**Нам пизда☕:(**

Вопросы к экзамену по дисциплине

«Методы сбора, хранения, обработки и анализа данных»

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

***Концептуальное проектирование***

Цель – создание концептуальной модели данных исходя из представлений пользователей о предметной области

* Отказ от выбора окончательной структуры на ранней стадии выполнения работ
* Определение сущностей
* Определение атрибутов и доменов
* Определение связей
* Определение бизнес-правил

1. Определение сущностей и их документирование
2. Определение связей между сущностями и их документирование
3. Создание ER-модели предметной области
4. Определение атрибутов и их документирование:
   * + имя атрибута и его описание;
     + домен атрибута;
     + тип и размерность значений;
     + значение, принимаемое для атрибута по умолчанию;
     + может ли атрибут иметь Null-значения;
     + является ли атрибут составным
5. Определение значений атрибутов и их документирование
6. Определение первичных ключей для сущностей и их документирование
7. Обсуждение концептуальной модели данных с конечными пользователями

***Логическое проектирование систем хранения данных.***

Цель – преобразование концептуальной модели на основе выбранной модели данных в логическую модель, не зависимую от особенностей используемой в дальнейшем СУБД для физической реализации базы данных

1. Выбор модели данных
2. Определение набора таблиц исходя из ER-модели и их документирование
3. Нормализация таблиц
4. Проверка логической модели данных на предмет возможности выполнения всех транзакций, предусмотренных пользователями
5. Определение требований поддержки целостности данных и их документирование:
   * обязательные данные
   * ограничения для значений атрибутов
   * целостность сущностей
   * ссылочная целостность
   * ограничения, накладываемые бизнес-правилами
6. Создание окончательного варианта логической модели данных и обсуждение его с пользователями

Логическая схема базы данных для курсовых и дипломных проектов (IDEF1x):

* + Изобразить сущности, каждой дать имя
  + Изобразить связи
  + Вначале – составляющие ключа, затем прочие атрибуты
  + Типы данных не указываются
  + PK и FK указываются

Сбор данных:

Процесс получения детальных знаний о предметной области, требованиях пользователей и бизнес-процессах. Используются интервью, анализ документов, анкетирование, наблюдение.

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:

Формализованное описание всех аспектов проекта БД, включая требования, модели, правила.

Перечень границ проекта. Перечень отрицательного опыта пользователей (чего избегать). Запросы пользователей. Поддержка общедоступного хранилища документации с версионностью.

Говорим с заказчиком: **интервью**, **анкеты**, **обратная связь**, — надо понять, **кто**, **зачем** и **как будет юзать данные**.

Смотрим в старую **документацию**, **старый код**, **журналы**, **отчёты** — вдруг что полезное найдём.

\* \*\*Минимальные информационные требования:\*\* Определение основного набора данных, без которого система не сможет выполнять свои основные функции и удовлетворять ключевые потребности бизнеса. Сущности, атрибуты, связи, обязательность данных ,уникальность и ограничения, бизнес-правила.

\* \*\*Источники правил данных:\*\* Бизнес-политики, нормативные документы, существующие системы, говорим с заказчиком: интервью с экспертами предметной области, пользовательские интерфейсы и отчеты.

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

**Логическое проектирование систем хранения данных (выше)**

Реляционная и нереляционная модели

Реляционная- СУБД хранит данные в таблицах с жёсткой структурой, использует SQL и поддерживает ACID-транзакции (Oracle, MySQL)

Нереляционная СУБД (NoSQL) использует гибкие модели (документы, ключ-значение, графы), масштабируется лучше, но без строгой схемы

1. Ключ – значение (Cassandra)

2. Сетевые (графовые, Gremlin)

3. Документные (MongoDB)

4. Табличные (с частичной поддержкой SQL)

Сущности атрибуты, отношения и бизнес-правила.

Сущность (Entity) – объект, информация о котором хранится в БД (например, Студент, Заказ, Продукт).

Атрибут (Attribute) – характеристика сущности (например, для Студент: Имя, Возраст, Группа).

Отношение (Relationship) – связь между сущностями (например, Студент → зачислен в → Группа).

Бизнес-правила (Business Rules) – ограничения и логика работы данных (например, "Зарплата не может быть отрицательной" или "Заказ должен иметь хотя бы один товар").

Методологии моделирования

UML – стандартная методология для определения и документирования программных систем

IDEF1X – Integration Definition For Information Modelling – методология для реляционных данных

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

IE – Informational Engineering – моделирование связей между таблицами

Изображение выглядит как текст, снимок экрана, диаграмма, линия

Содержимое, созданное искусственным интеллектом, может быть неверным.

Нормализация — это процесс устранения избыточности в базе данных путём разбиения таблиц на связанные структуры с минимальным дублированием данных.

· Устранение NULL

· Устранение избыточности данных

· Устранение ненужного кодирования

· Максимизация кластерных индексов

· Уменьшение числа индексов на таблицу

· Хранение тонких таблиц

1 НФ:

Все атрибуты должны быть элементарными. Атомарные значения.

Экземпляры сущности должны иметь одно и то же количество значений

Все экземпляры сущности должны быть различны

2 НФ:

Сущность должна соответствовать 1 НФ

Каждый атрибут должен зависеть от ключа

3 НФ:

Сущность должна соответствовать 2 НФ

Каждый атрибут должен зависеть только ключа. Нет транзитивных зависимостей

4 НФ:

– Сущность находится в НФБК

– Не должно быть больше одной зависимости с многими значениями, представленной в сущности

Проблемы:

– Тройные отношения

– Скрытые атрибуты с несколькими величинами

– Атрибуты с предыдущими значениями

• НФ Бойса-Кодда:

– Все атрибуты полностью зависят от ключа

– Сущность находится в НФБК, если каждый детерминант – ключ

• **Детерминант** – любой атрибут или комбинация атрибутов, от которых функционально зависит любой другой атрибут или комбинация атрибутов

• Если набор столбцов является ключом, то необходимо внести ограничение уникальности

Денормализация- намеренное добавление дублирования данных или объединение таблиц для ускорения запросов за счёт увеличения избыточности. Используется в аналитических базах (OLAP) и при оптимизации чтения.

Используется для улучшения работы:

• Вычисленные атрибуты (TotalSum)

• Преимущественные значения (PreferredPhoneNumber)

• Отметка изменений (LastUsage)

План проекта использования данных

1. Цели – Определить бизнес-задачи

2. Требования – Выявить источники данных и потребности пользователей.

3. Архитектура – Выбрать СУБД (SQL/NoSQL), инструменты ETL и аналитики.

4. Разработка – Создать БД, настроить интеграцию и тестирование.

5. Аналитика – Построить дашборды (Power BI, Tableau) или ML-модели.

6. Внедрение – Запустить пилот, обучить пользователей, настроить мониторинг.

7. Оценка – Проверить KPI, собрать обратную связь, масштабировать при успехе.

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

Физическое проектирование и реализация

Цель – описание конкретной реализации базы данных

Проблемы:

• Размер и сложность данных

• Поиск

• Конкуренция за ресурс

• Своевременность и частота отчетов

• Бюджет

Реализация

1. Согласование архитектуры
2. Проектирование и разработка таблиц базы данных средствами выбранной СУБД
3. Реализация бизнес-правил в среде выбранной СУБД
4. Проектирование и реализация физической организации базы данных
5. Разработка стратегии защиты базы данных
6. Организация мониторинга функционирования базы данных и ее настройка

Физическая схема базы данных для курсовых и дипломных проектов (IDEF1x):

* + К логической добавляются типы данных через двоеточие
  + Указываются индексируемые поля

Проблемы использования данных, связанные с размером данных

• Медленные запросы из-за объема (миллионы строк)

• Ограничения памяти (буферный кэш, сортировка)

• Рост логов (архивация, место на диске)

• Блокировки при высокой конкурентности

Влияние физических характеристик

• Диск (HDD/SSD/NVMe) – скорость чтения/записи

• RAM – объем влияет на кэширование

• CPU – обработка сложных запросов

• Сеть – задержки при распределенных данных

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

Физическое проектирование и реализация (выше)

Основные топологии приложений.

Приложение – база данных

Приложение – несколько баз данных

SOA

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

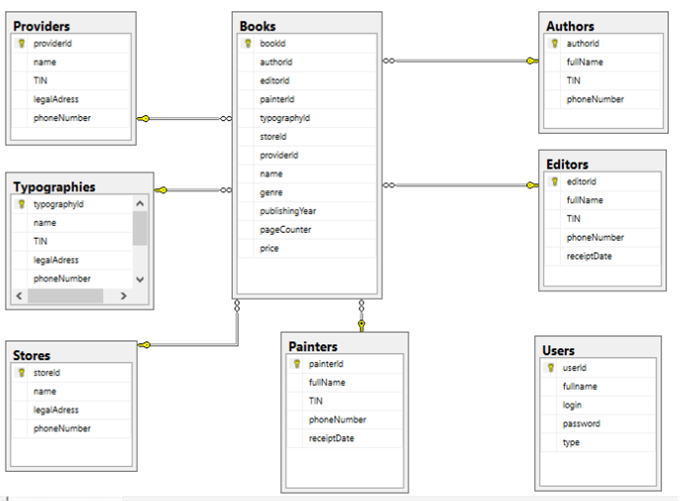
Содержимое, созданное искусственным интеллектом, может быть неверным.

* SOA – Service-Oriented Architecture – Сервис-ориентированная архитектура -  модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам

Изображение выглядит как текст, диаграмма, линия, План

Содержимое, созданное искусственным интеллектом, может быть неверным.

Схема физического проектирования



Обеспечение целостности данных.

Обеспечение целостности данных — задача, требующая использования различных методов и инструментов. Регулярное резервное копирование, контроль доступа, шифрование и использование цифровых подписей — все это помогает защитить данные от различных угроз.

Расширенный доступ к данным.

API и микросервисы: Предоставление API и разделение функциональности на микросервисы для удобного доступа к данным.

Использование универсальных интерфейсов доступа (ODBC, OLE DB, ADO.NET);

Провайдеры данных для разных СУБД (SQL Server, Oracle и др.).

**API для доступа к данным**

* Доступ к данным – прикладной программный интерфейс для СУБД
* Набор функций:
  + установление и закрытие соединения
  + обновление данных
  + передача запросов серверу
  + получение результатов выполнения запросов
  + получение кодов ошибок
  + характеристики структуры набора результата

**Универсальный механизм доступа к данным**

* Реализован в виде библиотек и дополнительных модулей – драйверов или провайдеров
* Библиотеки содержат стандартный набор функций или классов, подчиняющийся спецификации
* Дополнительные модули реализуют непосредственное обращение к функциям конкретных СУБД
* Легко модифицировать, если необходима смена СУБД
* Изменяются только настройки доступа к данным
* Невозможность доступа к уникальной функциональности, специфичной для конкретной СУБД
* Снижение производительности приложений
* Усложнение процедуры поставки приложения

**Microsoft Data Access Components**

* ODBC – Open Database Connectivity
* OLE DB – Object Linking and Embedding Database
* ADO.NET

**Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Постоянное подключение**

* Установка соединения
* Подготовка и выполнение команды
* Работа с данными
  + чтение, запись
  + фильтрация, сортировка
  + тоже в пакетном режиме
  + блокировки, совместное использование
* Закрытие соединение и обработка ошибок
* Работает в режиме удержания подключения к базе
* Обеспечивает максимальную гибкость и эффективность
* Обеспечивает минимальный расход оперативной памяти

**Отсоединенные данные**

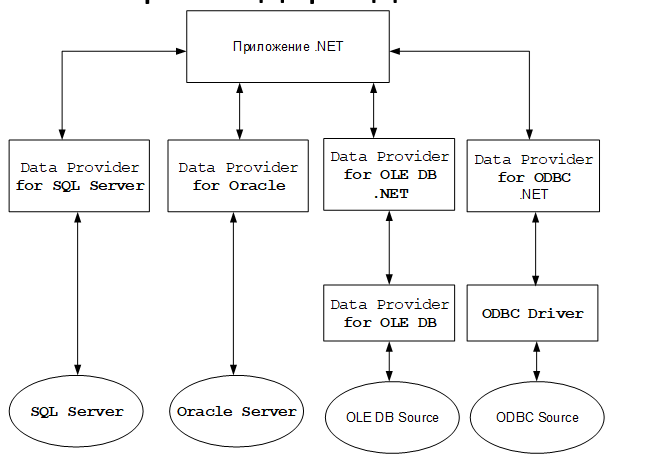
* Загрузка данных с сервера
* Изменение данных в наборе на локальной машине
* Обновление данных на сервере на основе локальной копии
* Обеспечивает работу с данными в отсутствии подключения к БД
* Удобна для переноса данных по сети
* Расходует достаточно много памяти

**Провайдеры данных**

* Провайдеры (или поставщики) данных – извлечение данных из источника данных
  + SQL Server .NET Data Provider
  + Oracle Data Provider
  + ODBC.NET Data Provider
  + OleDB.NET Data Provider

**Изображение выглядит как текст, Шрифт, снимок экрана, белый

Содержимое, созданное искусственным интеллектом, может быть неверным.**

****

* Обеспечивают большую производительность
* Позволяют работать со специфическими типами данных для данной СУБД
* Лучше выполняют специфические для данной СУБД функции
* Можно использовать неспециализированного провайдера данных
* Можно определить набор провайдеров данных (модель ProviderBase)

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

**Обеспечение безопасности**:

* **Аутентификация**: Важно использовать надежные методы проверки пользователей, такие как многофакторная аутентификация (MFA), чтобы предотвратить несанкционированный доступ. Параметры паролей (длина, сложность) также должны быть строгими.
* **Авторизация**: Используйте ролевую модель, чтобы назначать права на основе ролей пользователей (например, администраторы, обычные пользователи).
* **Шифрование**: Данные должны шифроваться как в состоянии покоя (например, на жестком диске), так и в транзите (при передаче по сети). Использование стандартов, таких как AES для шифрования, является хорошей практикой.
* **Ограничение прав доступа**

**Определение требований к аппаратным средствам**

* **Производительность**: Оценка потребностей в вычислительной мощности и памяти. Это включает выбор процессоров с высокой производительностью, достаточного объема оперативной памяти и быстрых дисков (SSD) для обеспечения быстрого доступа к данным.
* **Надежность**: Аппаратные средства должны быть надежными и включать механизмы резервирования (например, RAID для дисков) и бесперебойные источники питания (ИБП) для защиты от сбоев.
* **Сетевые требования**: Определение необходимых характеристик сети, таких как пропускная способность и задержка, для обеспечения стабильной работы базы данных.

**Определение параметров роста данных**:

* **Прогнозирование объема данных**: Нужно анализировать исторические данные и учитывать бизнес-планы для прогнозирования будущего роста. Это поможет в планировании расширения баз данных и оборудования.
* **Мониторинг**: Регулярное отслеживание роста данных с помощью специальных инструментов и метрик. Это позволит своевременно выявлять тенденции и принимать меры по масштабированию системы.

**Определение параметров архивирования**:

* **Политика архивирования**: Определить, какие данные подлежат архивированию на основе их использования. Например, данные, которые не изменяются более года, могут быть архивированы.
* **Хранение архивов**: Выбор безопасных мест для хранения архивов. Это могут быть облачные решения, которые обеспечивают доступность и безопасность, или внешние носители, которые защищены от киберугроз.
* **Сроки хранения**: Установить правила для удаления старых архивов, чтобы освобождать пространство и соблюдать требования законодательства о хранении данных.

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

Иерархии­структура данных или системы, где элементы организованы в виде дерева с отношениями «родитель-потомок

• организационные графики

• структуры предприятий

• списки файлов и папок

• каталоги продуктов

• ветки форумов

Виды организации иерархических данных.

· Adjacency List – список смежных вершин (parent\_id)

· Nested Set – указание вложенности (left/right lower/upper)

· Materialized Path – полный путь (1/2/1)

· Добавление уровня – level

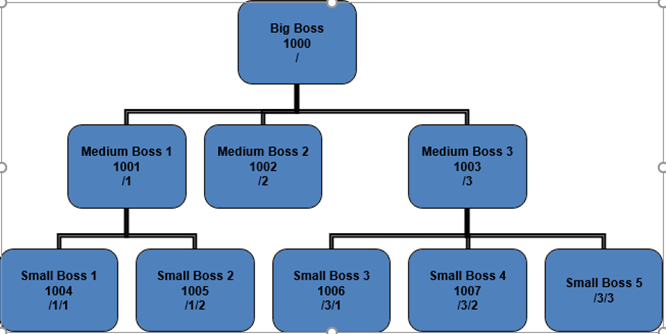
Иерархический тип данных в SQL Server.

Hierarchyid:

· системный тип данных переменной длины

· используется для представления положения в иерархии

· путь логически представлен в виде последовательности меток всех посещенных дочерних узлов, начиная с корня



Методы hierarchyid:

· GetRoot – получения корня

· GetLevel – получение уровня

· GetDescendant (child1, child2) – создание дочернего узла между child1 и child 2

· GetAncestor (n) – возвращает предка на уровне n

· IsDescendantOf – проверка, является ли узел потомком

· GetReparentedValue (oldRoot, newRoot)

· Parse – преобразование в число

ToString – для материализованного пути

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

Иерархии (выше). Хранение иерархических данных.

Рекурсивная связь (Adjacency List)

Самая простая модель. Таблица содержит ссылку на родительский элемент:

|  |
| --- |
| CREATE TABLE employees (  id NUMBER PRIMARY KEY,  name VARCHAR2(100),  manager\_id NUMBER REFERENCES employees(id)  ); |

Materialized Path (Путевой материал)

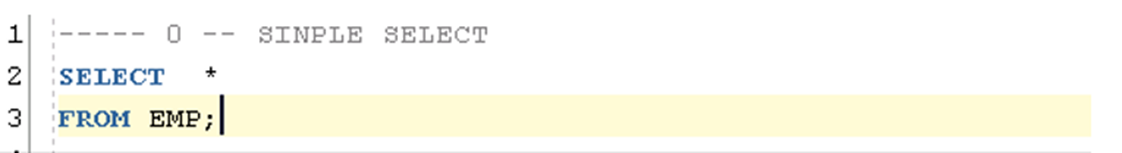
|  |
| --- |
| CREATE TABLE categories (  id NUMBER PRIMARY KEY,  name VARCHAR2(100),  path VARCHAR2(1000) -- например '1.4.12'  ); |

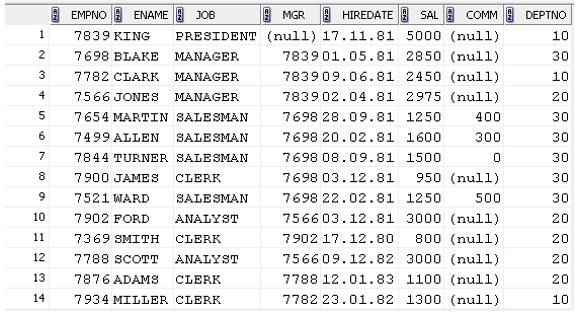
Nested Sets (Вложенные множества)

Использует левые и правые границы:

|  |
| --- |
| CREATE TABLE org\_structure (  id NUMBER PRIMARY KEY,  name VARCHAR2(100),  lft NUMBER,  rgt NUMBER); |

Иерархические запросы в Oracle

1.



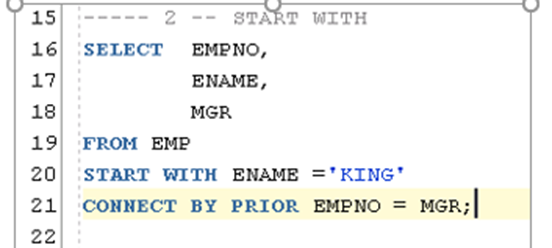
2. Start with – начальный узел иерархии

Connect by – связь текущего с родительским

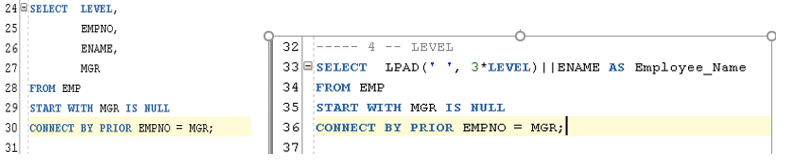
Порядок прохода записей – от корня обход в глубину



3. Start with – начальный узел иерархии

4. LEVEL – уровень иерархии

 Изображение выглядит как текст, число, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным. Изображение выглядит как текст, снимок экрана, число, Шрифт

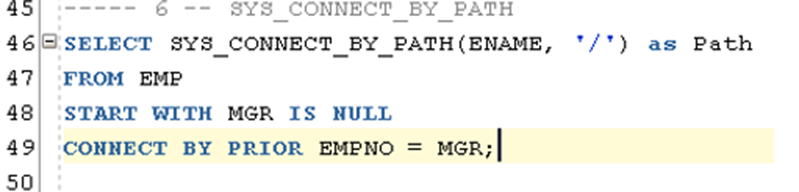
Содержимое, созданное искусственным интеллектом, может быть неверным.

5. ORDER SIBLINGS – упорядочение в рамках уровня

 Изображение выглядит как текст, снимок экрана, число, чек

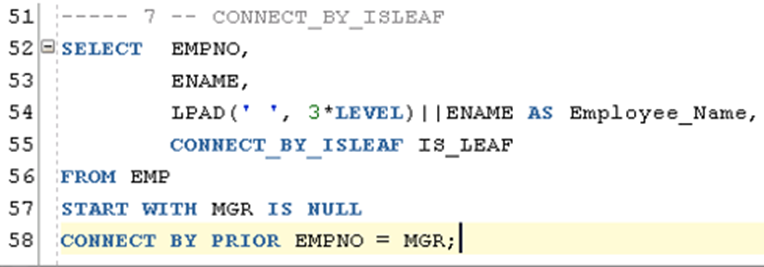
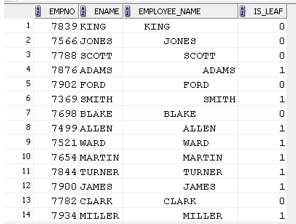
Содержимое, созданное искусственным интеллектом, может быть неверным.

6. SYS\_CONNECT\_BY\_PATH – материализованный путь в иерархии

 Изображение выглядит как текст, снимок экрана, Шрифт, число

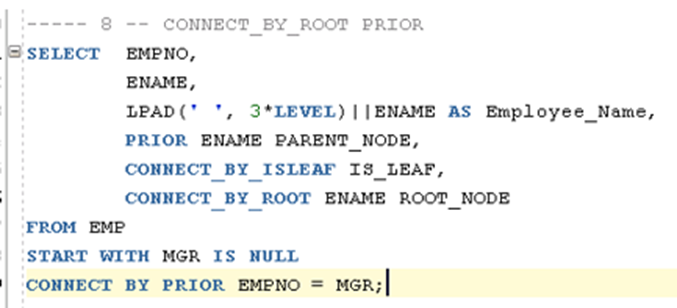
Содержимое, созданное искусственным интеллектом, может быть неверным.

7. CONNECT\_BY\_ISLEAF – является ли узел листовым

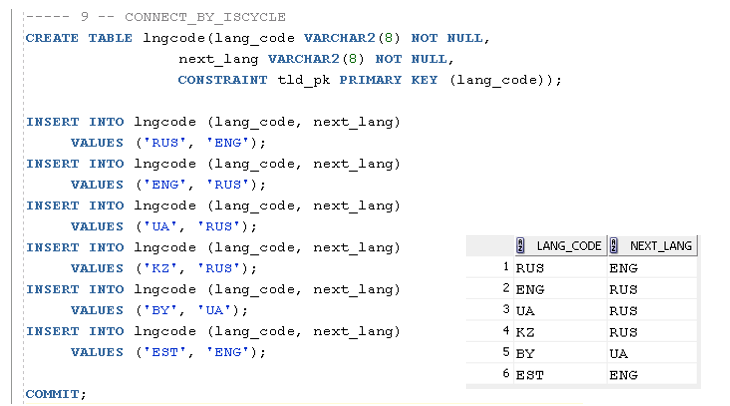
 

8. PRIOR – родительский узел

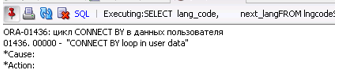
CONNECT\_BY\_ROOT – корневой узел иерархии

 Изображение выглядит как текст, снимок экрана, число, Шрифт

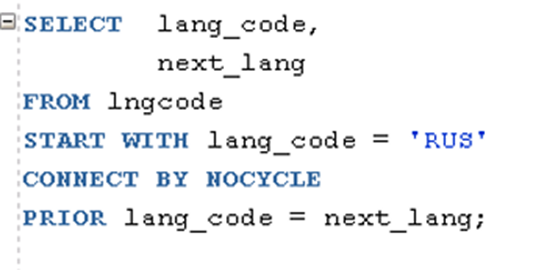
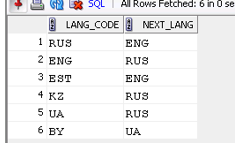
Содержимое, созданное искусственным интеллектом, может быть неверным.



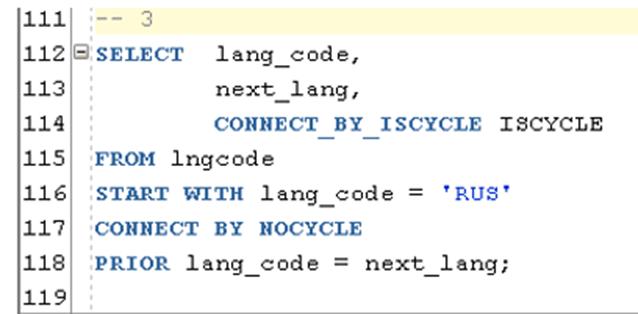
9. CONNECT\_BY\_LOOP – зацикливание

10.CONNECT BY NOCYCLE – cоединение до зацикливания

11. CONNECT\_BY\_ISCYCLE – 1, если вызывает зацикливание, 0 – в остальных случаях

 Изображение выглядит как текст, число, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.

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

Обобщенные табличные выражения – временные результирующие наборы, определенные в области выполнения инструкций SELECT. Не сохраняются в бд в виде объектов. Время жизни ограничено продолжительностью запроса. Могут ссылаться сами на себя. Один и тот же запрос может ссылаться на CTE (Common Table Expressions) несколько раз

Применение ОТВ.

Нельзя использовать:

· ORDER BY (только вместе с TOP )

· INTO

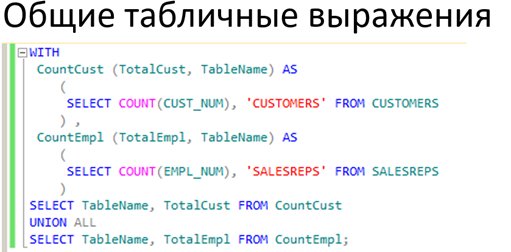
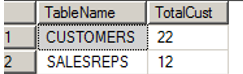
· OPTION (для хинтов)

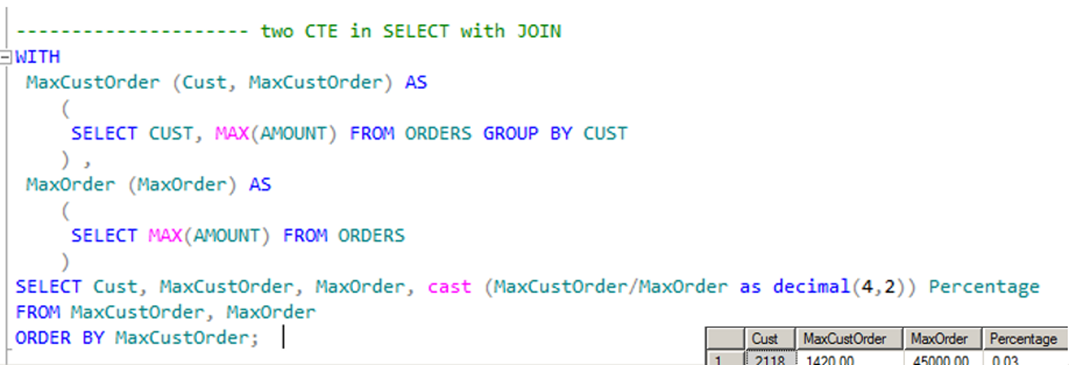
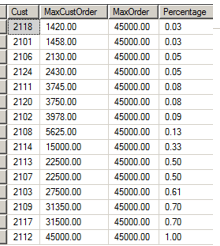
Можно использовать:

· Несколько CTE

· UNION (ALL), INTERSECT, EXCEPT

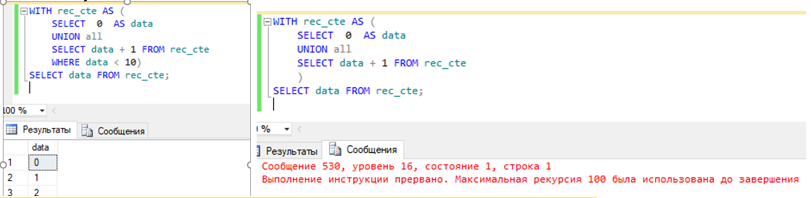
· Сослаться на предварительно объявленный СТЕ

Рекурсивные ОТВ

Рекурсивность – обращение через CTE к самому себе



Используются для

* Создания рекурсивных запросов
* Многократных ссылок на результирующую таблицу из одной и той же инструкции
* Замены представлений
* Позволяет значительно повысить читаемость и упростить работу со сложными запросами

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

Графы.

Графы — это структуры данных, состоящие из вершин (узлов) и ребер (связей) между ними. Они широко используются для представления сложных отношений: социальные сети, транспортные сети, рекомендательные системы.

1. Основные понятия

Вершина (Node) — сущность (например, пользователь, город, товар).

Ребро (Edge) — связь между вершинами (дружба, маршрут, покупка).

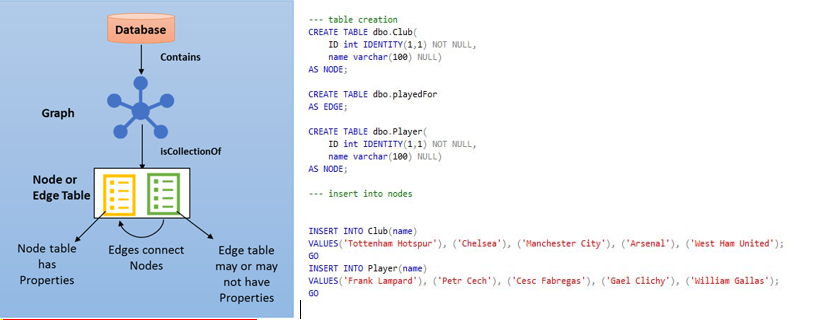
Свойства — атрибуты вершин и ребер (вес, дата создания).

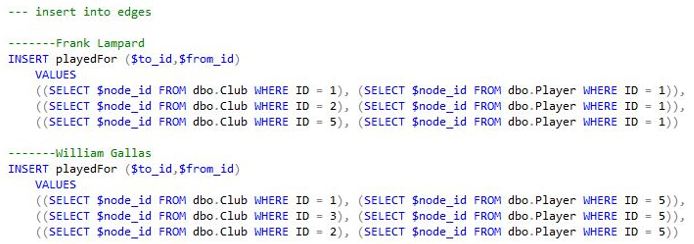
Графовые базы данных.

· С SQL Server 2017

· Можно создать один граф на базу данных

· Граф состоит из таблиц узлов и ребер





Ключевое слово MATCH.

Ключевое слово MATCH используется в графовых бд для поиска путей и шаблонов связей между узлами.

1. **Поиск игроков, которые играли за определенный клуб**:

SELECT p.name AS PlayerName, c.name AS ClubName

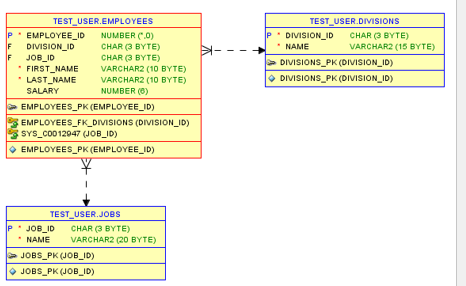
FROM Player p, playedFor pf, Club c

WHERE MATCH(p-(pf)->c)

AND c.name = 'Chelsea';

* (node1)-[edge]->(node2) - направленная связь от node1 к node2
* (node1)-[edge]-(node2) - ненаправленная связь

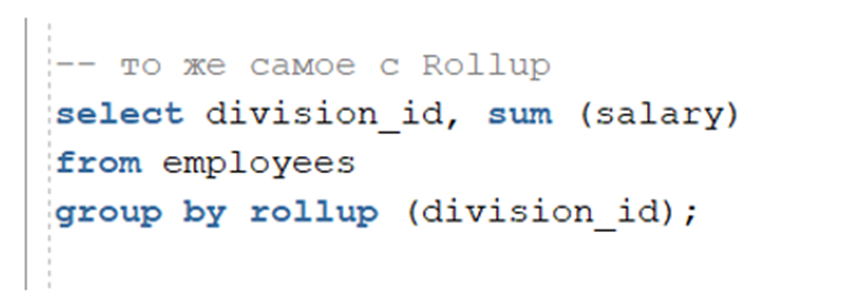
10. Расширенные группировки. Применение расширенных группировок в SQL Server и в Oracle.

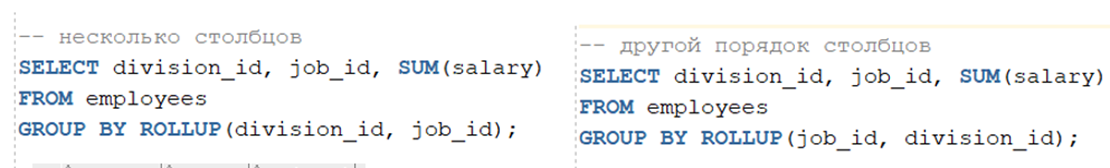


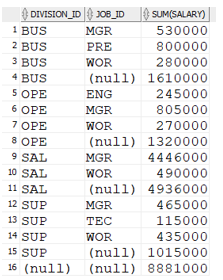
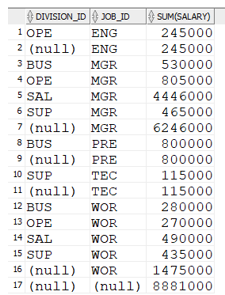
Расширенные группировки

•Rollup – возвращает строки итогов и предварительных итогов

ROLLUP Иерархические подытоги: ROLLUP создает иерархические промежуточные суммы, начиная с подробных группировок и заканчивая общим итогом. Применение: Полезен для иерархических данных, таких как "страна -> штат -> город" или "отдел -> менеджер -> продавец".

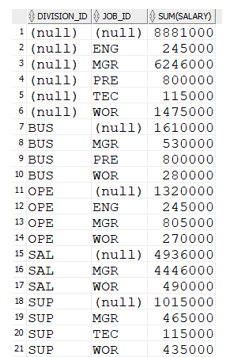
 

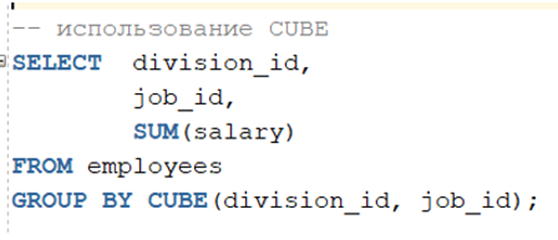
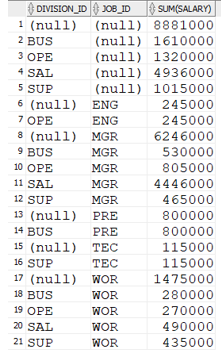


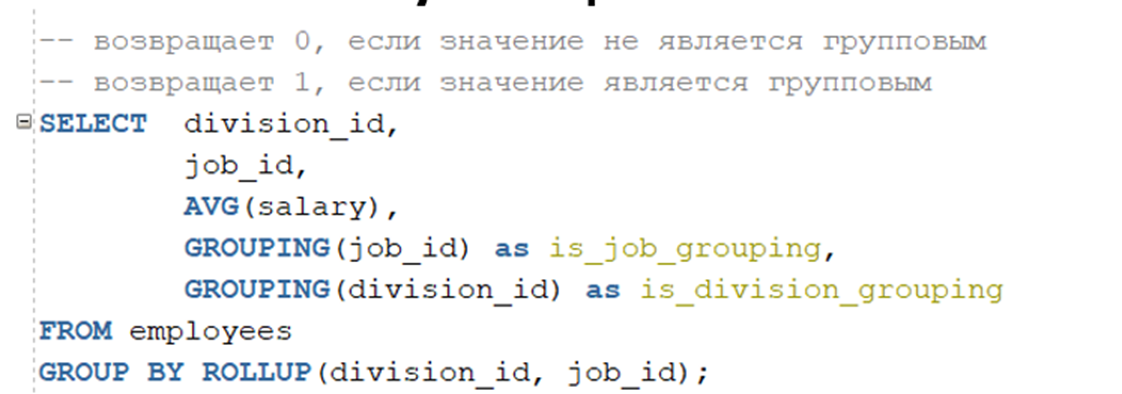
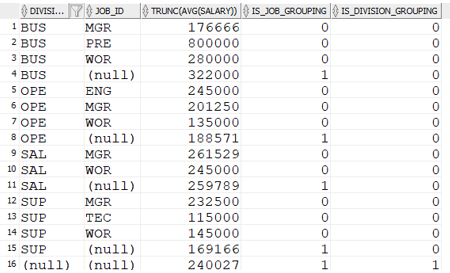
• Cube – предварительные итоги для каждой комбинации строк

Применение: Особенно полезен для многомерного анализа данных, когда необходимо исследовать различные комбинации характеристик.

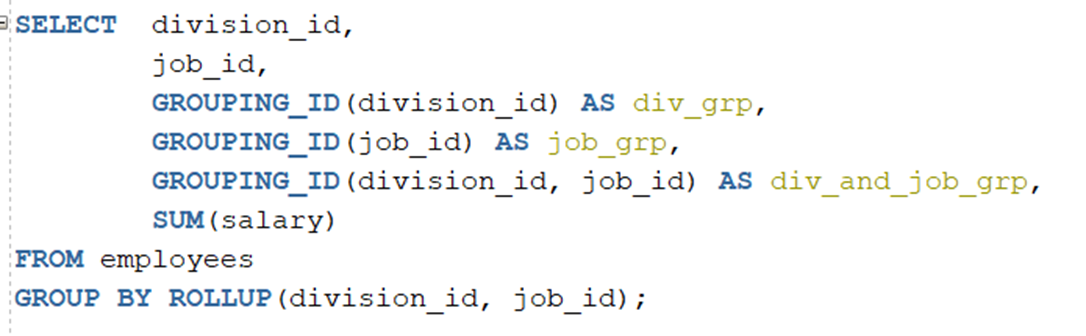
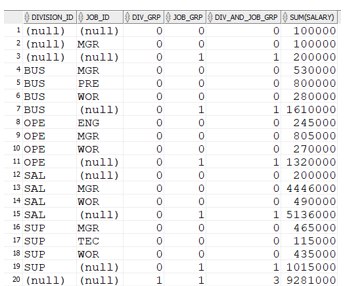
 

• GROUPING – Возвращает 1 или 0, указывая, участвует ли столбец в группировке.

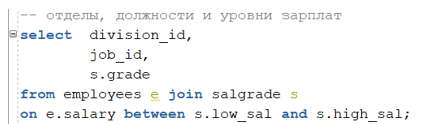
• GROUPING\_ID – уровень группировки.  
 Формирует значение битового вектора в зависимости от значений NULL в столбцах группировки

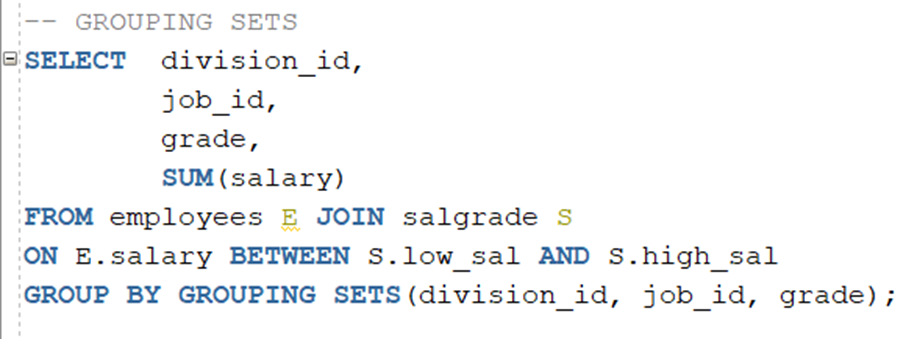
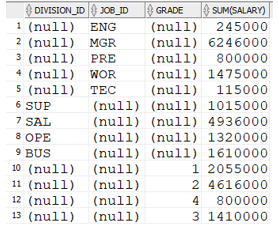
Изображение выглядит как текст, Шрифт, снимок экрана, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

• GROUPING SETS – наборы столбцов для группировки

 Изображение выглядит как текст, Шрифт, снимок экрана, белый

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Применение в SQL Server**

В SQL Server концепция аналогична, но синтаксис может слегка отличаться. Например, GROUP BY ROLLUP, CUBE и GROUPING SETS также поддерживаются.

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

Аналитические функции.

Аналитические функции (или оконные функции) - это функции SQL, которые позволяют осуществлять вычисления в заданном диапазоне строк (окне) внутри одного предложения SELECT. В отличие от агрегатных функций с GROUP BY, оконные функции не приводят к группированию строк - строки сохраняют свои идентификаторы, а агрегированное значение добавляется к каждой строке.

Типичные задачи: подсчет промежуточной суммы, подсчет процентов в группе, запросы первых N, подсчет скользящего среднего, выполнение ранжирующих запросов

employee\_id | salary

------------+-------

101 | 2500

102 | 3000

103 | 3500

104 | 4000

105 | 4500

106 | 5000

1. **PERCENT\_RANK() — "Какой процент людей получает меньше, чем я?"**

Функция PERCENT\_RANK() вычисляет относительный ранг строки в наборе результатов (от 0 до 1), показывая, какой процент значений меньше текущего значения.

SELECT

employee\_id,

salary,

PERCENT\_RANK() OVER (ORDER BY salary) AS percent\_rank

FROM employees;

employee\_id | salary | percent\_rank

------------+--------+-------------

101 | 2500 | 0.00 ← Никто не получает меньше 2500

102 | 3000 | 0.20 ← 20% получают меньше 3000 (только 101)

103 | 3500 | 0.40 ← 40% получают меньше 3500 (101 и 102)

104 | 4000 | 0.60 ← 60% получают меньше 4000

105 | 4500 | 0.80 ← 80% получают меньше 4500

106 | 5000 | 1.00 ← Все получают меньше 5000 (кроме самого 106)

PERCENT\_RANK = (количество людей с зарплатой меньше моей) / (общее количество людей - 1)  
Для 3000: (1 человек получает меньше) / (6 - 1) = 0.2 (20%).

**2. CUME\_DIST() — "Какая доля людей получает не больше, чем я?"**

Функция CUME\_DIST() вычисляет кумулятивное распределение значения в наборе данных - долю строк со значениями меньше или равными текущему значению.

SELECT

employee\_id,

salary,

CUME\_DIST() OVER (ORDER BY salary) AS cume\_dist

FROM employees;

employee\_id | salary | cume\_dist

------------+--------+----------

101 | 2500 | 0.17 ← 1/6 людей получают ≤2500

102 | 3000 | 0.33 ← 2/6 людей получают ≤3000

103 | 3500 | 0.50 ← 3/6 людей получают ≤3500

104 | 4000 | 0.67 ← 4/6 людей получают ≤4000

105 | 4500 | 0.83 ← 5/6 людей получают ≤4500

106 | 5000 | 1.00 ← 6/6 людей получают ≤5000

Разница с PERCENT\_RANK: CUME\_DIST включает текущую строку в расчет, поэтому значения обычно выше.

3. **PERCENTILE\_CONT() — "Какая зарплата на границе N%?" (среднее значение)**

Функция PERCENTILE\_CONT() вычисляет процентиль на основе непрерывного распределения с интерполяцией.

**Пример:** Найдём медиану зарплаты (50-й процентиль).

SELECT

PERCENTILE\_CONT(0.5) WITHIN GROUP (ORDER BY salary) AS median\_salary

FROM employees;

**Как считается:**

1. В таблице 6 записей.
2. 50% от 6 = 3-я запись (3500).
3. Но если бы записей было не**чётное** число (например, 5), то медиана = среднее между 2-й и 3-й зарплатой.

3500 <- медиана зарплат

Это вычисляет медиану (50-й процентиль) зарплаты для каждого отдела. Если точного значения нет, выполняется интерполяция между соседними значениями.

4. PERCENTILE\_DISC()

Функция PERCENTILE\_DISC() вычисляет процентиль на основе дискретного распределения, возвращая конкретное значение из набора данных.

SELECT

PERCENTILE\_DISC(0.5) WITHIN GROUP (ORDER BY salary) AS median\_salary

FROM employees;

3500 <- конкретное значение из данных

**Разница между PERCENTILE\_CONT и PERCENTILE\_DISC:**

* CONT интерполирует между значениями, если точного процентиля нет
* DISC всегда возвращает конкретное существующее значение из набора данных

Если бы медиана попадала **между двумя значениями** (например, между 3500 и 4000):

* + CONT вернёт **3750** (среднее).
  + DISC вернёт **3500** (меньшее из двух).

Типичный синтаксис

* **Функция – SUM()**
* **Аргумент - SALARY**
* **OVER** – срез данных (область, над которой будут производиться вычисления)
* **PARTITION BY** – фрагментация
* **ORDER BY –** сортировка в данном фрагменте
* **ROWS** или **RANGE** – выражение для ограничения окна в пределах фрагмента

Сужение окна

* **UNBOUNDED PRECEDING –** окно не ограничено снизу
* **CURRENT ROW –** окно начинается с текущей строки
* **N PRECEDING** – окно заканчивается текущей строкой, начинается с N строки до текущей
* **N FOLLOWING** – окно начинается текущей строкой, заканчивается N строкой от текущей
* **UNBOUNDED FOLLOWING –** окно не ограничено сверху

Виды аналитических функций.

Агрегатные функции - возвращают значение, полученное путем арифметических вычислений: SUM(), AVG(), COUNT(), MIN(), MAX()

Функции ранжирования - позволяют получить порядковые номера записей в окне:

· RANK ( ) Присваивает ранг строкам, учитывая дубликаты.

· DENSE\_RANK ( ) Как RANK(), но без пропусков в рангах.

· ROW\_NUMBER ( ) Нумерует строки в пределах группы.

· NTILE ( ) Разбивает набор строк на n равных частей и присваивает каждой строке номер группы.

Функции сдвига - возвращают значение из другой строки окна

· LEAD( ) Получает значение из следующей строки.

· LAG ( ) Получает значение из предыдущей строки.

· FIRST\_VALUE( ) и LAST\_VALUE ( ) первое и последнее значение в окне

·RATIO\_TO\_REPORT ( ) вычисляет отношение значения текущей строки к сумме значений в определенной группе.

·Конструкции KEEP FIRST и KEEP LAST Эти конструкции используются для выбора первого или последнего значения в окне.

Аналитические функции - предоставляют информацию о распределении данных:

PERCENT\_RANK, CUME\_DIST, PERCENTILE\_CONT, PERCENTILE\_DISC

Синтаксис аналитических функций.

**Функции агрегирования:**

функция(column\_name) OVER

(PARTITION BY group\_column - разделение данных на окна

ORDER BY order\_column) – порядок строк в окне

SELECT date, amount,

SUM(amount) OVER (ORDER BY date) AS running\_total

FROM sales;

date | amount | running\_total

2023-01-01 | 100 | 100

2023-01-02 | 150 | 250

2023-01-03 | 200 | 450

2023-01-04 | 50 | 500

Для каждой строки вычисляется сумма всех предыдущих значений amount вместе с текущим, упорядоченных по дате.

SELECT department, employee, salary,

AVG(salary) OVER (PARTITION BY department) AS avg\_dept\_salary

FROM employees;

department | employee | salary | avg\_dept\_salary

-----------+----------+--------+---------------

IT | John | 5000 | 5500

IT | Anna | 6000 | 5500

HR | Mike | 4500 | 4750

HR | Sarah | 5000 | 4750

Для каждого отдела вычисляется средняя зарплата, которая добавляется к каждой строке сотрудника этого отдела.

### **Функции ранжирования:**

SELECT product, sales,

RANK() OVER (ORDER BY sales DESC) AS rank,

DENSE\_RANK() OVER (ORDER BY sales DESC) AS dense\_rank

FROM products;

product | sales | rank | dense\_rank

--------+-------+------+-----------

A | 100 | 1 | 1

B | 90 | 2 | 2

C | 90 | 2 | 2

D | 80 | 4 | 3

* RANK() пропускает следующий ранг после одинаковых значений (после двух 2-х мест идет 4-е)
* DENSE\_RANK() не пропускает ранги (после двух 2-х мест идет 3-е)

SELECT customer, purchase\_date, amount,

ROW\_NUMBER() OVER (PARTITION BY customer ORDER BY purchase\_date) AS purchase\_num

FROM purchases;

customer | purchase\_date | amount | purchase\_num

---------+---------------+--------+-------------

Alice | 2023-01-01 | 100 | 1

Alice | 2023-01-15 | 200 | 2

Bob | 2023-01-05 | 150 | 1

Bob | 2023-01-20 | 300 | 2

Для каждого покупателя нумеруются покупки в хронологическом порядке.

### **Функции анализа:**

SELECT month, revenue,

LAG(revenue, 1) OVER (ORDER BY month) AS prev\_month,

LEAD(revenue, 1, 0) OVER (ORDER BY month) AS next\_month

FROM monthly\_revenue;

month | revenue | prev\_month | next\_month

--------+---------+------------+-----------

Jan | 1000 | NULL | 1200

Feb | 1200 | 1000 | 1500

Mar | 1500 | 1200 | 0

LAG() возвращает значение из предыдущей строки (для января предыдущего значения нет - NULL)

LEAD() возвращает значение из следующей строки (для марта следующего значения нет - возвращает 0, указанный как default\_value)

SELECT date, stock, price,

FIRST\_VALUE(price) OVER (PARTITION BY stock ORDER BY date) AS first\_price,

LAST\_VALUE(price) OVER (PARTITION BY stock ORDER BY date

ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS last\_price

FROM stock\_prices;

date | stock | price | first\_price | last\_price

-----------+-------+-------+-------------+-----------

2023-01-01 | AAPL | 150 | 150 | 170

2023-01-02 | AAPL | 155 | 150 | 170

2023-01-03 | AAPL | 170 | 150 | 170

2023-01-01 | MSFT | 250 | 250 | 270

2023-01-02 | MSFT | 260 | 250 | 270

Для каждой акции показывает цену в первый и последний день периода. Обратите внимание на обязательное использование ROWS BETWEEN для LAST\_VALUE().

SELECT department, employee, salary,

RATIO\_TO\_REPORT(salary) OVER (PARTITION BY department) AS salary\_ratio

FROM employees;

department | employee | salary | salary\_ratio

-----------+----------+--------+-------------

IT | John | 5000 | 0.4545

IT | Anna | 6000 | 0.5455

HR | Mike | 4500 | 0.4737

HR | Sarah | 5000 | 0.5263

Показывает долю зарплаты каждого сотрудника от общей суммы зарплат по отделу (5000/(5000+6000) ≈ 0.4545).

SELECT student, test\_score,

NTILE(4) OVER (ORDER BY test\_score DESC) AS quartile

FROM test\_scores;

student | test\_score | quartile

--------+------------+---------

Alice | 95 | 1

Bob | 90 | 1

Carol | 85 | 2

Dave | 80 | 2

Eve | 75 | 3

Frank | 70 | 3

Grace | 65 | 4

Разбивает студентов на 4 группы (квартили) по убыванию баллов.

### **Конструкции KEEP FIRST и KEEP LAST:**

SELECT department,

MAX(salary) KEEP (DENSE\_RANK FIRST ORDER BY hire\_date) AS first\_hire\_salary

FROM employees

GROUP BY department;

department | first\_hire\_salary

-----------+------------------

IT | 4000

HR | 3500

Для каждого отдела находит зарплату сотрудника, который был принят раньше всех (FIRST по hire\_date). Если таких несколько (с одинаковой датой), берет максимальную зарплату из них. Или LAST если принят позже всех.

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

Преобразование данных в SQL. Применение оператора MERGE.

Используется для объединения данных из одной таблицы в другую. Позволяет выполнять операции вставки, обновления и удаления в зависимости от наличия данных.

Типичные задачи:

* Необходимость слияния старых и новых данных в случае их расхождения
* Заменяет INSERT и UPDATE
* Требует commit / rollback для фиксации/отката

|  |
| --- |
| MERGE INTO target\_table USING source\_table  ON (target\_table.key = source\_table.key)  WHEN MATCHED THEN  UPDATE SET target\_table.column = source\_table.column  WHEN NOT MATCHED THEN  INSERT (target\_table.column1, target\_table.column2)  VALUES (source\_table.column1, source\_table.column2); |

Применение операторов PIVOT и UNPIVOT.

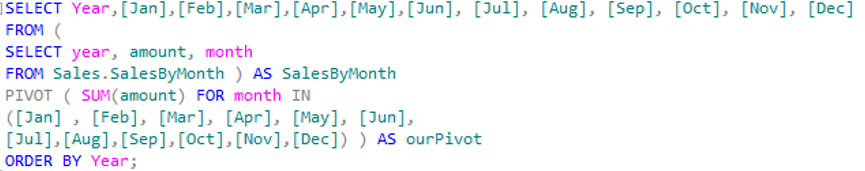
Pivot-Преобразует строки в столбцы, что позволяет более удобно представлять данные.

Типичные задачи:

-Необходимость получения сводных отчетов

-Подсчет итогов и промежуточных итогов

Выполняется обращение строк в столбцы



Ограничения

* Нельзя сформировать список столбцов подзапросом напрямую
* Для динамического PIVOT можно использовать:
  + Динамический SQL (собирая запрос как строку)
  + Временные таблицы

UNPIVOT-Преобразует столбцы в строки, полезен для нормализации данных.

|  |
| --- |
| SELECT \*  FROM (SELECT column1, month1, month2, month3 FROM table)  UNPIVOT (value\_column FOR month IN (month1 AS 'January', month2 AS 'February', month3 AS 'March')); |

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

**Объектные типы данных в SQL Server**

#### **1. Описание объектных типов данных**

Объектные типы данных в SQL Server позволяют создавать структуры данных, которые могут содержать как простые, так и сложные типы данных. Они обеспечивают возможность работы с более сложными данными в реляционных базах данных.

#### **2. Виды объектных типов данных**

* **User-defined types (UDTs)**: Пользовательские типы данных, которые можно создавать на основе существующих типов.

CREATE TYPE dbo.PhoneNumber FROM VARCHAR(15) NOT NULL;

CREATE TYPE dbo.EmailAddress FROM NVARCHAR(255) WITH VALIDATION (

VALUE LIKE '%\_@\_\_%.\_\_%'

);

CREATE TABLE Customers (

CustomerID INT PRIMARY KEY,

Phone dbo.PhoneNumber,

Email dbo.EmailAddress

);

**Преимущества:**

* Стандартизация форматов данных
* Встроенная валидация
* Улучшенная читаемость кода
* **Table types**: Позволяют создавать временные таблицы, которые могут использоваться в хранимых процедурах и функциях.

CREATE TYPE dbo.EmployeeTableType AS TABLE (

EmployeeID INT NOT NULL PRIMARY KEY,

FirstName NVARCHAR(50) NOT NULL,

LastName NVARCHAR(50) NOT NULL,

HireDate DATE NULL,

INDEX IX\_LastName NONCLUSTERED (LastName)

);

CREATE PROCEDURE dbo.ProcessEmployees

@Employees dbo.EmployeeTableType READONLY

AS

BEGIN

-- Обработка данных табличного параметра

SELECT \* FROM @Employees WHERE HireDate > '2020-01-01';

END;

**Особенности:**

* Передаются по ссылке, а не по значению
* Должны быть объявлены как READONLY
* Поддерживают индексы и ограничения
* **XML**: Тип данных для хранения XML-документов.

DECLARE @xmlData XML = '

<Employees>

<Employee ID="1">

<Name>John Doe</Name>

<Position>Developer</Position>

</Employee>

</Employees>';

-- Извлечение данных

SELECT

@xmlData.value('(/Employees/Employee/Name)[1]', 'NVARCHAR(100)') AS EmployeeName,

@xmlData.value('(/Employees/Employee/@ID)[1]', 'INT') AS EmployeeID;

-- Модификация данных

SET @xmlData.modify('

insert <Salary>5000</Salary>

as last into (/Employees/Employee)[1]

');

**Методы XML типа:**

* .query() - выполнение XQuery
* .value() - извлечение скалярного значения
* .exist() - проверка существования узла
* .modify() - изменение XML
* .nodes() - разбор на строки
* **JSON**: Хотя JSON не является отдельным типом данных, SQL Server поддерживает работу с JSON через функции.

**Особенности:**

* В SQL Server JSON хранится как обычный текст (NVARCHAR)
* Специальные функции для парсинга и генерации JSON

-- Преобразование строки в JSON объект

DECLARE @json NVARCHAR(MAX) = N'{"name":"John", "age":30, "city":"New York"}';

-- Извлечение значений

SELECT

JSON\_VALUE(@json, '$.name') AS Name,

JSON\_VALUE(@json, '$.age') AS Age,

JSON\_QUERY(@json, '$') AS FullJson;

-- Проверка валидности

SELECT ISJSON(@json) AS IsValidJson;

-- Генерация JSON из таблицы

SELECT

EmployeeID AS [id],

FirstName AS [name.first],

LastName AS [name.last]

FROM Employees

FOR JSON PATH, ROOT('employees');

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

**Сборки в SQL Server**

#### **1. Описание сборок**

Сборки (assemblies) в SQL Server представляют собой библиотеки, написанные на .NET, которые могут содержать код, типы и ресурсы. Они позволяют расширять функциональность SQL Server, используя возможности .NET.

#### **2. Применение сборок в SQL Server**

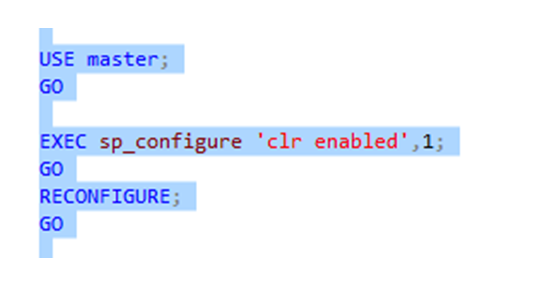
* Реализация сложной логики, которая не может быть эффективно выполнена с помощью T-SQL.
* Создание пользовательских функций, процедур и триггеров на .NET.
* Интеграция с другими приложениями на .NET.

#### **3. Виды программных конструкций, используемых в сборках**

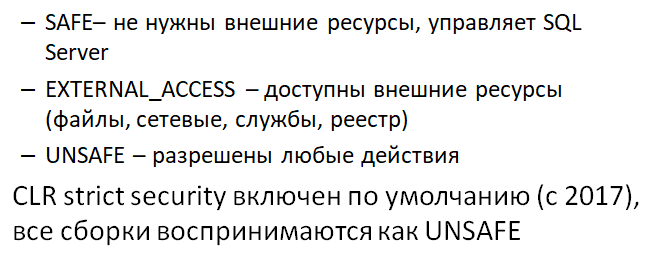
* **Функции**: Определяют методы, которые могут быть вызваны из SQL.
* **Процедуры**: Позволяют выполнять набор операций на сервере.
* **Триггеры**: Позволяют выполнять код при изменении данных в таблицах.
* **Типы**: Пользовательские типы, определенные в сборке.

#### **4. Регистрация сборок**

Сборки регистрируются в базе данных с помощью команды CREATE ASSEMBLY. Необходимо указать путь к сборке и, при необходимости, дополнительные параметры.

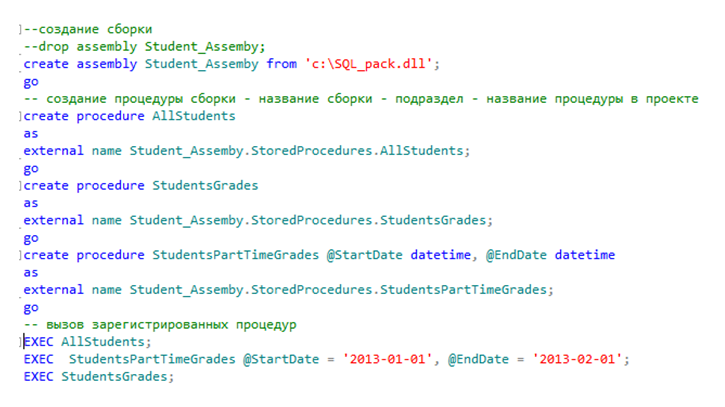


|  |
| --- |
| CREATE ASSEMBLY MyAssembly  FROM 'C:\Path\To\MyAssembly.dll'  WITH PERMISSION\_SET = SAFE; |



#### **5. Использование сборок**

После регистрации сборки можно создавать функции, процедуры и триггеры, которые используют код из сборки.



|  |
| --- |
| CREATE FUNCTION dbo.MyFunction(@param INT)  RETURNS INT  AS EXTERNAL NAME MyAssembly.[Namespace.ClassName].MethodName; |

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

**Пространственные данные и ГИС-приложения**

#### **Описание пространственных данных**

Пространственные данные представляют собой информацию, связанную с географическими объектами и их местоположением. Эти данные используются для описания форм, размеров и распределения объектов на поверхности Земли.

#### **ГИС-приложения**

Геоинформационные системы (ГИС) предназначены для сбора, хранения, анализа и визуализации пространственных данных. Примеры приложений:

* **ArcGIS**: Программное обеспечение для создания карт и анализа пространственных данных.
* **QGIS**: Открытая ГИС-платформа.
* **Google Earth**: Интерактивный инструмент для визуализации географической информации.

#### **Импорт пространственных данных из ГИС-систем**

1. **Загрузку данных** из источников (например, с сайта [freegisdata.rtwilson.com](https://freegisdata.rtwilson.com/)).
2. **Подключение к СУБД** (например, SQL Server или Oracle) из QGIS.
3. **Копирование данных** в базу данных с сохранением их пространственных свойств.
4. **Определение типа данных** (точка, линия, полигон и др.) и системы координат (SRID).

**Виды пространственных данных**

1. **Точки (Point)** — объекты с координатами (X, Y), например, метки на карте. Пример: местоположение автомобиля в аренде.
2. **Линии (LineString)** — последовательности точек, образующие линии. Пример: маршруты движения транспорта.
3. **Полигоны (Polygon)** — замкнутые области, например, границы городов. Пример: зоны покрытия арендных услуг.
4. **Мультиобъекты** (MultiPoint, MultiLineString, MultiPolygon) — группы однотипных объектов.

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

**Пространственные данные и SRID**

#### **1. Описание пространственных данных**

Пространственные данные представляют собой информацию, связанную с географическими объектами и их местоположением. Эти данные используются для описания форм, размеров и распределения объектов на поверхности Земли.

#### **2. SRID (Spatial Reference System Identifier)**

* **Определение**: SRID — это уникальный идентификатор, который определяет систему координат для пространственных данных. Он указывает, как интерпретировать координаты объектов.
* **Примеры**:
  + SRID 4326 соответствует системе WGS 84, которая используется в GPS.
  + SRID 3857 используется в веб-картах, таких как Google Maps.

#### **3. Методы и свойства пространственных данных**

* **Методы**:
* STIntersection — проверяет пересечение объектов.  
  Пример: определение, пересекаются ли два полигона.  
  STUnion — объединяет объекты в один.  
  STWithin — проверяет, содержится ли один объект внутри другого.  
  STDistance — вычисляет расстояние между объектами.  
  STArea — вычисляет площадь полигона.  
  STLength — вычисляет длину линии.
* **Свойства**:

**GeometryType**: Определяет тип геометрии (точка, линия, полигон).

**SRID**: Возвращает SRID объекта, указывающий на его систему координат.

#### **4. Индексирование пространственных данных**

* Индексирование пространственных данных позволяет ускорить запросы и операции, связанные с пространственными объектами
* **Типы индексов**:
* **R-дерево (R-tree)** — индекс для быстрого поиска объектов в пределах заданной области.
* **Quadtree** — разбивает пространство на квадраты для оптимизации запросов.

CREATE SPATIAL INDEX index\_name ON table\_name (geometry\_column);

Индексирование существенно улучшает производительность при выполнении пространственных запросов, таких как поиск объектов в пределах заданной области.

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

Объектные типы данных в Oracle.

Объектные типы позволяют создать составной тип данных с атрибутами и методами.

CREATE TYPE PersonType AS OBJECT (

id NUMBER,

name VARCHAR2(100),

MEMBER FUNCTION get\_name RETURN VARCHAR2

);

Атрибуты.

Атрибуты — поля объекта (аналог переменных-членов).

• Методы — функции и процедуры, определённые внутри объекта. • Типы методов:

o MEMBER: работает с конкретным экземпляром

o STATIC: работает на уровне класса

Тип может содержать методы:

* Методы member
* Методы конструкторы
* Статические методы
* Методы сравнения

Заголовок и тело метода компилируются отдельно

CREATE TYPE BODY PersonType AS

MEMBER FUNCTION get\_name RETURN VARCHAR2 IS

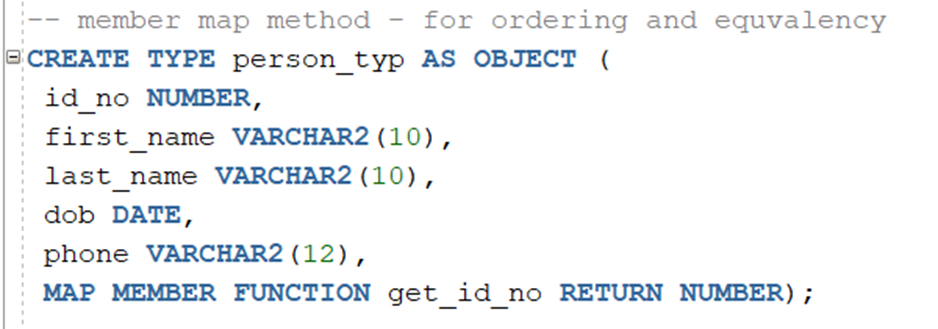
BEGIN

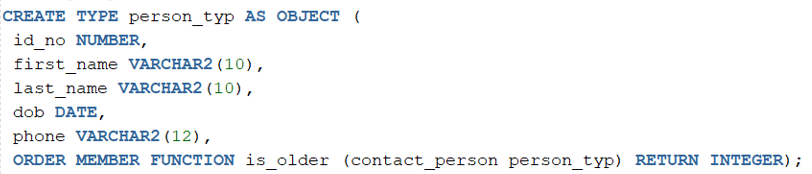
RETURN name;

END;

END;

•Методы MAP – предназначены для сравнения, сортировки и UNION



•Методы ORDER – предназначены для сортировки по значениям полей

•Статические методы относятся к типу в целом, а не к экземпляру

Объектные таблицы — таблица, строки которой представляют экземпляры объекта.

•DML операции в объектных таблицах – аналогично стандартным

CREATE TABLE persons OF PersonType; Позволяет обращаться к атрибутам: SELECT p.name FROM persons p;

Объектные представления — это view, основанное на объектных типах, которое отображает строковые таблицы как объектные.

CREATE VIEW person\_view OF PersonType

WITH OBJECT IDENTIFIER (id) -- указываем первичный ключ

AS

SELECT id, name FROM person\_table; -- исходная реляционная таблица

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:  
CREATE INDEX idx\_person\_name ON persons(name);  
Индекс по вложенному объекту:  
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](https://web.telegram.org/a/address.city));  
**2. Функциональные индексы (на метод)**  
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 c выражением FORALL.

Коллекции в Oracle.

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

Виды коллекций.

**Ассоциативные массивы (INDEX-BY таблицы)**

* Одномерные коллекции с гибкой индексацией (числовой или строковый ключ).
* Не сохраняются в базе данных.
* Могут быть разреженными (иметь "пропуски" в индексах).
* Пример:

TYPE emp\_table IS TABLE OF emp%ROWTYPE INDEX BY PLS\_INTEGER;

**Вложенные таблицы (Nested Tables)**

* + Одномерные коллекции без фиксированного размера.
  + Могут быть плотными (все элементы подряд) или разреженными (если элементы удаляются).
  + Могут храниться в базе данных.
  + Пример:

TYPE name\_table IS TABLE OF VARCHAR2(100);

**VARRAY (Variable-Size Arrays)**

* + Коллекции с фиксированным максимальным размером.
  + Всегда плотные (элементы идут подряд).
  + Могут храниться в базе данных.
  + Пример:

TYPE grades\_array IS VARRAY(5) OF NUMBER;

Методы и исключения коллекций.

•count – подсчет количества элементов коллекции

•delete – удаление элементов без сжатия коллекции

•trim (n) – удаление элементов с сжатием коллекции (для nested table/VARRAY)

•extend – расширение коллекции (для nested table/VARRAY)

•first / last – первый /последний индекс коллекции

•exists(i) – проверка существования i элемента

•prior / next – предыдущий /следующий индекс коллекции

Исключения могут возникать при работе с коллекциями, такие как:

* NO\_DATA\_FOUND: Возникает, если запрашиваемый элемент отсутствует.
* SUBSCRIPT\_OUTSIDE\_LIMIT: Возникает при доступе к элементу с индексом, который выходит за пределы коллекции.

Множественная обработка записей c выражением FORALL

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

TYPE id\_table IS TABLE OF employees.employee\_id%TYPE;

v\_ids id\_table := id\_table(100, 101, 102);

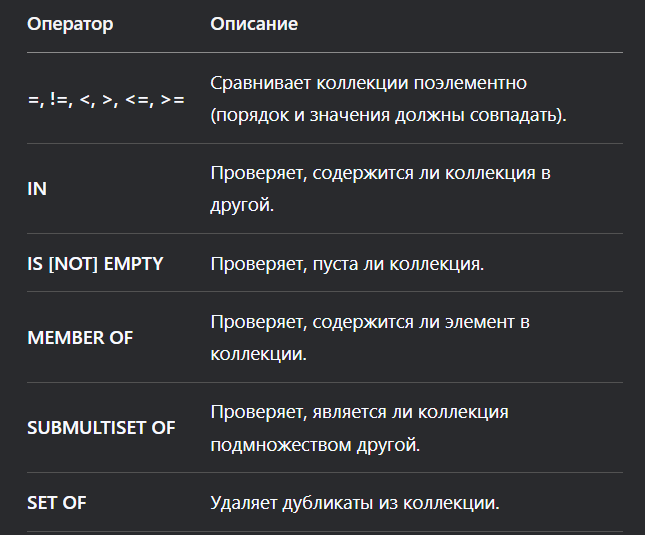
FORALL i IN 1..v\_ids.COUNT

DELETE FROM employees WHERE employee\_id = v\_ids(i);

20. Коллекции в Oracle. Сравнение коллекций. MULTISET.

Коллекции в Oracle ПРОШЛЫЙ ВОПРОС

Сравнение коллекций



MULTISET

MULTISET: Оператор MULTISET используется для сравнения коллекций.

-- Объединение коллекций

SELECT \* FROM (SELECT 'A', 'B', 'C' FROM dual) MULTISET UNION (SELECT 'B', 'D' FROM dual);

-- Пересечение коллекций

SELECT \* FROM (SELECT 'A', 'B', 'C' FROM dual) MULTISET INTERSECT (SELECT 'B', 'D' FROM dual);

-- Вычитание коллекций

SELECT \* FROM (SELECT 'A', 'B', 'C' FROM dual) MULTISET EXCEPT (SELECT 'B', 'D' FROM dual);

**Ключевые особенности**

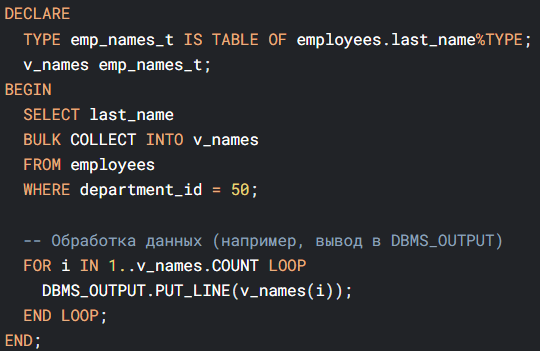
* MULTISET работает только с **вложенными таблицами** и **VARRAY**.
* **Ассоциативные массивы** нельзя сравнивать напрямую.
* Для сравнения порядок элементов **важен** (если не используется SET OF для удаления дублей).

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

Коллекции в Oracle ПРОШЛЫЙ ВОПРОС

Множественная обработка записей c выражением BULK COLLECT.

**BULK COLLECT** — это механизм, который позволяет **массово загружать данные** из SQL-запроса в PL/SQL-коллекции, **минимизируя переключения контекста** между SQL и PL/SQL. Это значительно ускоряет обработку больших наборов данных.

  
**Можно использовать и с курсорами:**  
OPEN c;  
FETCH c BULK COLLECT INTO v\_collection;  
CLOSE c;  
**BULK COLLECT с LIMIT**  
Для обработки больших объёмов данных можно использовать LIMIT:  
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.

**Динамический SQL** — это способ выполнения SQL-запросов, которые формируются **во время выполнения программы** (runtime), а не на этапе компиляции.

**1. Области применения**

* Запросы с **неизвестной структурой** (например, имена таблиц или условия фильтрации задаются пользователем).
* **Гибкие DML/DDL-операции** (создание таблиц, обновление данных с переменными условиями).
* **Администрирование БД** (динамическая генерация скриптов).

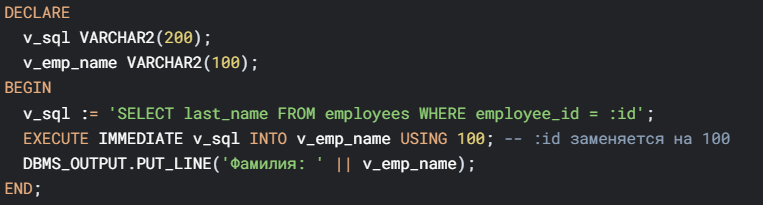
**2. Динамический SQL в Oracle (PL/SQL)**

**Основной метод:**EXECUTE IMMEDIATE

Выполняет SQL-запрос, переданный в виде строки.

**Примеры:**

**1. Простой запрос (SELECT)**



**2. DML (INSERT/UPDATE/DELETE)**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

**3. DDL (CREATE, DROP, ALTER)**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

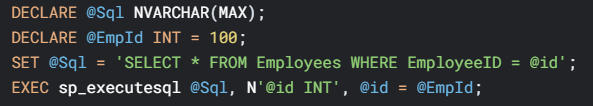
**4. Многострочный запрос (OPEN-FOR)  
Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**3. Динамический SQL в SQL Server (T-SQL)**

**Основные методы:**

* **EXEC / EXECUTE — простой запуск SQL-строки.**
* **sp\_executesql — более безопасный способ (поддержка параметров и кеширования планов).**

****

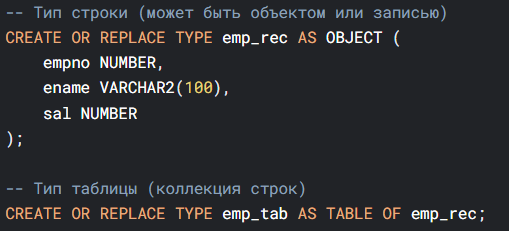
23. Программные конструкции PL/SQL. Табличные функции.

Табличные функции

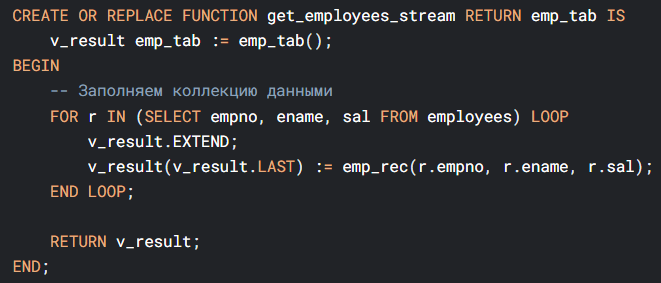
Табличные функции - это специальные функции PL/SQL, которые возвращают набор данных в виде таблицы, что позволяет использовать их непосредственно в SQL-запросах.

Основные виды табличных функций

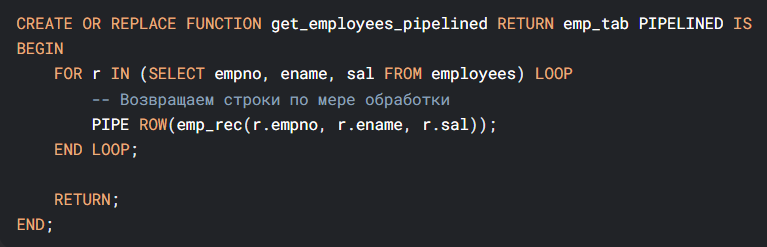
1. **Потоковые (неконвейерные) функции**
   * Возвращают полный набор данных сразу
   * Все результаты загружаются в память перед использованием
   * Проще в реализации, но менее эффективны для больших данных
2. **Конвейерные (PIPELINED) функции**
   * Возвращают данные по мере их обработки (построчно)
   * Более эффективны для больших объемов данных
   * Используют ключевое слово PIPE ROW

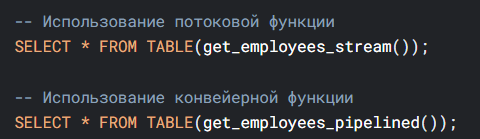


Потоковая



Конвейерная





24. Программные конструкции PL/SQL. Конвейерные функции.

**Конвейерные функции в PL/SQL**

Конвейерные функции - 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.

Основные особенности конвейерных функций

1. **Построчная обработка** - данные возвращаются немедленно с помощью PIPE ROW
2. **Эффективность памяти** - не требуется хранить весь результат в памяти
3. **Ранний доступ к данным** - клиент может начать обработку до завершения функции
4. **Использование в SQL** - могут вызываться в FROM-части SQL-запросов

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

**Пакеты** - контейнеры, которые группируют связанные процедуры, функции, переменные и другие элементы. Они позволяют организовать код, улучшить его читаемость и повторное использование.

Пакет состоит из двух частей:

1. **Спецификация (PACKAGE)** - публичный интерфейс
   * Объявления процедур, функций
   * Объявления переменных, констант, типов
   * Объявления курсоров
   * Объявления исключений
2. **Тело пакета (PACKAGE BODY)** - реализация
   * Код процедур и функций
   * Приватные элементы (невидимые извне)

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Ключевые особенности пакетов

1. Пакетные переменные

* Сохраняют состояние между вызовами в рамках сессии
* Доступны всем процедурам/функциям пакета
* Инициализируются в секции инициализации или при объявлении

2. Пакетные курсоры

* Могут быть объявлены в спецификации или теле пакета

4. Секция инициализации

* Выполняется один раз при первом обращении к пакету
* Используется для инициализации переменных, кэширования данных

Использование пакета

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Преимущества пакетов

1. **Инкапсуляция** - скрытие реализации
2. **Производительность** - весь пакет загружается в память сразу
3. **Управление зависимостями** - меньше перекомпиляций
4. **Организация кода** - логическая группировка функционала
5. **Сохранение состояния** - переменные сохраняют значения между вызовами

26. Технологии высокой доступности.

**Технологии высокой доступности**

Высокая доступность (HA) обеспечивает непрерывную работу систем и приложений, минимизируя простои и обеспечивая быструю реакцию на сбои.

#### **Уменьшение случаев внеплановых простоев**

* **Кластеризация (RAC/RAC One)**: Объединяет несколько серверов, позволяя им работать как единое целое. Если один узел выходит из строя, другие продолжают работу.
* **Глобальные данные**: Используются для обеспечения отказоустойчивости и балансировки нагрузки.

#### **Быстрое восстановление после сбоев**

* **Active Data Guard**: Обеспечивает защиту данных и быстрое восстановление после сбоев, минимизируя время простоя.
* **RMAN и GoldenGate**: Инструменты для резервного копирования и восстановления, которые позволяют быстро восстановить данные и обеспечить их целостность.

#### **Сокращение планируемых простоев**

* **Edition-based Redefinition**: Позволяет вносить изменения в схемы и объекты базы данных без остановки работы приложения.
* **Flashback**: Позволяет восстановить данные до состояния на определенный момент времени, что снижает время простоя при возникновении ошибок.

**Основные принципы и технологии:**

**Избыточность**: Использование дублированных компонентов, таких как сервера, сети и хранилища данных, чтобы в случае отказа одного из них, работа могла продолжаться на другом.

**Отказоустойчивость**: Проектирование системы так, чтобы она могла выдерживать и быстро восстанавливаться после сбоев, например, переключение на резервный сервер или база данных.

**Кластеризация**: Соединение нескольких серверов в единую систему, чтобы они могли работать вместе и обеспечивать высокую доступность.

**Автоматическое восстановление**: Использование механизмов, которые автоматически восстанавливают систему в рабочее состояние после сбоя.

**Репликация данных**: Сохранение копий данных на нескольких серверах, что позволяет продолжать работу даже в случае отказа одного из хранилищ.

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**  
*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)

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  
SELECT CURRENT\_SCN FROM V$DATABASE;

28. Задания. Системные пакеты DBMS\_JOB и DBMS\_SCHEDULER в Oracle.

Oracle предоставляет два основных пакета для управления заданиями и планирования: **DBMS\_JOB** и **DBMS\_SCHEDULER**. Эти пакеты позволяют автоматизировать выполнение задач и управлять ими в базе данных.

#### **1. DBMS\_JOB** DBMS\_JOB позволяет создавать, управлять и контролировать задания, которые выполняются в фоновом режиме.

**Основные функции**:

* **Создание задания**: Позволяет планировать выполнение PL/SQL кода.
* **Управление заданиями**: Включает в себя функции для активации, деактивации и удаления заданий.

DECLARE

job\_number NUMBER;

BEGIN

DBMS\_JOB.SUBMIT(job\_number, 'my\_procedure;', sysdate + 1/24); -- Запланировать выполнение через 1 час

COMMIT;

END;

**Основные процедуры**:

* DBMS\_JOB.SUBMIT: Создает новое задание.
* DBMS\_JOB.REMOVE: Удаляет задание.
* DBMS\_JOB.RUN: Запускает задание немедленно.
* DBMS\_JOB.BROKEN: Устанавливает флаг для задания, чтобы оно не выполнялось.

#### **2. DBMS\_SCHEDULER** DBMS\_SCHEDULER — более современный и мощный пакет для планирования задач по сравнению с DBMS\_JOB. Он поддерживает более сложные сценарии планирования и управления заданиями.

**Основные функции**:

* **Создание заданий**: Поддерживает различные типы заданий, включая выполнение внешних программ.
* **Управление заданиями**: Позволяет более гибко управлять заданиями, включая расписания и условия их выполнения.

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB (

job\_name => 'my\_job',

job\_type => 'PLSQL\_BLOCK',

job\_action => 'BEGIN my\_procedure; END;',

start\_date => SYSTIMESTAMP + INTERVAL '1' HOUR,

repeat\_interval => 'FREQ=DAILY; BYHOUR=10; BYMINUTE=0',

enabled => TRUE

);

END;

**Основные процедуры**:

* DBMS\_SCHEDULER.CREATE\_JOB: Создает новое задание.
* DBMS\_SCHEDULER.DROP\_JOB: Удаляет задание.
* DBMS\_SCHEDULER.ENABLE: Включает задание.
* DBMS\_SCHEDULER.DISABLE: Отключает задание.

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

### **Резервное копирование и восстановление в SQL Server**

Резервное копирование — это процесс создания копий базы данных, который позволяет восстановить данные в случае их потери или повреждения. В SQL Server резервное копирование может быть полным, дифференциальным или транзакционным.

#### **Виды резервного копирования**

* **Полное резервное копирование**:
  + Создает полную копию базы данных. Это основной метод резервного копирования.   
     BACKUP DATABASE [YourDatabase] TO DISK = 'C:\Backups\YourDatabase.bak';
* **Дифференциальное резервное копирование**:
  + Сохраняет изменения, сделанные после последнего полного резервного копирования.   
     BACKUP DATABASE [YourDatabase] TO DISK = 'C:\Backups\YourDatabase\_diff.bak' WITH DIFFERENTIAL;
* **Транзакционное резервное копирование**:  
  Сохраняет изменения, сделанные с момента последнего транзакционного резервного копирования. Это позволяет восстанавливать базу данных до конкретного момента времени.  
   BACKUP LOG [YourDatabase] TO DISK = 'C:\Backups\YourDatabase\_log.trn';

#### **Применение резервного копирования**

* **Защита данных**.
* **Восстановление базы данных**.
* **Планирование резервного копирования**.

#### **Восстановление базы данных**

* Восстановление базы данных может быть полным или частичным.

Пример команды для полного восстановления:

RESTORE DATABASE [YourDatabase] FROM DISK 'C:\Backups\YourDatabase.bak';

Если требуется восстановление с использованием дифференциальных или транзакционных резервных копий, необходимо учитывать порядок их применения.

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

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

### **Резервное копирование и восстановление в Oracle**

Резервное копирование в Oracle — это процесс создания копий данных, который позволяет защитить базу данных от потери информации из-за сбоев, ошибок или других проблем. Oracle предоставляет несколько методов резервного копирования, включая встроенные инструменты и утилиты.

#### **Виды резервного копирования**

* **Полное резервное копирование**:
  + Создает копию всей базы данных, включая все данные и структуру. Обычно выполняется в начале.
* **Инкрементное резервное копирование**:
  + Сохраняет только изменения, сделанные с момента последнего резервного копирования. Это может быть:
    - **Дельта**: изменения с последнего полного резервного копирования.
    - **Логическое**: изменения, сделанные с последнего инкрементного резервного копирования.
* **Резервное копирование по схемам**:  
  Позволяет создавать резервные копии отдельных схем (пользовательских объектов) в базе данных.

#### **Применение резервного копирования**

* **Защита данных**: Резервное копирование обеспечивает защиту данных от потерь, вызванных аппаратными или программными сбоями.
* **Восстановление базы данных**: Позволяет восстановить базу данных до состояния на момент резервного копирования.
* **Планирование резервного копирования**: Рекомендуется настраивать регулярные резервные копирования для минимизации потерь данных.

#### **Инструменты для резервного копирования**

* **RMAN (Recovery Manager)**:
  + Утилита для резервного копирования и восстановления баз данных Oracle. Поддерживает полное, инкрементное и дельта резервное копирование.

Пример команды для полного резервного копирования:  
 RMAN> BACKUP DATABASE;

* **Oracle Data Pump**:
  + Используется для экспорта и импорта данных, может быть применен для создания резервных копий объектов базы данных.

#### **Восстановление базы данных**

* Восстановление может быть полным или частичным, в зависимости от типа резервного копирования.

Пример команды для восстановления с использованием RMAN:  
 RMAN> RESTORE DATABASE;

RMAN> RECOVER DATABASE;

31. Репликация. Участники репликации. Настройка репликации.

**Репликация** — это механизм, с помощью которого SQL Server (или Oracle) **синхронизирует данные между несколькими базами данных**. Цель — **обеспечить доступ к актуальной информации** на разных серверах, либо для распределения нагрузки, либо для работы в офлайн-режиме.

Репликация может:

· выполняться **непрерывно** или **по расписанию**;

· **копировать изменения** (а не всю БД);

· использоваться в **распределённых системах**, географически удалённых.

**Когда не требуется репликация?**

· Разовое копирование БД (например, миграция).

· Импорт/экспорт данных с преобразованием (ETL, выгрузка в Excel и т.д.).

**Когда требуется репликация?**

· Есть **центральный сервер**, и нужно синхронизировать с филиалами.

· Требуется **перенос нагрузки** между несколькими серверами.

· Необходимо **распространять данные** на множество узлов.

· Требуется **двусторонняя синхронизация** (в случае Merge репликации).

**Участники репликации:**

1. **Издатель (Publisher)**:

o Источник данных.

o Отслеживает изменения.

o Может быть один на каждый набор данных.

2. **Дистрибьютор (Distributor)**:

o Отвечает за **передачу данных подписчикам**.

o Хранит данные в БД распространения (distribution database).

o Может быть на том же сервере, что и издатель, либо на отдельном.

3. **Подписчик (Subscriber)**:

o Получает и сохраняет данные.

o Может быть пассивным (только читает) или активным (вносит изменения — при Merge).

4. **Публикация (Publication)**:

o Набор данных (таблиц, представлений, процедур и т.д.), доступных для подписчиков.

5. **Статья (Article)**:

o Минимальная единица публикации (например, таблица, её строки, представление).

**Настройка репликации — этапы:**

1. **Выбор топологии и типа репликации**.

2. **Подготовка**:

o Настроить дистрибьютора.

o Создать публикационную БД.

o Указать издателя и дистрибьютора.

3. **Создание публикации**:

o Указываются статьи (таблицы, представления).

4. **Создание подписок**:

o Назначаются подписчики.

o Выбирается режим: *Push* (запуск у дистрибьютора) или *Pull* (у подписчика).

5. **Настройка агентов репликации**:

o Управляют выполнением операций.

6. **Обеспечение безопасности и прав доступа**.

7. **Мониторинг**: Используется репликационный монитор (Replication Monitor).

8. **Тестирование и обслуживание**.

**Агенты репликации:**

|  |  |
| --- | --- |
| **Агент** | **Назначение** |
| **Snapshot Agent** | Создаёт снимок данных и схемы. |
| **Distribution Agent** | Рассылает данные подписчикам. |
| **Merge Agent** | Обеспечивает слияние изменений. |
| **Log Reader Agent** | Считывает журнал транзакций издателя (только для transactional). |
| **Queue Reader Agent** | Обрабатывает очередь при асинхронной передаче изменений. |

32. Репликация. Топологии репликации. Виды репликации.

**Виды репликации:**

**1.** **Snapshot (репликация снимков):**

o **Что делает:** Создаёт **полную копию данных** на момент снимка.

o **Особенности:**

§ Простота.

§ Подходит для **нечасто меняющихся** данных.

§ Данные на подписчике могут **устаревать между снимками**.

o **Пример:** Список товаров, обновляющийся 1 раз в день.

**2.** **Transactional (транзакционная репликация):**

o **Что делает:** Сначала создаёт снимок, затем **реплицирует все изменения в реальном времени**.

o **Особенности:**

§ Использует **журнал транзакций**.

§ Высокая производительность.

§ Требует **первичных ключей**.

o **Пример:** Финансовые транзакции, которые должны быть согласованы между системами.

**3.** **Merge (слияние):**

o **Что делает:** Обе стороны **могут изменять данные**. При синхронизации выполняется **слияние с разрешением конфликтов**.

o **Особенности:**

§ Идеально для **мобильных пользователей**, **офлайн-режима**.

§ Поддержка конфликтов, обработка строк.

o **Пример:** Мобильные приложения, где пользователь может редактировать данные офлайн, а затем синхронизироваться.

**Топологии репликации:**

**1.** **Центральный издатель:**

o Один **издатель**, несколько подписчиков.

o Издатель и дистрибьютор могут быть **на одном сервере**.

o **Простой** в управлении, но может быть **узким местом по нагрузке**.

**2.** **Центральный издатель + удалённый дистрибьютор:**

o Позволяет **снизить нагрузку на издателя**, переместив дистрибуцию на отдельный сервер.

o Требует дополнительного обслуживания.

**3.** **Центральный подписчик:**

o Несколько **издателей** отправляют данные **одному подписчику**.

o Используется для **агрегации** данных с разных источников.

**4.** **Издающий подписчик (Publishing Subscriber):**

o Подписчик сам становится **издателем**.

o Позволяет **распространять изменения дальше**.

o Подходит для **цепочек распределения**.

**5.** **Равноправные участники (Peer-to-peer):**

o Все узлы **равны** — могут быть и издателями, и подписчиками.

o **Высокая сложность администрирования**, но **максимальная отказоустойчивость**.

o Требует точного управления конфликтами.

**Планирование и подготовка**

Для разных типов репликации — свои требования:

|  |  |
| --- | --- |
| **Тип** | **Особенности подготовки** |
| **Snapshot** | Требует место для файлов снимков, планирования расписания. |
| **Transactional** | Требует **первичных ключей**, мониторинга накопленных транзакций. |
| **Merge** | Требует первичных ключей + обработку **внешних ключей**, **timestamp**. |

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

Кэширование - механизм хранения часто используемых данных в оперативной памяти для ускорения доступа. Основные цели:

* Уменьшение количества физических операций ввода-вывода
* Повышение производительности запросов
* Оптимизация использования ресурсов сервера

**Типы кэширования**:

* **Кэширование запросов**: Хранение результатов часто выполняемых запросов.
* **Кэширование данных**: Хранение часто используемых данных в оперативной памяти для быстрого доступа.

### **Кэширование в Oracle**

#### **Buffer Cache** - основной кэш для хранения блоков данных.

ALTER SYSTEM SET db\_cache\_size=4G SCOPE=BOTH;

* **Shared Pool -** хранит SQL-запросы, PL/SQL-код и метаданные.
* **Result Cache -** кэширование результатов запросов.
* **Мониторинг**

### **Кэширование в MS SQL Server**

#### **Buffer Pool -** Основной кэш данных.

#### **Plan Cache -** Кэш планов выполнения.

#### **Query Store -** Механизм для хранения истории выполнения запросов.

## **Журналирование (WAL - Write Ahead Logging)**

**Журналирование** — это процесс записи всех изменений, происходящих в базе данных, для обеспечения возможности восстановления данных в случае сбоя.

Журналирование обеспечивает:

* Надежность хранения данных
* Возможность восстановления после сбоев
* Поддержку транзакционной целостности

**Типы журналов**:

* **Журнал транзакций**: Хранит информацию о всех транзакциях, включая изменения и их последовательность.
* **Журнал изменений**: Записывает изменения в структуре базы данных и объектах.

## **Параллельный доступ**

Механизмы для обеспечения:

* Одновременного доступа многих пользователей
* Поддержания целостности данных
* Минимизации блокировок

## **Оптимизация производительности**

### **В Oracle**

-- Настройка параллельного выполнения

ALTER TABLE sales PARALLEL 4;

-- Сбор статистики

EXEC DBMS\_STATS.GATHER\_TABLE\_STATS('SCHEMA', 'TABLE');

### **В MS SQL Server**

-- Настройка степени параллелизма

EXEC sp\_configure 'max degree of parallelism', 4;

RECONFIGURE;

-- Обновление статистики

UPDATE STATISTICS Sales.Orders WITH FULLSCAN;

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

•Бизнес-аналитика – business intelligence – BI – область технологии баз данных

•Процесс интегрирования всех необходимых данных в один источник данных, к которому конечные пользователи могут осуществлять нерегламентированные запросы для анализа этих данных

•Цель: принятие лучших бизнес решений

•Хранилище данных – база данных всех данных организации, к которым пользователи могут иметь единообразный доступ

•Проблемы:

–большой объем данных

–получены в разное время

–хранятся в разных базах данных

Основы бизнес-аналитики:

Цель:

Помогать компаниям принимать решения, основанные на данных, для улучшения производительности и прибыльности.

Основы:

Задействование различных инструментов анализа данных, включая отчетность, информационные панели, анализ данных, моделирование и прогнозирование.

Польза:

Даёт возможность увидеть тенденции и закономерности в данных, чтобы принимать более обоснованные решения.

Хранилища данных (Data Warehouse):

Определение:

Системы управления данными, предназначенные для анализа, хранения и обработки больших объемов данных.

Особенности:

Организация данных для удобства запросов и анализа, часто содержат исторические данные, которые можно использовать для сравнения.

Предназначение:

Служат основой для бизнес-аналитики, обеспечивая возможность выполнять сложные запросы и сравнивать различные наборы данных для принятия стратегических решений.

Киоски данных (Data Marts):

Определение:

Часть хранилища данных, предназначенная для определенной области бизнес-аналитики.

Особенности:

Содержат данные, необходимые для анализа конкретной тематики, часто менее объемные, чем хранилище данных.

Предназначение:

Снижают сложность анализа данных для конкретных пользователей или подразделений.

Проектирование хранилищ данных:

Этапы:

Определение потребностей: Анализ требований и целей бизнес-аналитики.

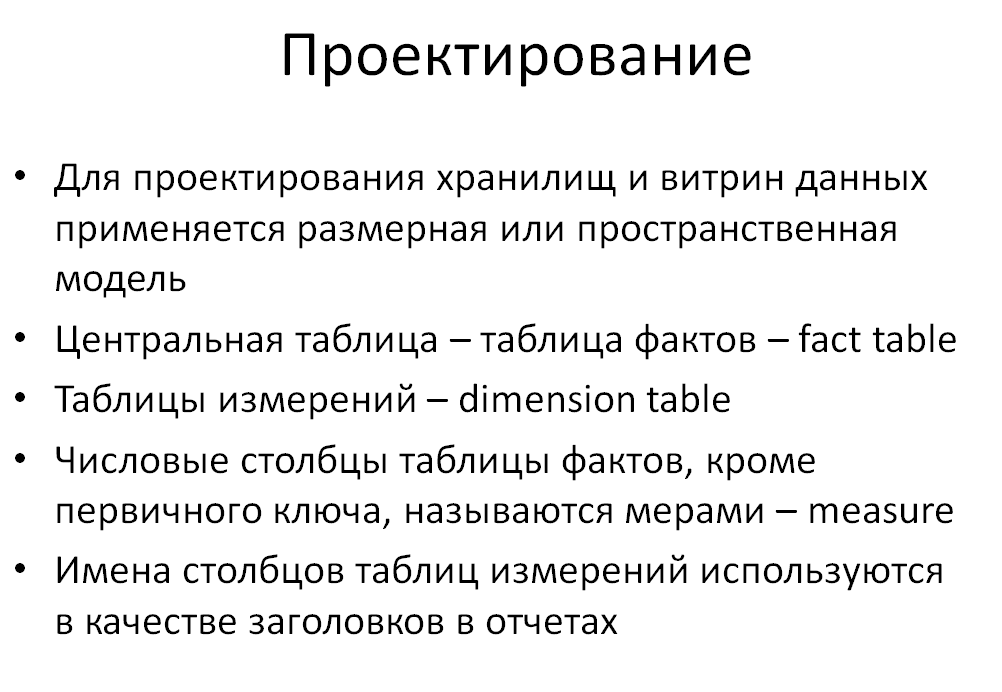
Анализ данных: Определение источников данных, структуры и объема данных.

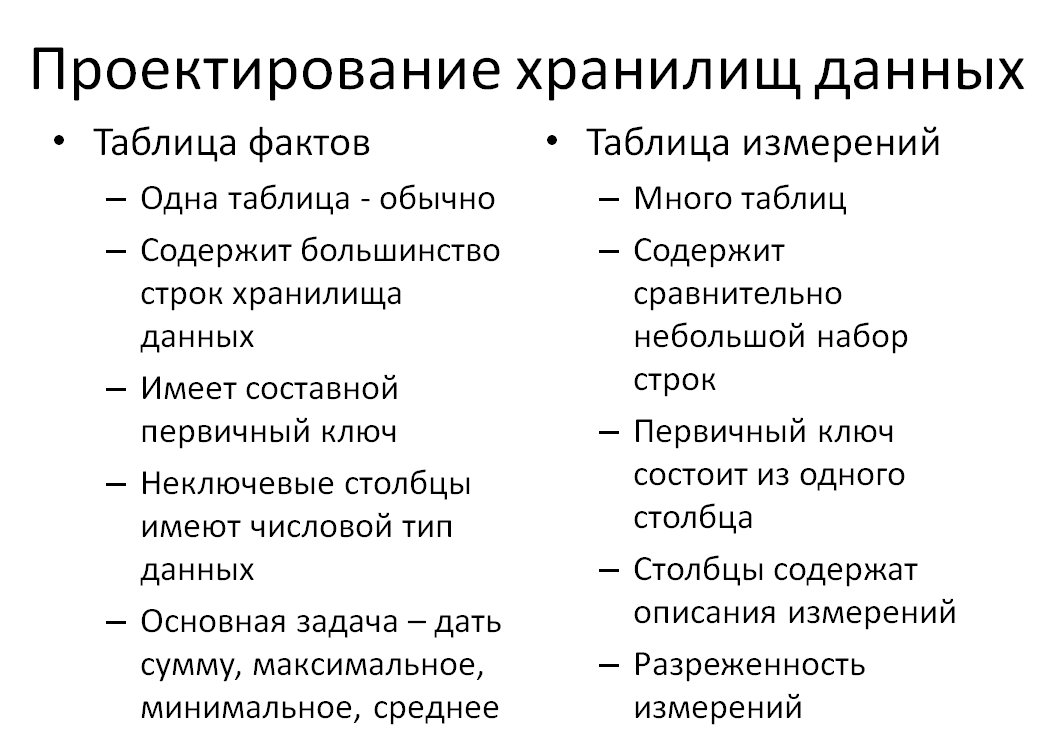
Планирование структуры: Создание схемы хранения данных, оптимизированной для запросов и анализа.

Разработка ETL (Extract, Transform, Load): Планирование процесса извлечения, преобразования и загрузки данных в хранилище.

Оптимизация: Подбор подходящих технологий и инструментов для хранения и анализа данных.

Тестирование и внедрение: Проверка работоспособности и внедрение хранилища данных в рабочую среду.

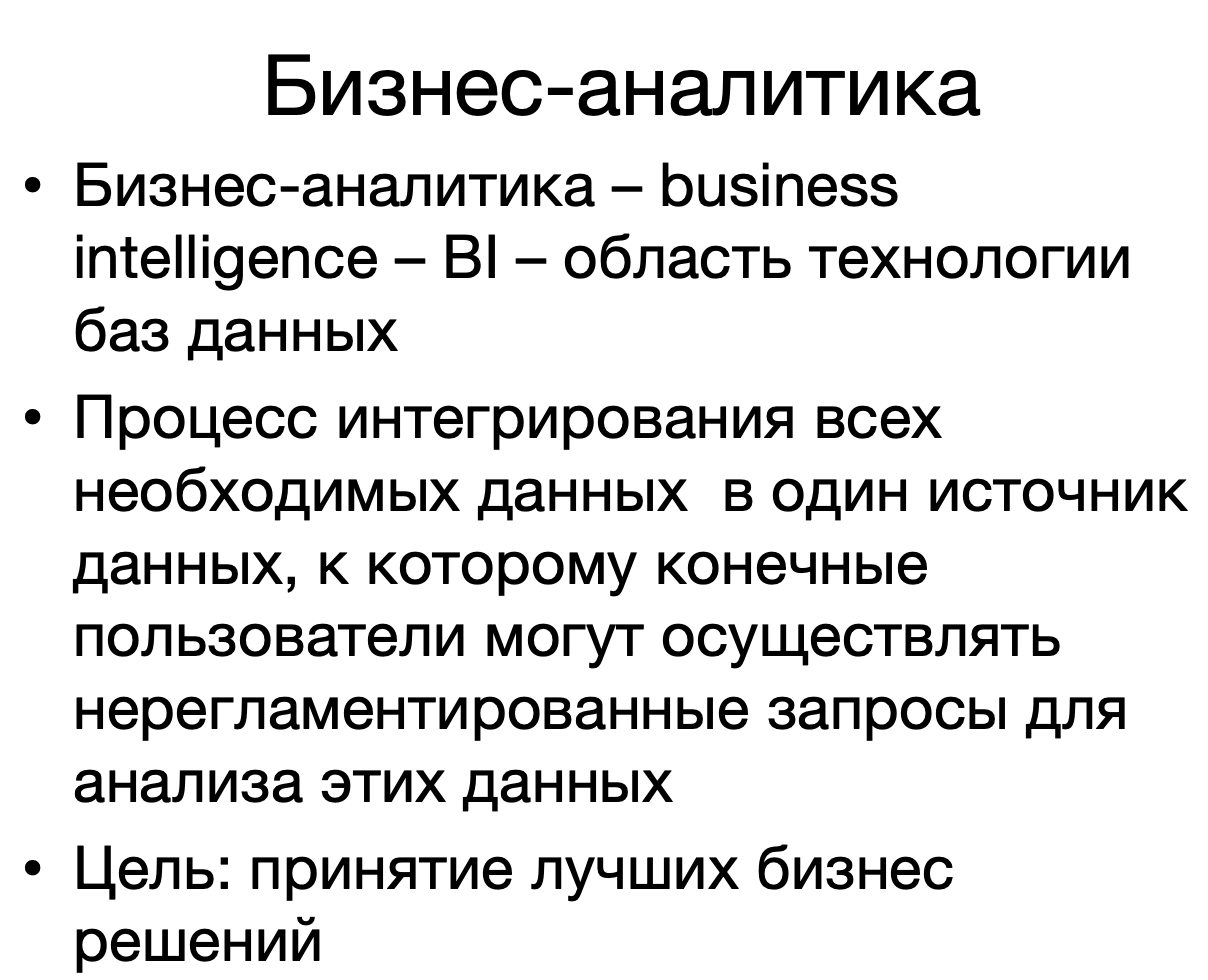


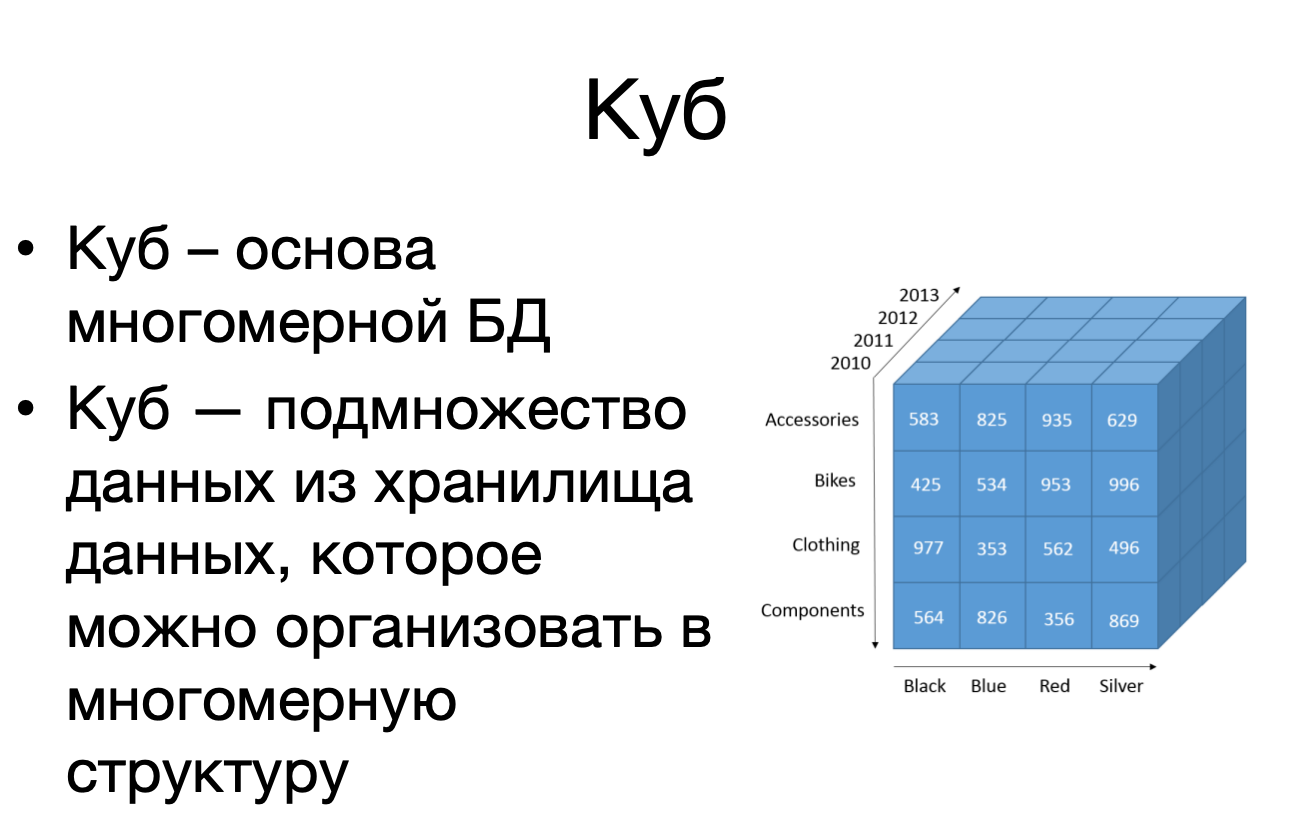
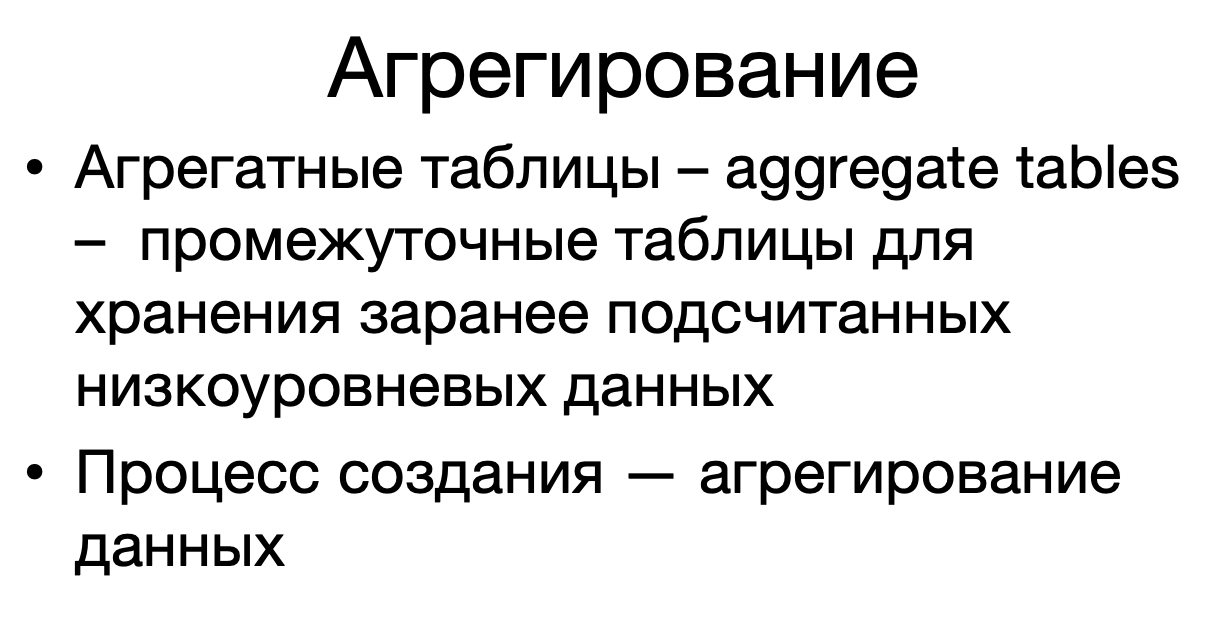






35. Основы бизнес-аналитики. Кубы и их архитектура. Агрегирование. Уровень агрегирования. Физическое хранение куба.





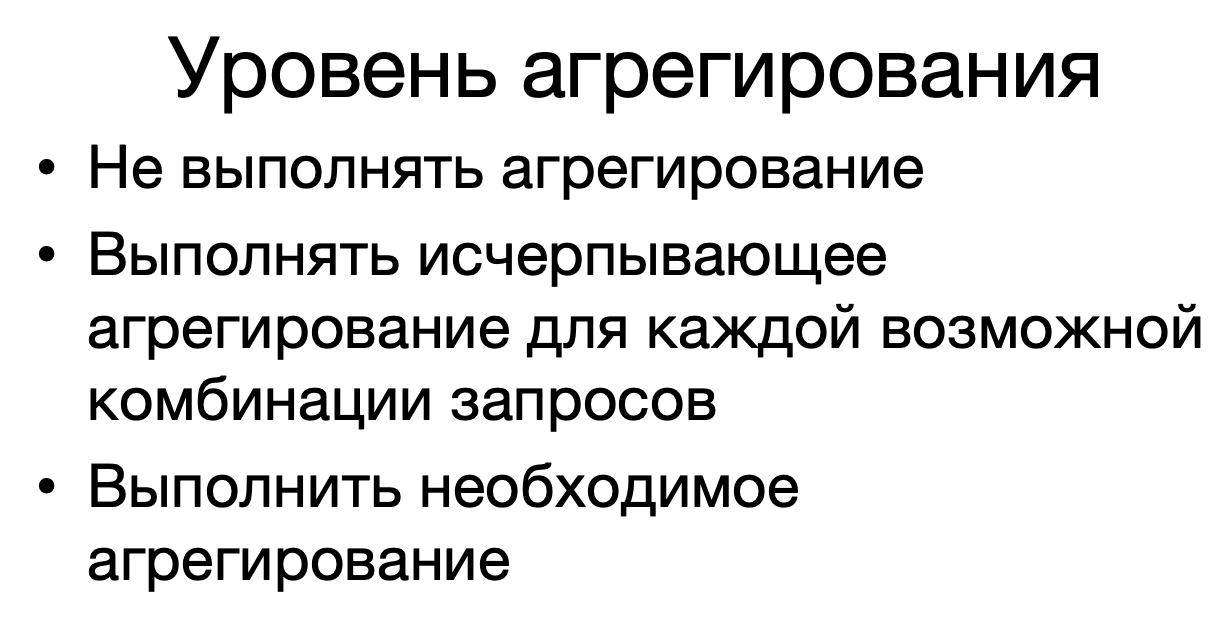
OLAP-куб (Online Analytical Processing) — это многомерная структура данных, которая позволяет анализировать информацию по нескольким измерениям одновременно.

#### **Архитектура куба**

1. **Измерения**: Это категории, по которым данные могут быть агрегированы, например, время, география, продукт.
2. **Факты**: Это количественные показатели, которые мы анализируем, например, продажи, доход, количество заказов.
3. **Атрибуты**: Дополнительные характеристики измерений, которые помогают уточнить анализ (например, категория продукта, регион).
4. **Мера**: Значение, которое мы хотим агрегировать (например, сумма продаж).

### Типы архитектур

1. MOLAP (Multidimensional OLAP)
   * Данные хранятся в специальной многомерной БД
   * Примеры: Microsoft Analysis Services, Oracle Essbase
   * Плюсы: высокая скорость, сложные вычисления
   * Минусы: долгая загрузка, ограниченный объем
2. ROLAP (Relational OLAP)
   * Использует реляционную БД с "виртуальным" кубом
   * Пример: Snowflake, Redshift с SQL-агрегациями
   * Плюсы: работа с большими данными, гибкость
   * Минусы: ниже производительность для сложных срезов
3. HOLAP (Hybrid OLAP)
   * Комбинация MOLAP и ROLAP
   * Текущие данные — в реляционной БД, агрегаты — в MOLAP



**Физическое хранение куба** включает в себя способы организации и хранения данных на диске для оптимизации доступа и производительности. Основные аспекты физического хранения:

1. **Структура хранения**:  
   * Данные могут храниться в виде многомерных массивов, что позволяет эффективно выполнять запросы.
   * Можно использовать реляционные базы данных для хранения кубов и создания представлений для быстрого доступа.
2. **Индексация**:  
   * Индексы могут использоваться для ускорения доступа к данным в кубе, особенно при выполнении запросов на агрегацию.
3. **Кэширование**:  
   * Кэширование результатов запросов позволяет уменьшить время отклика при повторных запросах к одним и тем же данным.

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

### **SQL Server Analysis Services (SSAS) — это компонент Microsoft SQL Server, который позволяет создавать и управлять многомерными кубами и моделями данных для бизнес-аналитики. Вот основные термины, с которыми вы столкнетесь в SSAS:**

### **Куб: Многомерная структура данных, которая хранит факты и измерения. Кубы позволяют выполнять сложные аналитические запросы.**

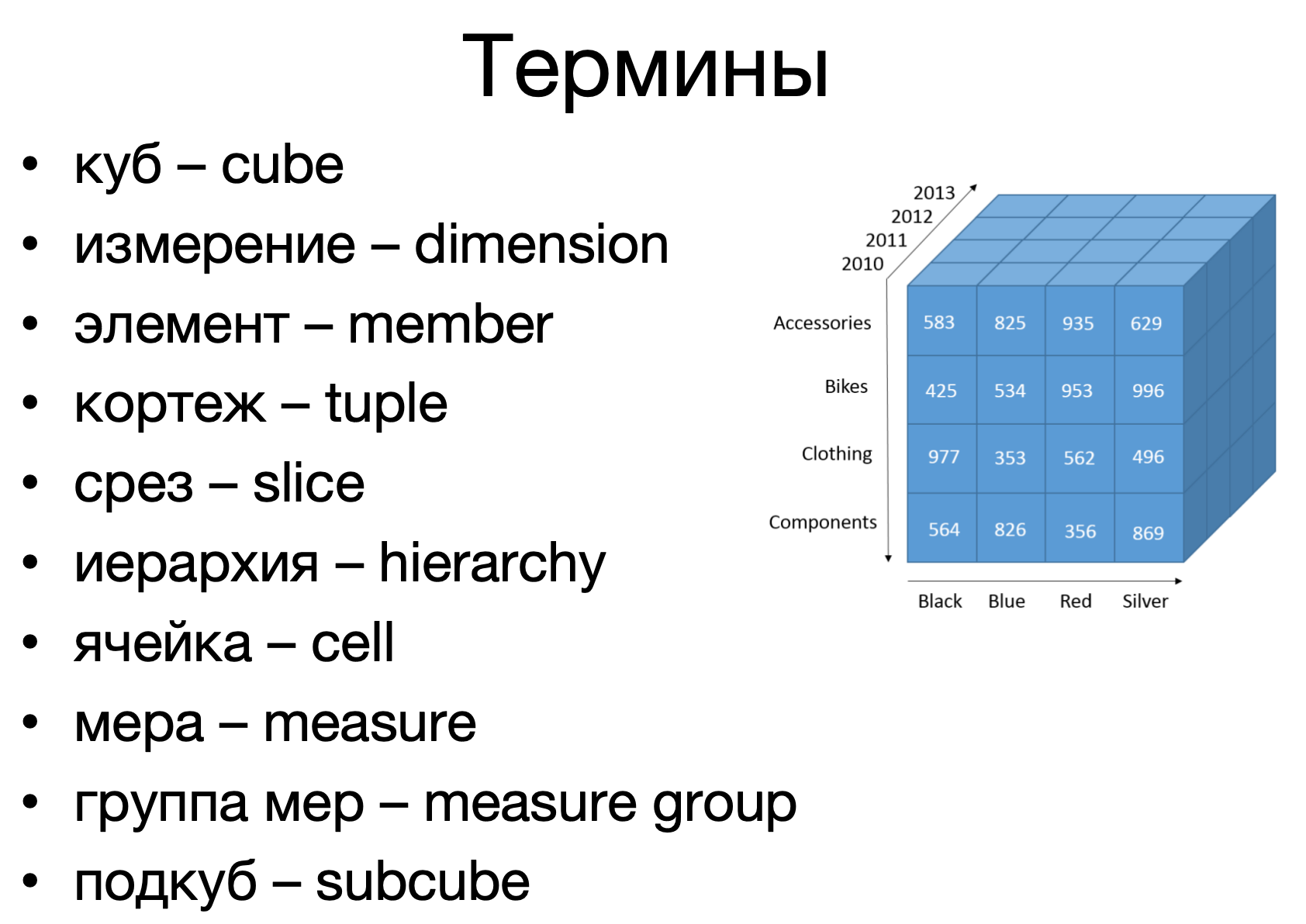
### **Факты: Количественные данные, которые собираются и анализируются. Например, сумма продаж или количество заказов.**

### **Измерения: Категории, по которым факты могут быть агрегированы. Например, время, география и продукт.**

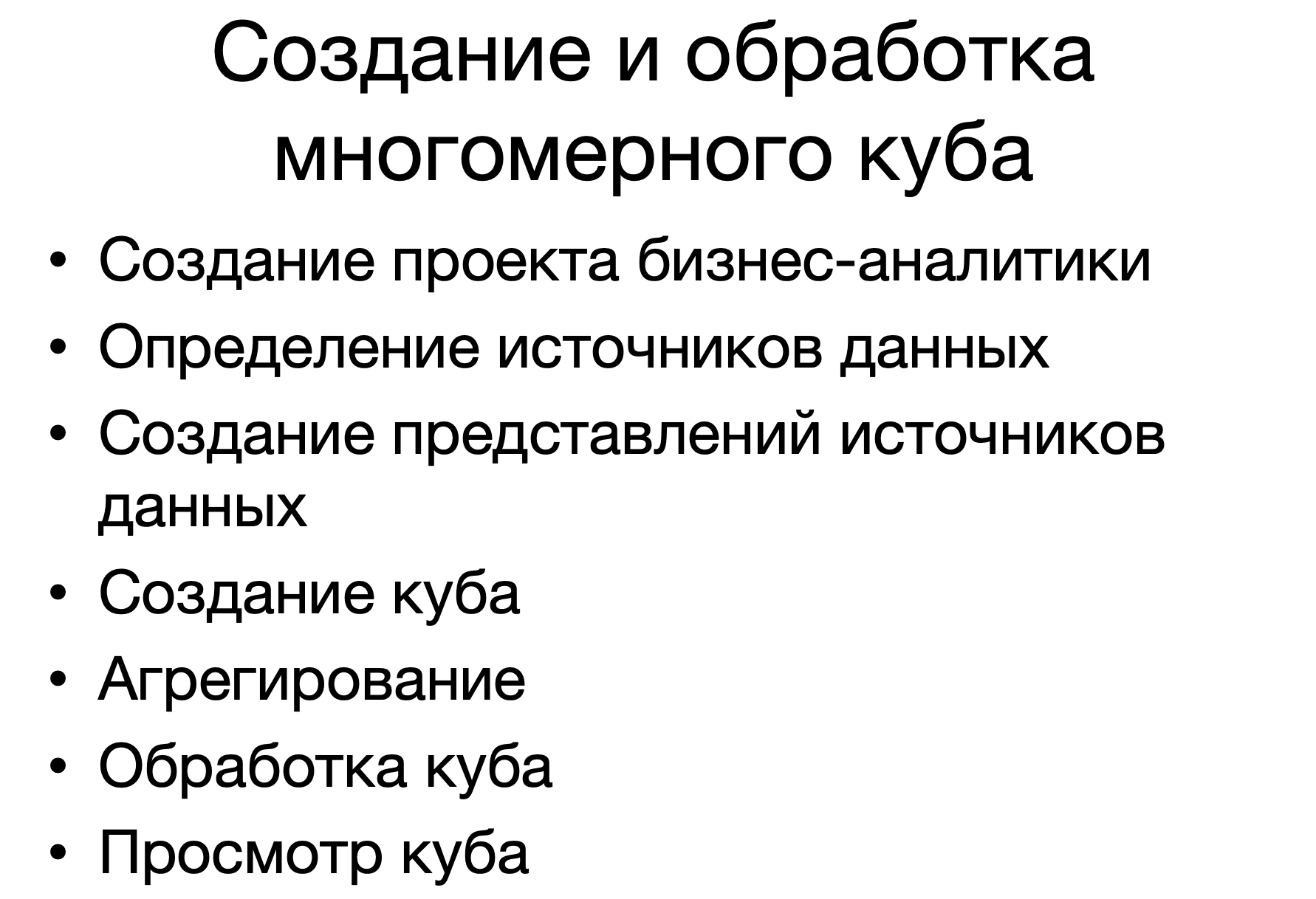
### **Атрибуты: Дополнительные характеристики измерений, такие как название продукта или регион.**

### **Меры: Значения, которые мы хотим агрегировать, например, общая сумма продаж.**

### 



* Куб (Cube) - многомерная структура данных для анализа
* Измерение (Dimension) - атрибуты для анализа данных (например, время, география)
* Мера (Measure) - количественные показатели для анализа (продажи, прибыль)
* Иерархия (Hierarchy) - логическая структура уровней в измерении (Год → Квартал → Месяц)
* Уровень (Level) - элемент иерархии (например, "Месяц" в иерархии времени)
* Ячейка (Cell) - точка пересечения измерений в кубе
* Секция (Partition) - часть куба для управления хранением и обработкой
* Перспектива (Perspective) - подмножество куба для конкретных пользователей
* KPI (Key Performance Indicator) - ключевые показатели эффективности
* Вычисляемый элемент (Calculated Member) - динамически вычисляемые значения



### **Просмотр многомерного куба**

1. **Использование SQL Server Management Studio (SSMS)**:  
   * Подключитесь к вашему серверу SSAS и найдите ваш куб. Щелкните правой кнопкой мыши и выберите "Browse" (Просмотр).
2. **Анализ данных**:  
   * В интерфейсе просмотра вы можете исследовать данные, используя различные измерения и меры. Вы можете перетаскивать атрибуты в области фильтров или строк для анализа.
3. **Создание отчетов**:  
   * Вы можете использовать инструменты, такие как Excel или Power BI, для создания отчетов на основе данных из вашего куба.

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

NoSQL (Not Only SQL) — нереляционные базы данных, предназначенные для работы с большими объемами данных и обеспечивающие высокую производительность и гибкость.

**NoSQL – преимущества**

•Гибкость: быстрее разработка, возможна поэтапная реализация

–Частично структурированные или неструктурированные данных

•Масштабируемость: использование кластеров

•Высокая производительность: Оптимизированы для больших данных

•Широкие функциональные возможности

**Основные типы NoSQL БД:**

1. **Документные** (MongoDB, CouchDB) - данные хранятся в виде JSON-документов
2. **Ключ-значение** (Redis, DynamoDB) - простейшая модель хранения
3. **Колоночные** (Cassandra, HBase) - оптимизированы для аналитики
4. **Графовые** (Neo4j) - для хранения и анализа связей

**MongoDB -** Документо-ориентированная СУБД

База данных представляет собой набор коллекций

Коллекции содержат документы

Документы - основная единица хранения в MongoDB, могут иметь разную структуру. Для каждого документа имеется \_id. Если явно не указать, то автоматически сгенерируется

**MongoDB поддерживает операции CRUD (Create, Read, Update, Delete)**

Создание коллекции: db.createCollection("users")

Вставка одного документа: db.users.insertOne({ name: "Иван", age: 30, city: "Москва"})

Поиск всех документов в коллекции db.users.find(), db.users.find({ age: { $gt: 25 } })

**Обновление одного документа**: Изменить город для пользователя с именем "Иван": db.users.updateOne( { name: "Иван" }, { $set: { city: "Казань" } })

Удаление одного документа: db.users.deleteOne({ name: "Иван" })

Удаление коллекции:db.users.drop()

38. MongoDB. Поиск в документах. Проекции. Ограничение выборки. Сортировка.(В ЛЕКЦИИ БЫЛО НЕ ВСЕ)

Проекции позволяют указать, какие поля документа должны быть возвращены в результате запроса.

db.users.find({}, { name: 1, age: 1 }) включить поля

db.users.find({}, { password: 0 }) исключить поля

Поиск в MongoDB осуществляется с помощью метода find() или findOne(). Он позволяет находить документы в коллекции по заданным критериям, используя фильтры (условия).

**Простой поиск**: Найти все документы: db.products.find()

**Поиск по условию**: Найти товары категории "Электроника": db.products.find({ category: "Электроника" })

**Поиск с несколькими условиями**: Найти товары категории "Электроника" с ценой выше 40000: db.products.find({ category: "Электроника", price: { $gt: 40000 } })

**Поиск с операторами**: db.products.find({ stock: { $gte: 5, $lte: 10 } })

db.products.find({ name: { $regex: "Телефон", $options: "i" } })

**Поиск одного документа**: db.products.findOne({ \_id: 1 })

### **Ограничение выборки**

Для ограничения количества возвращаемых документов используется метод limit(): db.products.find().limit(2)

Пропуск документов: Метод skip() позволяет пропустить указанное количество документов: db.products.find().skip(2).limit(2)

### **Сортировка**

Сортировка результатов выполняется с помощью метода sort(). Поля сортируются по возрастанию (1) или убыванию (-1). db.products.find().sort({ price: 1 }); db.products.find().sort({ category: 1, price: -1 })

КОМБИНИРОВАННЫЙ ПРИМЕР

db.products.find(

{ category: "Электроника", price: { $gt: 20000 } },

{ name: 1, price: 1, \_id: 0 }

).sort({ price: -1 }).limit(2)

39. MongoDB. Индексирование. Виды индексов. Планы запросов. Оценка планов. Хинты.(В ЛЕКЦИИ БЫЛО НЕ ВСЕ)

db.collection.createIndex({ <поле>: 1 }) // 1 — по возрастанию, -1 — по убыванию

db.collection.getIndexes()

db.collection.dropIndex("index\_name")

Виды индексов:

•Простые индексы db.products.createIndex({ price: 1 })

•Составные индексы db.products.createIndex({ category: 1, price: -1 })

•Уникальные индексы db.users.createIndex({ email: 1 }, { unique: true })

•Индексы покрытия все запрашиваемые поля и условия фильтрации содержатся в самом индексе. Содержат все поля, необходимые для запроса (не требуется обращение к документам):

db.products.createIndex({ category: 1, name: 1 })

// Запрос, который может быть покрыт индексом:

db.products.find({ category: "Электроника" }, { \_id: 0, category: 1, name: 1 })

•Пространственные индексы

•Частичные индексы индексируют только документы, соответствующие условию:

db.products.createIndex(

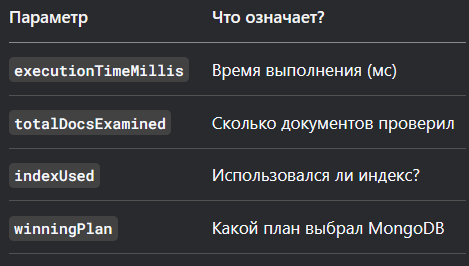
{ price: 1 },

{ partialFilterExpression: { stock: { $gt: 0 } } }

)

MongoDB использует план запроса для определения, как выполнять запрос (например, использовать индекс или сканировать коллекцию).

db.products.find({ category: "Электроника" }).explain("executionStats")



### Режимы explain():

### **"queryPlanner"** (по умолчанию):

### Показывает выбранный план запроса

### Не выполняет запрос фактически

### **"executionStats"**:

### Выполняет запрос

### Показывает статистику выполнения

### Включает:

### Кол-во просмотренных документов (totalDocsExamined)

### Время выполнения

### Использованные индексы

### **"allPlansExecution"**:

### Сравнивает все возможные планы выполнения

### Показывает статистику для каждого

### **Хинты (Hints)**

Хинты позволяют явно указать MongoDB, какой индекс использовать для запроса, переопределяя выбор оптимизатора.

db.products.createIndex({ category: 1 })

db.products.createIndex({ price: 1 })

db.products.find({ category: "Электроника" }).hint({ price: 1 })

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

MongoDB поддерживает два типа пространственных индексов для работы с геоданными:

**1.1.**2dsphere**(для сферической геометрии)**

* Использует **WGS84** (стандарт GPS: долгота/широта).
* Поддерживает **GeoJSON** (точки, линии, полигоны).
* Оптимизирован для запросов на сфере (реальная Земля).

**Создание индекса:**

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

**1.2.**2d**(для плоской геометрии)**

* Работает с **плоскими координатами** (например, игровые карты).
* Использует **массив**[x, y] (не долгота/широта!).

**Создание индекса:**

db.places.createIndex({ position: "2d" })

**3. Методы геопоиска (с индексами)**

**3.1.**$geoWithin**— поиск внутри области**

Находит объекты **внутри** заданной геометрической фигуры.

**Примеры:**

**а) Внутри прямоугольника (**$box**)**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

**б) Внутри круга (**$center**или**$centerSphere**)**

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

**в) Внутри полигона (**$polygon**)**

Изображение выглядит как текст, Шрифт, снимок экрана, число

Содержимое, созданное искусственным интеллектом, может быть неверным.

**3.2.**$near**/**$nearSphere**— поиск ближайших точек**

Находит объекты **близко** к заданной точке, сортируя по расстоянию.

**Пример для**2dsphere**:**

Изображение выглядит как текст, снимок экрана, Шрифт

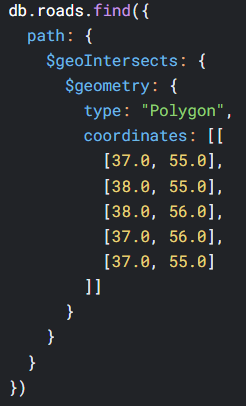
Содержимое, созданное искусственным интеллектом, может быть неверным.

**Пример для**2d**:**

Изображение выглядит как текст, Шрифт, снимок экрана

Содержимое, созданное искусственным интеллектом, может быть неверным.  
**3.3.**$geoIntersects**— поиск пересечений**

Находит объекты, **пересекающие** заданную геометрию (например, дороги через полигон).



**3.4.**$geoNear**(агрегация) — расширенный поиск**

Аналог $near, но в агрегационном пайплайне. Возвращает **расстояние** до точки.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

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

**1. Текстовые индексы в MongoDB**

Текстовые индексы предназначены для **полнотекстового поиска** по строковым данным. Они поддерживают:

* Поиск по словам и фразам.
* Игнорирование стоп-слов (например, "и", "в", "на").
* **Стемминг** (приведение слов к корневой форме).
* Оценку **релевантности** результатов.

**Ограничения:**

* В коллекции может быть **только один текстовый индекс**.
* Нельзя комбинировать с другими типами индексов (например, числовыми).
* Не поддерживает $regex и точное совпадение.

**2. Создание текстовых индексов**

**2.1. Базовый текстовый индекс**

Индексирует одно поле:

db.articles.createIndex({ content: "text" })

**2.2. Индекс на несколько полей**

Индексирует несколько полей для комплексного поиска:

db.articles.createIndex({

title: "text",

description: "text",

tags: "text"

})

**2.3. Индекс на все строковые поля**

Создает индекс для **всех** строковых полей коллекции:

db.articles.createIndex({ "$\*\*": "text" })

**3. Индексы с весами**

Можно задать **веса** полям, чтобы увеличить их влияние на релевантность.

db.articles.createIndex(

{ title: "text", content: "text" },

{ weights: { title: 10, content: 5 } } // title в 2 раза важнее content

)

При поиске совпадения в title будут иметь **больший вес**, чем в content.

**4. Стемминг (приведение слов к корневой форме)**

MongoDB автоматически применяет **стемминг** для указанного языка (по умолчанию — английский).  
Пример для русского языка:

db.articles.createIndex(

{ content: "text" },

{ default\_language: "russian" } // включаем стемминг для русского

)

db.articles.find({ $text: { $search: "учиться" } }) // найдет "учусь", "учебник"

**5. Релевантность результатов**

MongoDB оценивает релевантность по:

1. **Частоте слова** в документе.
2. **Весу поля** (если задан в индексе).
3. **Коэффициенту точности** (редкие слова считаются более релевантными).
4. **Количеству совпавших слов** из запроса.

db.articles.find(

{ $text: { $search: "MongoDB индексы" } },

{ score: { $meta: "textScore" } } // добавляет поле с оценкой релевантности

).sort({ score: { $meta: "textScore" } }) // сортировка по релевантности

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

Фреймворк агрегации в MongoDB — это **мощный инструмент для последовательного пайплайна**, который позволяет:

* Фильтровать, группировать и преобразовывать документы.
* Выполнять вычисления (суммы, средние, статистику).
* Объединять данные из разных коллекций (аналог JOIN в SQL).
* Создавать сложные отчеты и аналитические выборки.

db.collection.aggregate([ ...стадии... ])

Пайплайн — это **последовательность стадий**, через которые проходят документы. Каждая стадия обрабатывает данные и передает результат следующей.

db.orders.aggregate([

{ $match: { status: "completed" } }, // 1-я стадия: фильтрация

{ $group: { \_id: "$customer", total: { $sum: "$amount" } } }, // 2-я стадия: группировка

{ $sort: { total: -1 } } // 3-я стадия: сортировка

])

**$match** — Фильтрует документы по заданным условиям, аналогично find(). Уменьшает количество документов для последующих стадий.

**$group** — Группирует документы по указанному ключу и выполняет агрегационные операции (например, $sum, $avg, $min, $max, $push).  
 Используется для подсчёта статистик или объединения данных.

**$project** — Изменяет структуру документов: включает или исключает поля, создаёт вычисляемые поля. Форматирует вывод или добавляет новые данные.

**$sort** — Сортирует документы по указанному полю (1 — по возрастанию, -1 — по убыванию).

**$limit** — Ограничивает количество возвращаемых документов.

**$skip** — Пропускает указанное количество документов.

**$unwind** — Разворачивает массив в отдельные документы для каждого элемента массива. Позволяет работать с элементами массивов как с отдельными документами.

**$lookup** — Выполняет соединение (аналог JOIN) с другой коллекцией, добавляя данные из неё в документы.

**$addFields** — Добавляет новые поля в документы, сохраняя исходную структуру. Используется для вычисляемых полей.

**$count** — Подсчитывает количество документов на текущей стадии.

**$bucket** — Группирует документы по диапазонам значений указанного поля. Удобно для создания гистограмм или категоризации.

**$facet** — Выполняет несколько под-пайплайнов в рамках одного запроса, возвращая разные результаты.  
 Используется для многогранного анализа данных.

**$out** — Записывает результаты пайплайна в новую коллекцию, заменяя её содержимое. Используется для сохранения результатов агрегации.

**$merge** — Записывает результаты пайплайна в существующую или новую коллекцию с возможностью обновления или добавления данных.

**$geoNear** — Выполняет поиск ближайших точек с использованием геопространственного индекса, сортируя по расстоянию. Применяется для географических данных.

**$sample** — Возвращает случайную выборку документов из коллекции.

**$set** — Добавляет, обновляет или удаляет поля в документах (аналог $addFields, но с возможностью перезаписи). Используется для изменения структуры документов.

**$unset** — Удаляет указанные поля из документов.

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

**1. Аутентификация в MongoDB**

MongoDB поддерживает несколько методов аутентификации:

**1.1. Базовая аутентификация (логин/пароль)**

* Пользователи создаются в конкретной БД (обычно admin для администрирования).
* Пароли хранятся в зашифрованном виде.

**1.2. Другие методы аутентификации**

* **x.509** – аутентификация по SSL-сертификатам.
* **LDAP** – интеграция с корпоративными каталогами (Active Directory).
* **Kerberos** – для enterprise-сред.

**2. Пользователи и их управление**

Пользователи привязаны к конкретной БД, но могут иметь роли в других БД.

**2.1. Создание пользователя**

use mydb // создаем пользователя в БД mydb

db.createUser({

user: "appUser",

pwd: "appPassword",

roles: [

{ role: "readWrite", db: "mydb" }, // чтение и запись в mydb

{ role: "read", db: "reporting" } // только чтение в reporting

]

})

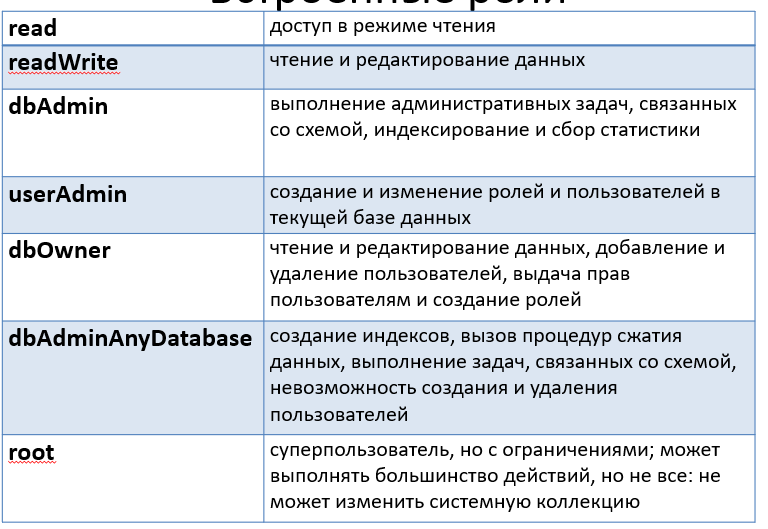
**2.2. Управление пользователями**

| **Команда** | **Описание** |
| --- | --- |
| db.getUsers() | Список пользователей |
| db.updateUser("user", { ... }) | Изменить пользователя |
| db.changeUserPassword("user", "newPass") | Смена пароля |
| db.dropUser("user") | Удалить пользователя |
| db.grantRolesToUser("user", [{ role: "read", db: "test" }]) | Добавить роли |
| db.revokeRolesFromUser("user", [{ role: "read", db: "test" }]) | Отозвать роли |

**3. Роли и привилегии**

Роли определяют, какие действия может выполнять пользователь.

**3.1. Встроенные роли MongoDB**



**3.2. Пользовательские роли**

Можно создавать роли с конкретными привилегиями.

**Пример:**

use admin

db.createRole({

role: "orderManager",

privileges: [

{

resource: { db: "shop", collection: "orders" },

actions: ["find", "insert", "update"]

}

],

roles: [] // можно наследовать другие роли

})

**4. Привилегии (разрешения)**

Привилегии определяют **конкретные действия** над **ресурсами** (БД, коллекции, кластер).

**Основные действия:**

| **Действие** | **Описание** |
| --- | --- |
| find | Чтение документов |
| insert | Добавление документов |
| update | Изменение документов |
| delete | Удаление документов |
| createCollection | Создание коллекций |
| createIndex | Создание индексов |
| shutdown | Остановка сервера |
| replSetConfigure | Настройка Replica Set |

**5. Включение аутентификации**

Чтобы требовать аутентификацию, нужно:

1. Добавить первого пользователя в admin.
2. Запустить MongoDB с параметром --auth (или прописать в конфиге security.authorization: enabled).

**Подключение с аутентификацией:**

mongosh -u "adminUser" -p "securePassword123" --authenticationDatabase "admin"

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

**1. Устройство базы данных**

MongoDB поддерживает несколько архитектурных подходов для хранения данных:

**1.1. Одиночный сервер**

* Простейшая конфигурация (для разработки и тестирования).
* Один сервер MongoDB, хранящий все данные.
* **Не обеспечивает отказоустойчивость** (при падении сервера данные теряются).

**1.2. Набор реплик (Replica Set)**

* Группа из нескольких серверов (минимум 3 узла).
* **Первичный узел (Primary)** – принимает все операции записи.
* **Вторичные узлы (Secondaries)** – хранят копии данных, могут обслуживать чтение.
* **Автоматическое переключение при отказе** (если Primary падает, один из Secondaries становится новым Primary).

**1.3. Шардинг (Sharding)**

* Горизонтальное масштабирование: данные разбиваются на части (**шарды**) и распределяются по нескольким наборам реплик.
* **Маршрутизатор (mongos)** управляет распределением запросов.
* Позволяет обрабатывать большие объемы данных и высокие нагрузки.

**2. Файлы и механизмы хранения**

**2.1. Организация файлов**

* Каждая **коллекция** хранится в отдельном файле.
* Каждый **индекс** также хранится в отдельном файле.
* Используются **журналы транзакций (WAL)** для надежности.

**2.2. Движки хранения (Storage Engines)**

**WiredTiger (стандартный с MongoDB 3.2+)**

* **Оптимизирован для записи и чтения**.
* Поддерживает **сжатие данных** (экономит место на диске).
* Использует **многопоточность** для высокой производительности.
* Поддерживает **транзакции** (ACID на уровне документа).
* **Журналирование (Write-Ahead Log, WAL)** – гарантирует сохранность данных при сбоях.

**MMAPv1 (устарел, использовался до MongoDB 4.0)**

* **Оптимизирован для чтения**.
* Использует **отображение файлов в память (memory-mapped files)**.
* **Блокировка на уровне коллекции** (может снижать производительность при записи).
* **Фрагментация данных** – требует периодического сжатия (compact).

**3. Механизмы чтения и записи**

**3.1. Чтение данных**

* По умолчанию чтение происходит с **Primary**.
* Можно настроить чтение с **Replica Set** (readPreference).
* **Индексы ускоряют поиск** (используется B-дерево).

**3.2. Запись данных**

* Все операции записи идут на **Primary**.
* Данные сначала записываются в **журнал (WAL)**, затем в основное хранилище.
* Поддерживается **подтверждение записи (write concern)** для контроля надежности.

**4. Ограничения целостности (Data Validation)**

MongoDB (в отличие от SQL) **не имеет строгих ограничений по умолчанию**, но их можно задать через **валидацию схемы**:

**Пример валидации коллекции**



**5. Типы данных в MongoDB**

MongoDB использует **BSON (Binary JSON)** – расширенную версию JSON с поддержкой дополнительных типов:

Типы данных:

•String – строковый тип данных, UTF-8

•Array – массив

•Binary data – данные в двоичном формате

•Boolean – логические значения, TRUE/FALSE

•Date – дата

•JavaScript – код JavaScript

•Object – объект, который содержит набор свойств

•ObjectId – идентификатор документа

•Integer – целочисленные значения, 32 бита

•Long – целочисленные значения, 64 бита

•Double – числа с плавающей точкой

•Decimal128 – десятичные числа, 128 бит

•Regular expression - регулярные выражения

45. Анализ данных в Python. Специализированные библиотеки.

1. **NumPy**: Численные вычисления с массивами.
   * Математические функции (линейная алгебра, преобразования Фурье)
   * Пример: np.array(), np.mean(), np.linalg.inv()
   * Основа для Pandas, SciPy, Scikit-learn.
2. **SciPy**: Научные вычисления и статистика. Расширяет NumPy.
   * Научные и технические вычисления
   * Оптимизация, статистика, обработка сигналов
   * Пример: scipy.stats.ttest\_ind()
3. **Pandas**: Обработка и анализ табличных данных.
   * Структуры данных: Series (одномерные) и DataFrame (двумерные)
   * Чтение/запись данных (CSV, Excel, SQL)
   * Пример: df = pd.read\_csv('data.csv')
4. **Matplotlib**: Визуализация данных (статические графики).
   * Создание линейных графиков, гистограмм, диаграмм рассеяния.
   * Настройка осей, подписей, стилей.
   * Интеграция с Pandas для визуализации.
   * Пример: plt.plot(x, y)
5. **Scikit-learn**:
   * Машинное обучение (классификация, регрессия, кластеризация)
   * Метрики оценки моделей
   * Пример: from sklearn.linear\_model import LinearRegression

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

**Анализ данных в Python**

Анализ данных в Python осуществляется с использованием библиотек, таких как **NumPy**, **SciPy**, **Pandas**, **Matplotlib** и **Scikit-learn** для визуализации. Основные этапы анализа данных:

* **Сбор и загрузка данных**: Используйте pandas.read\_csv(), read\_excel() или другие методы для импорта данных.
* **Очистка данных**: Обработка пропущенных значений (df.dropna(), df.fillna()), удаление дубликатов (df.drop\_duplicates()), преобразование типов данных.
* **Описательный анализ**: Вычисление базовых статистических показателей (среднее, медиана, мода, дисперсия и т.д.) с помощью df.describe() или функций NumPy (np.mean(), np.std()).
* **Визуализация**: Построение графиков с использованием Matplotlib.
* **Проверка гипотез**: Использование статистических тестов из SciPy или Statsmodels для подтверждения или опровержения гипотез.
* Сравнение средних: t-тест, ANOVA.
* Проверка корреляции: Пирсон, Спирмен.
* **Моделирование**: Применение регрессионных моделей, кластеризации или других методов машинного обучения (Scikit-learn).

#### **Основные понятия статистики**

* **Меры центральной тенденции (где центр данных?)**:
  + **Среднее** (np.mean()): — обычное среднее.
  + **Медиана** (np.median()): значение в середине.
  + **Мода** (df.mode()):  — самое частое значение.
* **Меры изменчивости (как данные разбросаны?)**:
  + **Дисперсия** (np.var()): средний квадрат отклонений.
  + **Стандартное отклонение** (np.std()): корень из дисперсии.
  + **Размах**: Разница между максимумом и минимумом.
  + **Квантили/перцентили** (df.quantile()): Значения, ниже которых лежит определённый процент данных.
* **Распределения (форма данных)**:
  + **Нормальное распределение**: описывается средним и стандартным отклонением ("Колокол" (среднее = медиана = мода)).
  + **Смещённые распределения** (экспоненциальное).

#### **Проверка гипотез**

Проверка гипотез — это метод статистического вывода, позволяющий оценить, подтверждаются ли данные гипотезой.

* **H₀ (нулевая): "Различий нет" (например, "Кофе не влияет на продуктивность").**
* **H₁ (альтернативная): "Различия есть" ("Кофе повышает продуктивность").**

Статистический вывод (инференциальная статистика) позволяет делать обобщения о генеральной совокупности на основе выборки. Основные подходы:

* **Оценка параметров**:
  + **Точечная оценка**: Например, среднее выборки как оценка среднего совокупности.
  + **Интервальная оценка**: Доверительные интервалы (например, с statsmodels.stats).
* **Проверка гипотез**: Определение, является ли наблюдаемый эффект случайным.
* **Регрессионный анализ**: Моделирование зависимости между переменными (statsmodels или scikit-learn).

47. Анализ данных в Python. Сравнение средних значений.

Сравнение средних значений — это статистический метод, используемый для определения, различаются ли средние значения двух или более выборок.

* **Среднее значение — это средний результат в выборке.**
* **Нулевая (H₀):** "Средние значения **равны**" (например, "Кофе не влияет на продуктивность").
* **Альтернативная (H₁):** "Средние значения **различаются**" ("Кофе повышает продуктивность").
* **Уровень значимости (α)**: Если вероятность ошибиться (p-value) < 5%, считаем различия **значимыми**.

Основные тесты для сравнения средних:

* **t-тест**: Для двух выборок (независимых или парного соответствия).
* **ANOVA**: Для трёх и более выборок.
* **Непараметрические тесты** (например, тест Манна-Уитни): Если данные не соответствуют нормальному распределению.

48. Анализ данных в Python. Корреляция и регрессия.

**1. Корреляция: "Есть ли связь между X и Y?"**

Корреляция измеряет силу и направление линейной связи между двумя переменными.

**Пример из жизни:**

* Ты замечаешь, что чем больше человек тратит времени на учёбу (X), тем выше его оценка (Y).
* Корреляция **показывает**, насколько чётко эта связь прослеживается.

##### **Основные понятия:**

**Коэффициент корреляции Пирсона**: нормальное распределение

* 1 — идеальная прямая связь (X растёт → Y растёт).
* -1 — идеальная обратная связь (X растёт → Y падает).
* 0 — связи нет.

**Коэффициент корреляции Спирмена**: Непараметрический метод, основан на рангах, не требует нормальности.

Для вычисления корреляции можно использовать метод .corr()

#### **2. Регрессия: "Как именно X влияет на Y?"**

Регрессия моделирует зависимость одной переменной (зависимой) от одной или нескольких других (независимых).

**Пример:**  
Хочешь предсказать оценку (Y) на основе времени учёбы (X).

**Линейная регрессия** строит прямую линию по формуле:

Оценка = β₀ + β₁ \* Время\_учёбы kx + b

* β₀ — начальная оценка (если не учиться вообще).
* β₁ — насколько повышается оценка за каждый дополнительный час.
* **Коэффициент детерминации (R²)**: Доля объяснённой дисперсии (0–1). Чем выше R², тем лучше модель объясняет данные.

**Когда что использовать?**

* **Корреляция** — для быстрой проверки связи между двумя переменными.
* **Регрессия** — если нужно предсказать значение или понять математическую зависимость.

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

**1. Что такое Bootstrap?**

Представьте, что у вас есть мешок с 100 шариками разного цвета (это ваши данные).

**Метод Bootstrap — это когда:**

1. Вы достаёте случайный шарик, **записываете его цвет** и **кидаете обратно**.
2. Повторяете это 100 раз.
3. В итоге у вас новая выборка, где некоторые шарики попались несколько раз, а некоторые — ни разу.

**Зачем это нужно?**

* Чтобы создать много "клонов" исходных данных с небольшими отличиями.
* Это помогает понять, насколько устойчивы ваши выводы.

**2. Как это связано с деревьями решений?**

**Дерево решений** — это алгоритм, который задаёт вопросы вроде:

* "Если возраст > 30, то идём налево, иначе направо".
* Проблема: одно дерево может "переучиться"

**Решение:**

1. Создаём **100 Bootstrap-выборок** из исходных данных.
2. Для каждой выборки строим **отдельное дерево**.
3. При предсказании **усредняем ответы всех деревьев** (для чисел) или **выбираем большинство голосов** (для категорий).

**Пример:**

* Дерево 1: "Этот клиент вернёт кредит (60% вероятности)".
* Дерево 2: "Не вернёт (55%)".
* Дерево 3: "Вернёт (70%)".
* **Итог:** Среднее = 62% → клиент скорее вернёт деньги.

**3. Random Forest — "лес" из Bootstrap-деревьев**

**Random Forest** — это улучшенная версия:

* Каждое дерево обучается на своей Bootstrap-выборке.
* Плюс случайно выбираются **не все признаки**, а только часть (например, 5 из 10).
* Результат: модель становится **стабильнее и точнее**.

**Аналогия:**

* Одно дерево = один врач (может ошибиться).
* Random Forest = консилиум из 100 врачей (вероятность ошибки меньше).

**4. Как это сделать в Python?**

**Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.**

**Что здесь происходит?**

* n\_estimators=100 — строим 100 деревьев.
* bootstrap=True — каждое дерево обучается на своей случайной выборке.

**5. Зачем это всё?**

**Точность**: Ансамбли деревьев работают лучше, чем одно дерево.  
**Устойчивость**: Меньше риск переобучения.  
**Оценка надёжности**: Можно посчитать доверительные интервалы

* **Bootstrap** — это способ создавать много похожих, но разных выборок из данных.
* **Деревья решений + Bootstrap** = Random Forest (мощный алгоритм для предсказаний).

50. Анализ данных в Python. Применение методов Airflow.

Apache Airflow — платформа для автоматизации, планирования и мониторинга рабочих процессов (workflow), часто используемая для управления пайплайнами обработки данных. В контексте анализа данных Airflow помогает организовать сложные процессы ETL (Extract, Transform, Load), автоматизировать запуск скриптов Python для анализа данных и интегрировать их с различными источниками данных и инструментами.

#### **Применение Airflow для анализа данных**

Airflow позволяет автоматизировать пайплайны анализа данных, включая:

* Извлечение данных из источников (например, CSV, SQL, API).
* Преобразование данных (очистка, агрегация, статистический анализ).
* Загрузка результатов (в базы данных, файлы, системы визуализации).

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.