**Лабораторная работа по теме «Выбор хранилища данных»**

**Задача**:

Вы являетесь аналитиком данных в небольшой компании, которая решает перейти к современной системе управления данными. Вам поручено выбрать и обосновать выбор хранилища данных для компании. Ваша компания работает в сфере e-commerce и продает товары через онлайн-платформу. У вас есть следующие требования:

**Объем данных**: Ваша компания собирает миллионы записей о продажах, клиентах и инвентаре ежедневно.

**Аналитика**: Вам необходимо обеспечить возможность проведения сложных аналитических запросов к данным для выявления тенденций продаж, сегментации клиентов и определения эффективности маркетинговых кампаний.

**Масштабируемость**: Ваша компания растет, и вам нужна система, которая легко масштабируется с ростом объема данных и нагрузки.

**Реальное время**: Вам нужна возможность анализа данных в реальном времени для мониторинга заказов и складской информации.

**Задачи**:

***1. Выберите одно или несколько хранилищ данных (реляционная база данных, NoSQL, колоночное хранилище и т. д.), которые, по вашему мнению, наилучшим образом соответствуют требованиям компании.***

Для выбора оптимального хранилища данных для вашей компании, рассмотрим следующие варианты:

**Реляционные базы данных** (например, MySQL, PostgreSQL) — это классический вариант хранения данных, который хорошо подходит для структурированных данных, таких как данные о продажах, клиентах и инвентаре. Они обладают высокой производительностью при выполнении сложных запросов и обеспечивают возможность создания сложных структур данных. Однако они могут быть не очень подходящими для больших объемов неструктурированных данных, таких как данные о поведении пользователей на сайте.

*PostgreSQL* имеет более высокую производительность при выполнении сложных запросов, чем MySQL. Это связано с тем, что PostgreSQL использует более эффективные алгоритмы выполнения запросов и имеет более оптимизированный движок базы данных. Кроме того, PostgreSQL поддерживает более широкий набор функций и возможностей, чем MySQL, что позволяет более гибко работать с данными.

**NoSQL базы данных** (например, MongoDB, Cassandra) — это более гибкие хранилища данных, которые лучше подходят для больших объемов неструктурированных данных. Они позволяют хранить данные в формате JSON или ключ-значение, что делает их более удобными для работы с данными, которые не имеют жесткой структуры. Однако они могут быть менее производительными при выполнении сложных запросов и могут потребовать больше усилий для создания сложных структур данных.

**Колоночные хранилища** (например, Amazon DynamoDB, Google Cloud SQL) — это еще более гибкие хранилища данных, которые хорошо подходят для работы с большими объемами данных. Они позволяют хранить данные в виде таблиц с колонками, что делает их более удобными для работы с данными, которые не имеют жесткой структуры. Однако они могут быть менее производительными при выполнении сложных запросов и могут потребовать больше усилий для создания сложных структур данных.

Для нашей компании, учитывая ее потребности в аналитике, масштабируемости и реальном времени, можно рекомендовать использование комбинации реляционных баз данных и NoSQL баз данных. Реляционные базы данных могут использоваться для хранения структурированных данных, таких как данные о продажах и клиентах, а NoSQL базы данных могут использоваться для хранения неструктурированных данных, таких как данные о поведении пользователей на сайте.

***2. Обоснуйте свой выбор, предоставив аргументы, почему данное хранилище данных является оптимальным для конкретных требований.***

Реляционные базы данных (например, MySQL, PostgreSQL) — это классический вариант хранения данных, который хорошо подходит для структурированных данных, таких как данные о продажах, клиентах и инвентаре. Они обладают высокой производительностью при выполнении сложных запросов и обеспечивают возможность создания сложных структур данных.

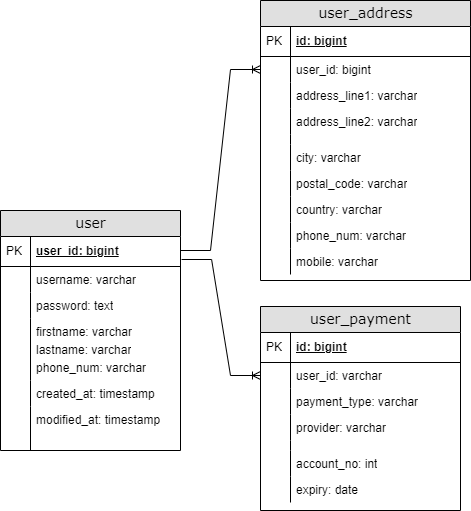
Поэтому, мы можем использовать MySQL БД для работы и хранения данных о пользователях ИС и технической информации веб-приложения (сайта платформы).

PostgreSQL имеет более высокую производительность при выполнении сложных запросов, чем MySQL. Это связано с тем, что PostgreSQL использует более эффективные алгоритмы выполнения запросов и имеет более оптимизированный движок базы данных. Кроме того, PostgreSQL поддерживает более широкий набор функций и возможностей, чем MySQL, что позволяет более гибко работать с данными. Мы можем использовать его для работы и хранения данных о заказах, продажах и т.п. А также, для обработки сложных аналитических запросов к данным для выявления тенденций продаж, сегментации клиентов и определения эффективности маркетинговых кампаний.

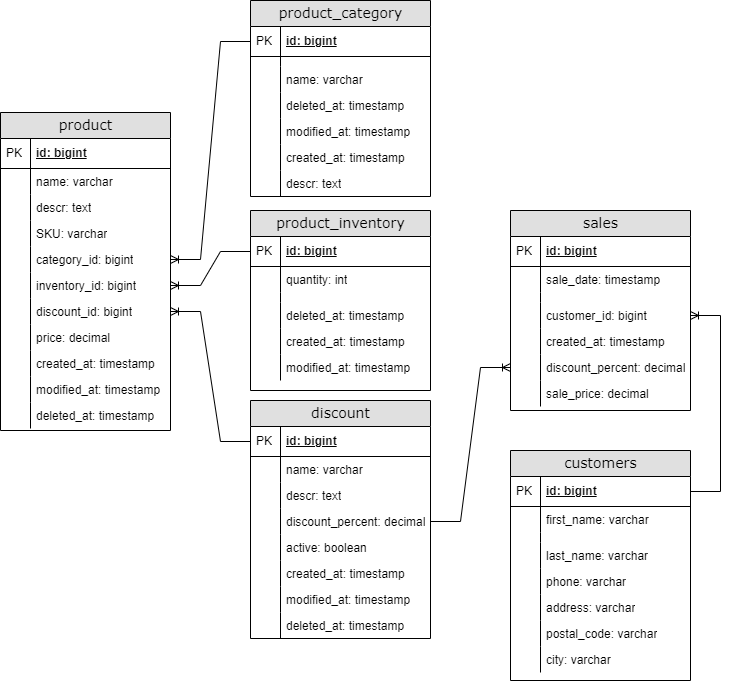
NoSQL базы данных (MongoDB) могут использоваться для хранения неструктурированных данных, таких как данные о поведении пользователей на сайте, BLOB-объектов (изображения и видео-файлы о товарах с метаданными).

***3. Опишите структуру данных, которые будут храниться в выбранном хранилище (таблицы, колонки и связи).***

MySQL – структура данных



PostgreSQL – структура данных



MongoDB – структура данных

Структура таблицы "behavior" в MongoDB для данных о поведении пользователей на сайте может выглядеть следующим образом:

{

"user\_id": {

"$type": "string",

"$minLength": 20,

"$maxLength": 255

},

"visits": {

"$type": "array",

"$minLength": 1,

"$maxLength": 255

},

"duration": {

"$type": "number",

"$minValue": 0,

"$maxValue": 6000000000000000000

},

"pageviews": {

"$type": "number",

"$minValue": 0,

"$maxValue": 6000000000000000000

},

"events": {

"$type": "array",

"$minLength": 1,

"$maxLength": 255

}

}

Эта структура таблицы позволяет хранить данные о поведении пользователей на сайте, включая идентификатор пользователя, количество посещений, продолжительность сессии, количество просмотров страниц и события, произошедшие во время сессии.

***4. Предложите схему интеграции данных из разных источников в хранилище.***

Для интеграции данных из разных источников MySQL, PostgreSQL и MongoDB в хранилище данных можно использовать следующую схему:

1. Создание таблиц в хранилище данных для каждого источника данных. Например, таблицу "sales" для данных о продажах, таблицу "customers" для данных о клиентах и таблицу "behavior" для данных о поведении пользователей на сайте.

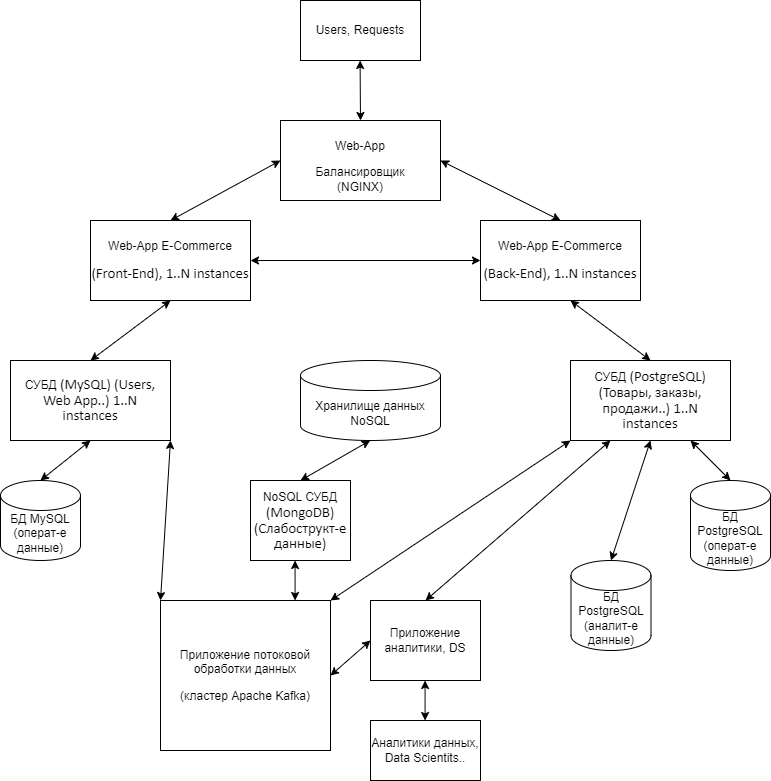
2. Загрузка данных из каждого источника данных в соответствующую таблицу в хранилище данных. Например, можно использовать инструмент ETL (Extract, Transform, Load) для автоматической загрузки данных из MySQL в таблицу "sales", из PostgreSQL в таблицу "customers" и из MongoDB в таблицу "behavior".

3. Создание связей между таблицами в хранилище данных. Например, можно создать связь между таблицей "sales" и таблицей "customers", чтобы связать данные о продажах и клиентах.

4. Использование инструментов для обработки данных в режиме реального времени. Например, можно использовать Apache Kafka для обработки больших объемов данных в режиме реального времени.

5. Использование инструментов для анализа данных. Например, можно использовать Python-библиотеки для анализа данных, такие как Pandas и NumPy, для анализа данных из таблиц в хранилище данных.

6. Использование инструментов для визуализации данных. Например, можно использовать библиотеки для визуализации данных, такие как Matplotlib и Seaborn, для визуализации данных из таблиц в хранилище данных.



***5. Опишите, как бы вы настроили систему для обработки данных в реальном времени.***

***Результат должен быть предоставлен в виде \*.docx файла со схемами***

Для настройки системы для обработки данных с помощью Apache Kafka для обработки больших объемов данных в режиме реального времени, можно использовать следующую схему:

1. Создание кластера Apache Kafka. Кластер Apache Kafka состоит из нескольких узлов, каждый из которых может обрабатывать сообщения. Для обработки больших объемов данных в режиме реального времени, рекомендуется использовать кластер с несколькими узлами.

2. Создание приложений для обработки данных. Приложения должны быть написаны на языке программирования, который поддерживается Apache Kafka (например, Java или Scala). Приложения должны принимать сообщения из кластера Apache Kafka и обрабатывать их в соответствии с требованиями.

3. Настройка схемы данных. Схема данных должна быть определена заранее, чтобы приложения могли правильно обрабатывать сообщения. Например, можно определить схему данных для таблицы "behavior" в MongoDB, как описано выше.

4. Настройка схемы сообщений. Схема сообщений определяет формат сообщений, которые передаются между узлами кластера Apache Kafka. Например, можно определить схему сообщений для таблицы "behavior" в MongoDB, как описано выше.

5. Настройка системы мониторинга. Система мониторинга должна быть настроена для отслеживания производительности приложений и узлов кластера Apache Kafka. Например, можно использовать инструменты мониторинга, такие как Prometheus и Grafana, для отслеживания производительности приложений и узлов кластера.

6. Настройка системы безопасности. Система безопасности должна быть настроена для защиты данных, передаваемых между узлами кластера Apache Kafka. Например, можно использовать инструменты безопасности, такие как OAuth и JWT, для аутентификации пользователей и защиты данных.