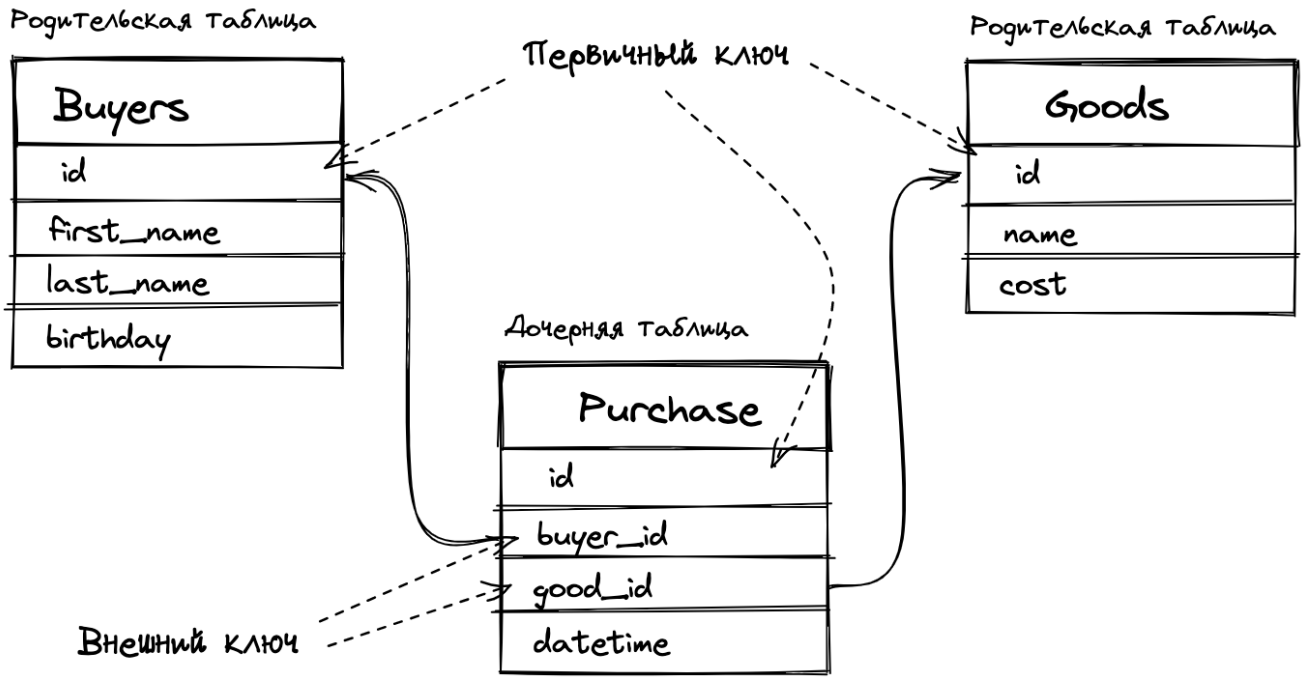
Первичный ключ (Primary Key) - это уникальный идентификатор каждой записи в таблице, обеспечивающий однозначное обращение к данным (первичный ключ - это потенциальный ключ)
Типы первичных ключей:

- Естественный ключ - уникальный атрибут, существующий сам по себе (например, ИСУ)
- Суррогатный ключ - искусственный идентификатор (например, автоинкрементный ID или UUID)

```
Пример:
Пусть у нас есть база данных с таблицей "students":
– Естественный ключ – "ISU" (номер ИСУ), который уникально идентифицирует
каждого студента
– Суррогатный ключ – "student_id" с автоинкрементом или UUID, если
уникального атрибута нет
```

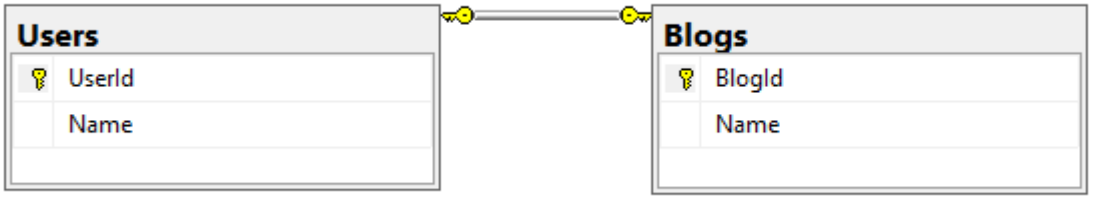
Внешний ключ (Foreign Key) - это поле, связанное с первичным ключом другой таблицы, обеспечивающее связь между таблицами и поддержание целостности данных

```
Пример:
Пусть у нас есть база данных с таблицами "students" и "gradebook":
– В таблице "students" есть первичный ключ "ISU", уникально
идентифицирующий каждого студента.
– В таблице "gradebook" есть поле "ISU", которое является внешним ключом,
ссылающимся на "ISU" в таблице "students"
```

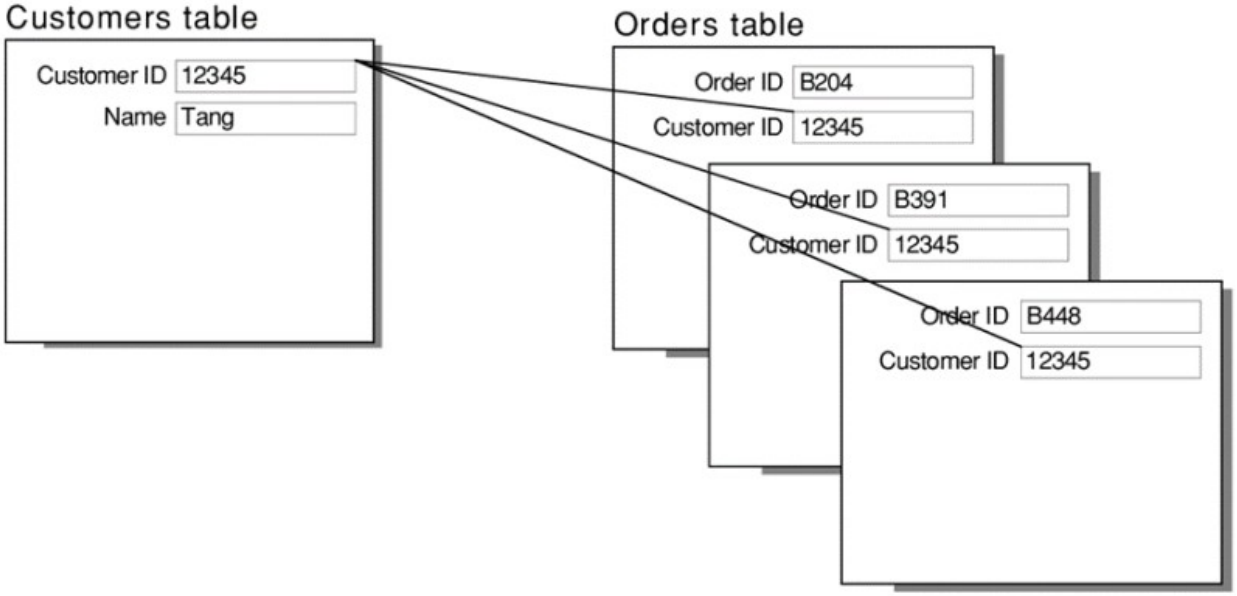


Связи в БД - это отношения между таблицами, определяющие, как данные в одной таблице соотносятся с данными в другой
Типы связей:

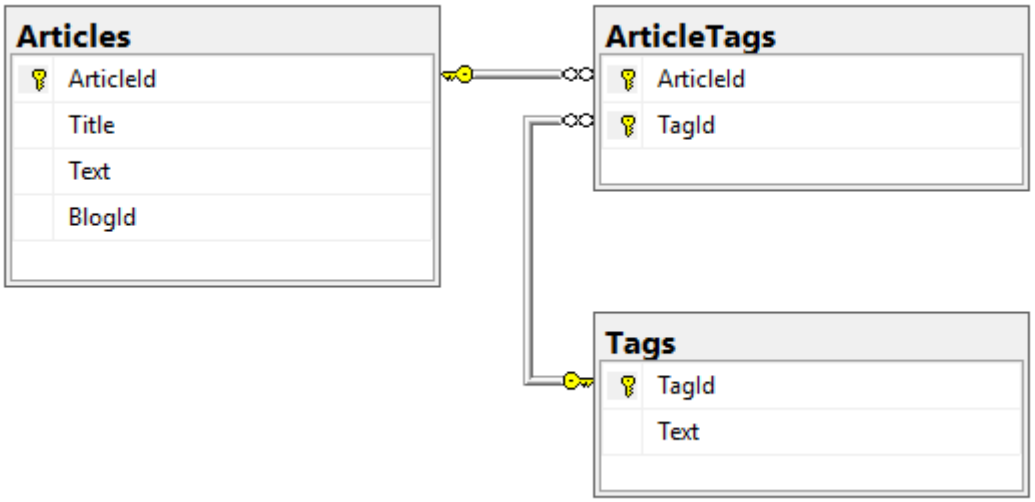
- **Один-к-одному** - одна запись соответствует одной записи



- **Один-ко-многим** - одна запись связана с несколькими записями



- **Многие-ко-многим** - записи из одной таблицы могут быть связаны с несколькими записями в другой таблице через промежуточную таблицу



Представления:

Представление — это динамически сформированный результат одной или нескольких реляционных операций, сохраненный в виде нового отношения. Они делятся на два основных типа:

1. **Материализованные** - хранятся в памяти и обновляются раз в некоторое время
2. **Представления замены** - в памяти хранится только запрос, который позволяет получить это представление. При получении запроса извне этот сохраненый запрос подставляется во внешний запрос и выполняется

Представления бывают обновляемыми

Плюсы представлений:

- Независимость от данных
- Повышение защищенности данных
- Снижение сложности работы с данными
  Минусы представлений:
- Ограниченность в возможности обновлений
- Структрурные ограничения
- Снижение производительности

Реляционная алгебра:

1. **Проекция** Па1 ...an (R)
   Результатом проекции является новое отношение, содержащее вертикальное подмножество исходного отношения, создаваемое посредством извлечения указанных атрибутов и исключения из результата атрибутов-дубликатов.
2. **Выборка** σ(предикат)(R)
   Результатом выборки является отношение, которое содержит только те кортежи

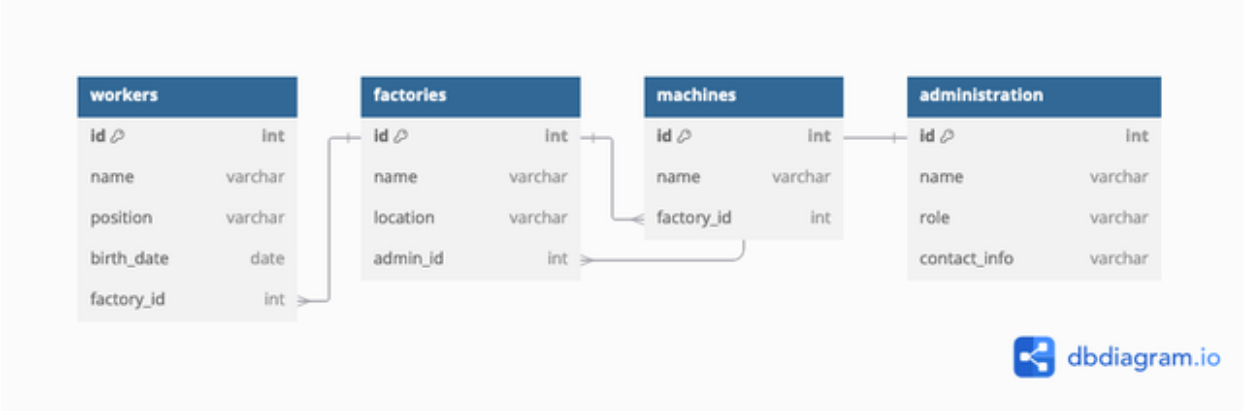
из исходного отношения, которые удовлетворяют заданному условию (предикату).

3. **Объединение** R ∪ S
   Объединение двух отношений R и S определяет новое отношение, которое включает все кортежи, содержащиеся только в R, все кортежи, содержащиеся только в S, и кортежи, содержащиеся и в R, и в S, исключая дубликаты. Объединение возможно только для совместных отношений, имеющих одинаковое количество атрибутов и одинаковый домен.
4. **Разность** R - S
   Разность состоит из кортежей, которые есть в R, но отсутствуют в S. Разность двух отношений определена только если они совместны по объединению.
5. **Пересечение** R ∩ S
   Операция пересечения определяет отношение, содержащее кортежи, находящиеся как в R, так и в S. Пересечение возможно только для совместных отношений.
6. **Декартово произведение** R × S
   Декартово произведение определяет новое отношение, которое является результатом конкатенации каждого кортежа из отношения R с каждым кортежем из отношения S.
7. **Тета-соединение** R ⋈θ S
   Определяет отношение, содержащее кортежи из декартового произведения R × S, удовлетворяющие предикату F = R.a_i θ S.b_j, где θ - одна из операций сравнения { >, <, =, ... }.
8. **Экви-соединение**
   Это тета-соединение, где θ = "=".
9. **Естественное соединение** R ⋈ S
   Соединение по эквивалентности двух отношений, выполненное по всем общим атрибутам, с исключением одного экземпляра каждого общего атрибута в результате.
10. **Левое внешнее соединение** R ⟕ S
    Это естественное соединение, при котором в результирующее отношение включаются также кортежи отношения R, не имеющие совпадающих значений в общих атрибутах отношения S.
11. **Полусоединение** R ▷θ S
    Это отношение, содержащее кортежи R, которые входят в тета-соединение R и S.
12. **Деление**

Проектирование БД

Концептуальная схема - это абстрактная модель данных, описывающая сущности и связи между ними, без учёта физической реализации (просто абстрактное описание данных)

- **Сущности и связи**: Определяются основные объекты и их взаимодействия (например, "Работник", "Завод" и связь "Работник работает на заводе")
- **ERD (диаграмма сущность-связь)**: Используется для визуализации сущностей и их атрибутов (например, "ФИО" у Работника)



Логическая схема - это представление данных с учётом реляционной модели, включая таблицы, типы данных и связи

- **Таблицы**: Каждая сущность становится таблицей (например, таблица "workers" с полями id, full_name, factory_id)
- **Типы данных и ограничения**: Определяются для каждого поля (например, VARCHAR, DATE, NOT NULL)
- **Связи**: Реализуются через внешние ключи (например, factory_id в "workers" ссылается на таблицу "Factories")

Порядок команд в SQL для написании запросов и порядок их выполнения:

Порядок команд при написании SQL-запросов:

1. **SELECT**
2. **DISTINCT**
3. **FROM**
4. **JOIN**
5. **WHERE**
6. **GROUP BY**
7. **HAVING**
8. **ORDER BY**
9. **LIMIT**

Порядок их выполнения:

1. **FROM**
2. **JOIN**
3. **WHERE**

4. **GROUP BY**
5. **HAVING**
6. **SELECT**
7. **DISTINCT**
8. **ORDER BY**
9. **LIMIT**

Индексы в реляционных БД:

Индексы в БД - это структура данных (key-value), которая ускорят выполнение запросов к БД, создавая отдельную структуры для быстрого поиска значений по атрибутам

Зачем нужны индексы?
Индексы повышают скорость выполнения запросов, позволяя базе данных быстро находить строк по значениям в индексируемых атрибутах

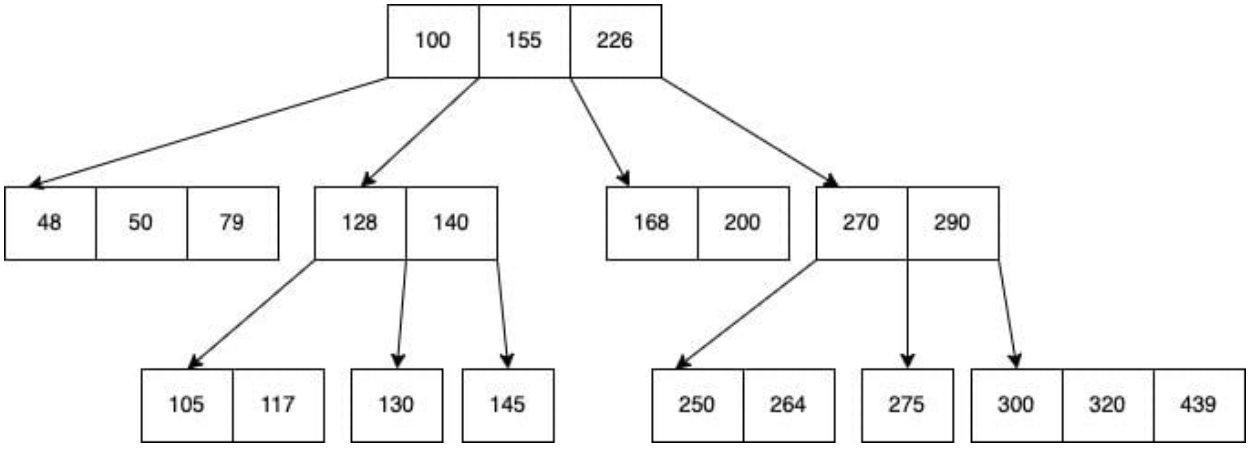
Первичный индекс — это индекс, построенный по первичному ключу при условии, что исходный файл отсортирован по нему же

Индекс кластеризации — это индекс, построенный по ключевому или неключевому полю при условии, что исходный файл отсортирован по нему же

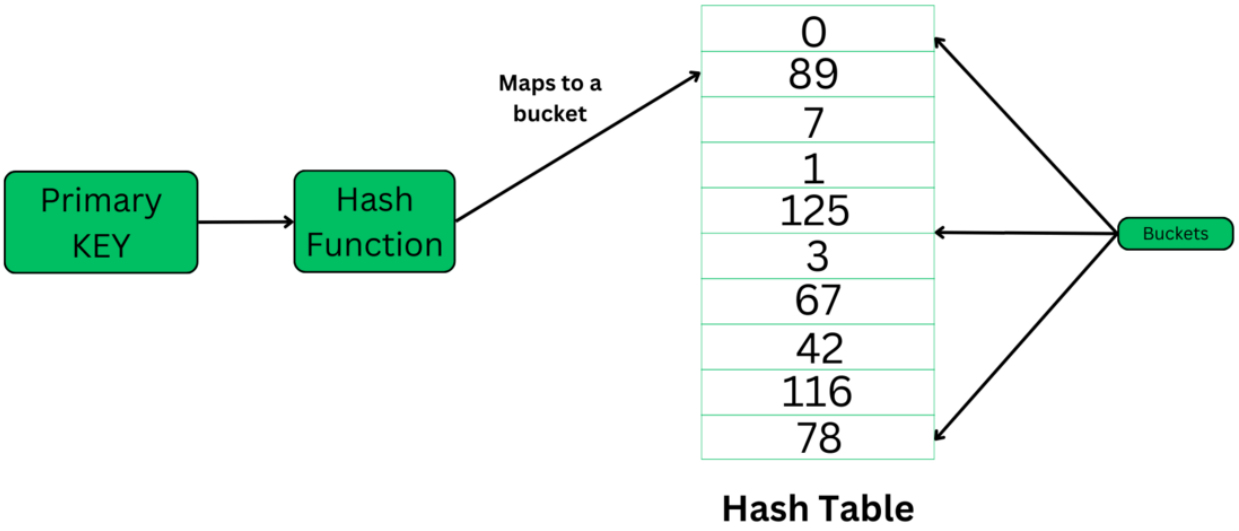
Вторичный индекс — это индекс, построенный по неключевому полю при условии, что исходный файл не отсортирован

Основные типы индексов:

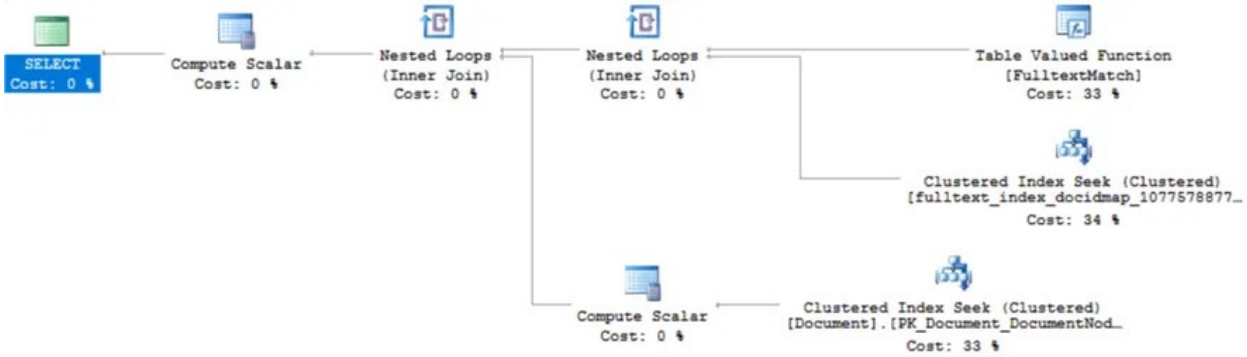
1. **B-дерево** - сбалансированное дерево, где данные организованы по иерархии. Каждый узел содержит несколько ключей и указатели на дочерни узлы. Подходит для поиска по диапазону и точного поиска, т.к поддерживает упорядоченность данных. Хорошо работает на большом количестве данных
   - **Поиск**: `O(logN)`
   - **Вставка**: `O(logN)`
   - **Удаление**: `O(logN)`
   - **Использование**: (`=`, `<`, `>`, `<=`, `>=`, `BETWEEN`)



2. **Хэш** - данный тип индексов использует хэш-функции в определенный хэш. Для каждого значения создаётся хэш, по которому находится место хранения строки. Например, если искомое значение - `123`, хэш-функция сопоставит его с конкретным местом в индексе, и система найдёт строки с нужным значением сразу. Быстрый для поиска равенства, но требует больше места для хранения, особенно падает скорость если встречаются коллизии
   - **Поиск**: `O(1)` в среднем, для точных совпадений, но при коллизии `O(N)`
   - **Вставка**: `O(1)`, при коллизии `O(N)`
   - **Удаление**: `O(logN)`, при коллизии `O(N)`
   - **Использование**: `=` и не поддерживает диапазонные запросы

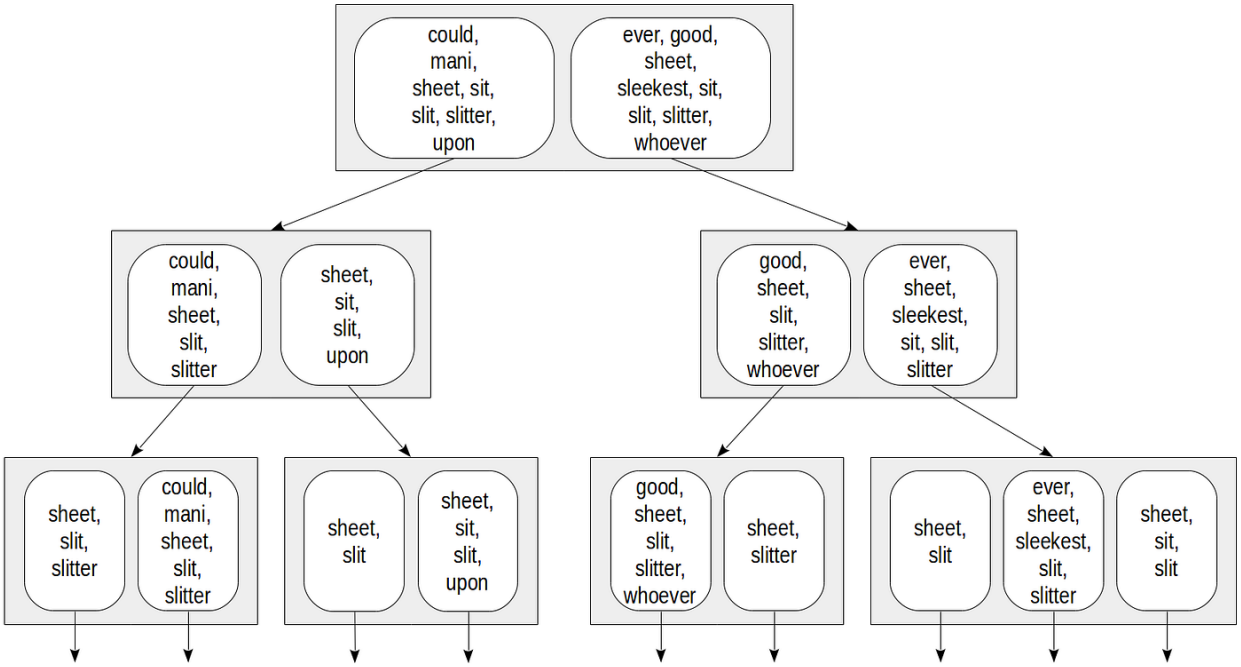


3. **Полнотекстовое индексирование** - метод индексации, ускоряющий текстовые запросы в базе данных. Текст разбивается на токены (слова), которые затем сохраняются в индекс
   - **Поиск**: `O(log N)` для извлечения токенов и `O(M)` для обработки результатов, где `M` - количество совпадений.
   - **Вставка**: `O(log N)`
   - **Удаление**: `O(log N)`
   - **Использование**: Для поиска по большим текстовым данным и базам с активными текстовыми запросами

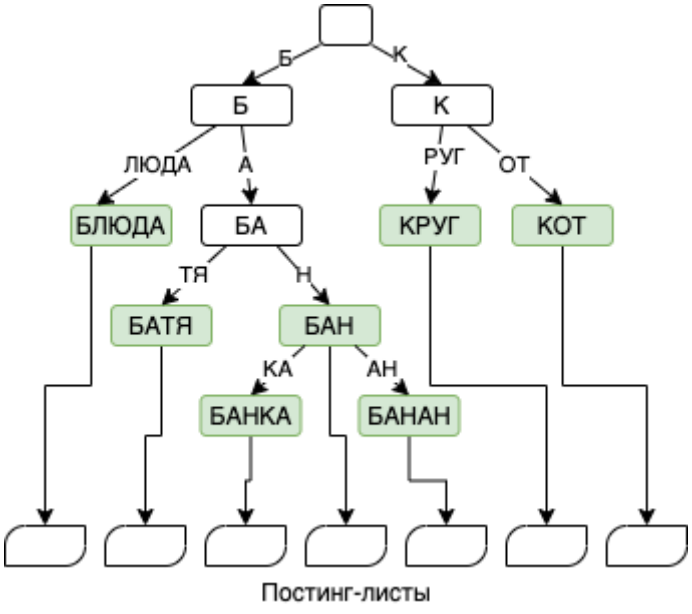


4. **GiST (Generalized Search Tree, обобщённое поисковое дерево)** - это индекс, используемый для работы со сложными типами данных, такими как геометрия, текст, массивы. GiST организован в виде сбалансированного дерева, где каждый узел представляет диапазон значений и связан с предикатом, проверяющим принадлежность значений диапазону

- **Поиск**: `O(log N)`
- **Вставка**: `O(log N)`, но может быть перебалансировка
- **Удаление**: `O(log N)`, но может быть перебалансировка
- **Использование**: Подходит для индексации нестандартных данных



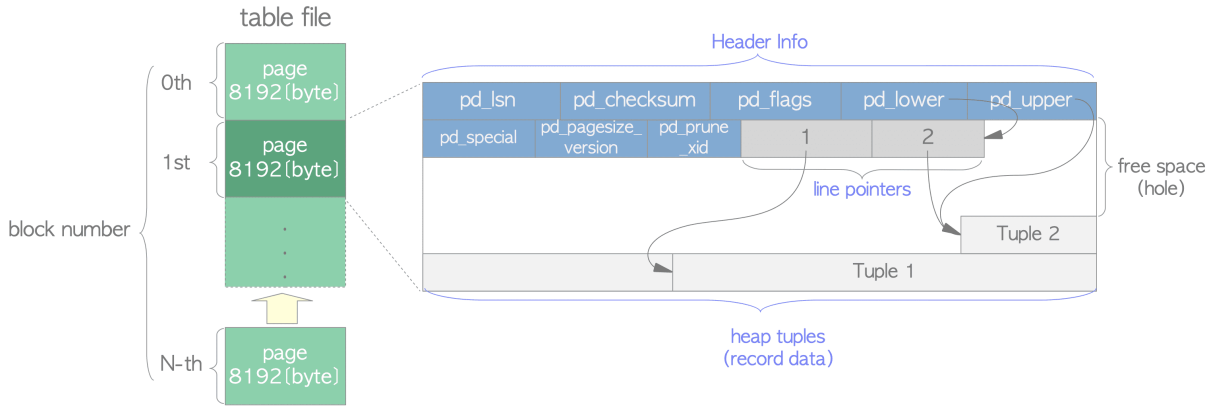
5. **Инвертированный индекс** - структура, используемая для быстрого поиска данных, особенно в полнотекстовом поиске. Каждый уникальный токен или слово указывается в индексе, и ему сопоставляются ссылки на документы или строки, содержащие этот токен. Обычно применяется в документоориентированных базах и поисковых движках
   - **Поиск**: `O(log N)` для быстрого поиска уникального токена и `O(M)` для извлечения всех строк или документов, где `M` - количество совпадений
   - **Вставка**: `O(log N)` для добавления нового токена и `O(1)` для добавления ссылки на существующий токен
   - **Удаление**: `O(log N)` для поиска и `O(1)` для удаления ссылки
   - **Использование**: Для полнотекстового поиска, фильтрации данных и ускоренного доступа к строкам с часто встречающимися словами



Как работает JOIN под капотом?

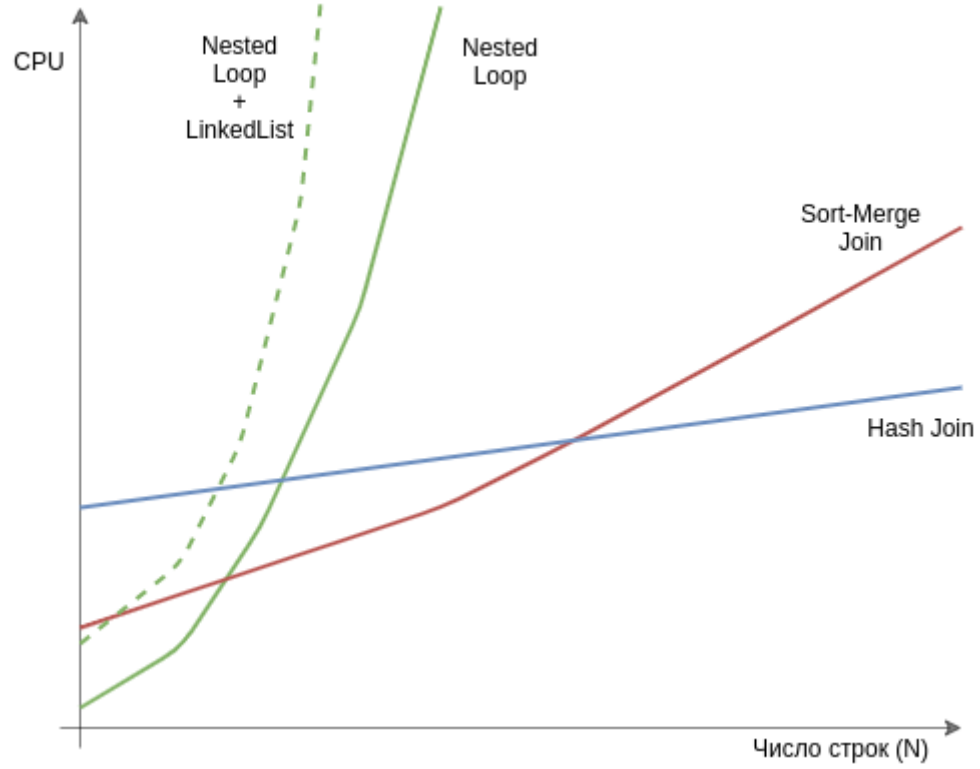
`JOIN` - это метод, который объединяет кортежи из двух таблиц на основании определенного условия. Существует несколько основных принципов работы `JOIN`:

1. **Nested Loop Join (Вложенные циклы)** - этот метод перебирает каждую строку из первой таблицы и для каждой строки ищет соответствующие строки во второй таблице. Если используется индекс по столбцу соединения, то поиск во второй таблице значительно ускоряется. Обычно СУБД использует это для маленьких таблиц. Асимптотика - `O(N * M)` где `N` и `M` количество кортежей в таблицах https://habrastorage.org/webt/vj/ck/ai/vjckaihodjn0a35_pyxfsdhvyry.gif
2. **Hash Join** - База данных сначала создает хэш-таблицу для одной из таблиц используя значения из атрибута соединения. Затем она проходит по другой таблице, выполняя поиск совпадений в хэш-таблице. Не требует индексов и хорошо работает с большим количеством данных. Отлично работает для операций `INNER JOIN` и `OUTER JOIN`. Асимптотика - `O(N + M)` где `N` и `M` количество кортежей в таблицах

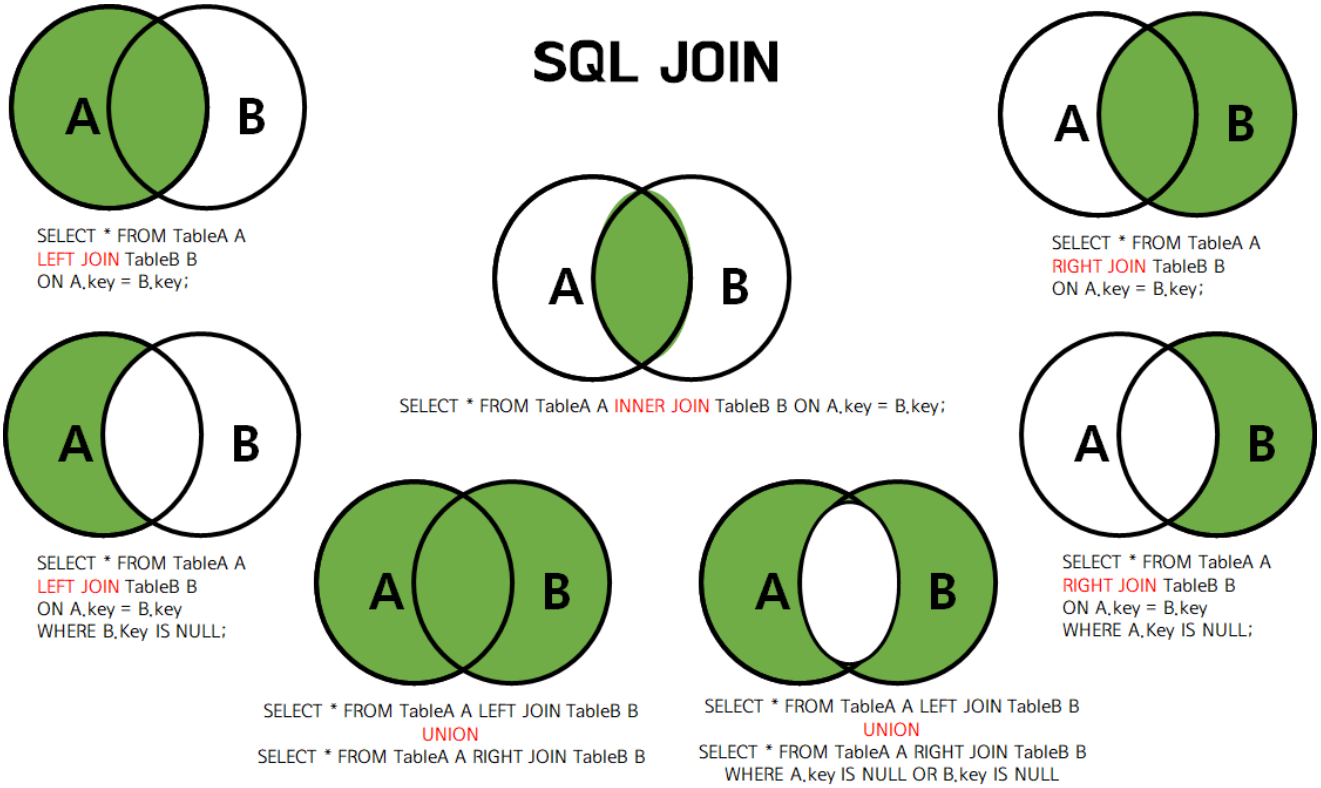


3. **Merge Join** - Сначала обе таблицы сортируются по столбцу соединения, после чего СУБД проходит по строкам обеих таблиц в отсортированном порядке и сопоставляет значения. Если данные уже отсортированы, например, из-за индекса B-дерева или GiST, Merge Join выполняет объединение сразу. Обычно используется для `JOIN` с операторами сравнения (`>`, `<`, `>=`, `<=`) и при наличии индексов, которые предоставляют отсортированные данные. Асимптотика - `O(N + M)`, если данные отсортированы; `O(N log N + M log M)`, если требуется сортировка, где `N` и `M` - количество строк в таблицах. https://habrastorage.org/webt/w9/wz/fv/w9wzfvnufmgbrzs79i7-ct_hlwk.gif

А вот общее сравнение алгоритмов:



Типы JOIN (INNER, OUTER, CROSS и т.д.):



Аномалии в БД

Аномалии данных - это проблемы, возникающие при внесении, обновлении или удалении данных в таблице, приводящие к нарушению целостности и избыточности данных.

Пример таблицы студентов

ФИО	Номер группы	Факультет
Иванов И.И.	М3210	ФИТИП
Петров П.П.	М3211	ФИТИП
Сидоров С.С.	М3212	ФИТИП
Кузнецов К.К.	М3213	ФИТИП
Смирнов С.С.	М3214	ФИТИП

Основные виды аномалий:

- **Аномалия модификации**
  Изменение данных в одной записи может потребовать внесения изменений в другие записи

```
Пример: Если изменить название факультета для группы М3212, то необходимо внести изменения для всех студентов этой группы. Если не внести их для одного из студентов, данные окажутся несогласованными
```

- **Аномалия удаления**
  Удаление одной записи может случайно удалить важную информацию

```
Пример: При удалении всех студентов группы М3212 мы потеряем информацию о принадлежности этой группы к факультету ФИТИП, хотя это свойство группы, а не конкретного студента.
```

- **Аномалия добавления**
  Невозможно добавить данные без указания дополнительных сведений, которые могут быть избыточными

```
Пример: Чтобы добавить нового студента в группу М3212, нужно указать факультет ФИТИП, даже если это значение повторяется для всех студентов этой группы.
```

Нормализация данных для СУБД (первая нормальная форма, вторая нормальная форма, третья нормальная форма, четвертая нормальная форма)

ФИО	Номер группы	Форма обучения	ОП	Факультет
Иванов И.И.	М3210	Контракт	ИС	ФИТИП
Петров П.П.	М3211	Бюджет	ИС	ФИТИП

ФИО	Номер группы	Форма обучения	ОП	Факультет
Сидоров С.С.	М3212	Бюджет	ИС	ФИТИП
Кузнецов К.К.	М3213	Контракт	ИС	ФИТИП
Смирнов С.С.	М3214	Бюджет	ИС	ФИТИП

Функциональная зависимость между атрибутами - это связь, при которой каждому значению атрибута *X* соответствует ровно одно значение атрибута *Y*.

$$R : X \to Y$$

1. **Полная функциональная зависимость** - явление, когда неключевой атрибут зависит от всего составного ключа

```
Пример:
Форма обучения, ОП и Факультет зависят от составного ключа ФИО + Номер
группы:
```

$$R : (ФИО, Номер\ Группы) \to Форма\ обучения,\ ОП,\ Факультет$$

2) **Частичная функциональная зависимость** - зависимость от части составного ключа

```
Пример:
Факультет и ОП зависят только от Номера группы, а не от полного ключа ФИО
+ Номер группы
Нет необходимости учитывать ФИО для определения факультета и
образовательной программы, так как номер группы уникально определяет эти
атрибуты
```

$$R : Номер\ Группы \to Факультет,\ Форма\ обучения$$

3. **Транзитивная функциональная зависимость** - зависимость через промежуточный атрибут

```
Пример:
Факультет транзитивно зависит от Номера группы через ОП
Поэтому, факультет можно определить через ОП, которая, в свою очередь,
зависит от номера группы
```

$$R : Факультет\ и\ Номер\ группы \to\ ОП \implies Факультет$$

Нормальные формы - это стандартизированные правила для организации данных в реляционных таблицах, которые помогают минимизировать избыточность и избежать аномалий

Нормализация - преобразования отношения к виду, отвечающему нормальной форме

1-я Нормальная Форма - таблица находится в **первой нормальной форме** (1НФ), если все ее атрибуты являются атомарными, то есть неделимыми
Текущая таблица находится в 1-ой нормальной форме

2-я Нормальная Форма - таблица находится во **второй нормальной форме**, если находится в первой нормальной форме и все неключевые атрибуты функционально полностью зависят от первичного ключа
Чтобы таблица соответствовала **2-ой нормальной форме** необходимо устранить **частичные функциональные зависимости**. В данном случае "ОП" и "Факультет" зависят только от "Номера группы", а не от составного ключа "ФИО + Номер группы". Поэтому нужно декомпозировать таблицу, разделив информацию на две таблицы

Таблица 1: Студент

ФИО	Номер группы	Форма обучения
Иванов И.И.	М3210	Контракт
Петров П.П.	М3211	Бюджет
Сидоров С.С.	М3212	Бюджет
Кузнецов К.К.	М3213	Контракт
Смирнов С.С.	М3214	Бюджет

Таблица 2: Группа

Номер группы	ОП	Факультет
М3210	ИС	ФИТИП
М3211	ИС	ФИТИП
М3212	ИС	ФИТИП
М3213	ИС	ФИТИП
М3214	ИС	ФИТИП

3-я Нормальная Форма - таблица находится в **третьей нормальной форме**, если она находится во второй нормальной форме и все неключевые атрибуты независимы друг от друга и функционально полностью зависят от первичного ключа (отсутствуют транзитивные зависимости)
Чтобы таблица соответствовала **3-ей нормальной форме** необходимо устранить **транзитивные зависимости**. Чтобы устранить транзитивную зависимость, нужно создать еще одну таблицу, связав атрибут "ОП" и "Факультет"

Таблица 1: Студент (без изменений)

ФИО	Номер группы	Форма обучения
Иванов И.И.	М3210	Контракт
Петров П.П.	М3211	Бюджет
Сидоров С.С.	М3212	Бюджет
Кузнецов К.К.	М3213	Контракт
Смирнов С.С.	М3214	Бюджет

Таблица 2: Группа (обновленная)

Номер группы	ОП
М3210	ИС
М3211	ИС
М3212	ИС
М3213	ИС
М3214	ИС

Таблица 3: Образовательная программа

ОП	Факультет
ИС	ФИТИП

Нормальная форма Бойса-Кодда (BCNF) - таблица находится в **нормальной форме Бойса-Кодда**, если детерминанты всех функциональных зависимостей являются потенциальными ключами. Разделение на 3 таблицы уже принадлежит **нормальной форме Бойса-Кодда**

4-я Нормальная Форма - таблица находится в **4-ой нормальная форме** если находится **нормальной форме Бойса-Кодда**, если детерминанты всех функциональных зависимостей являются потенциальными ключами