**Тестовое задание на позицию “Junior Data Scientist”**

**Задание №1**

Есть таблица с товарами: items

item\_id    name                 price       update\_date

-------    ----------------     ------      -----------

1          Ручка гелевая        10          2020-02-01

2          Карандаш 1HH         2           2020-01-01

1          Ручка шариковая      10          2020-03-01

3          Ластик               5           2020-07-01

2          Карандаш 1HH         3           2020-05-01

1          Ручка шариковая      5           2020-05-01

2          Карандаш 1H          7           2020-06-01

----------------------------------------------------------------

И таблица с заказами: orders

order\_id    user\_id     item\_id     order\_date

-------    --------     -------     ----------

1           1           1           2020-02-01

2           2           2           2020-02-01

3           1           3           2020-07-01

4           3           2           2020-07-01

5           2           1           2020-04-01

6           1           1           2020-06-01

**Написать запрос, который:**

**1.** Выведет актуальное состояние товаров на 2020-06-01;

**2.** Товары, купленные по цене больше или равно чем 3;

**3.** Сумму покупок клиента 1;

**4.** Сумма всех покупок до 2020-05-01 включительно;

**5.** Сумму всех заказов и среднее цена заказа поквартально;

**6.** Объясните, как вы будете оптимизировать запросы для больших объемов данных.

**1.** Выведет актуальное состояние товаров на 2020-06-01

```SQL

SELECT item\_id, name, price

FROM (

SELECT item\_id, name, price, update\_date,

ROW\_NUMBER() OVER (PARTITION BY item\_id ORDER BY update\_date DESC) as rn

FROM items

WHERE update\_date <= '2020-06-01'

) subquery

WHERE rn = 1;

```

**2.** Товары, купленные по цене больше или равно чем 3

```SQL

SELECT o.order\_id, o.user\_id, i.name, i.price

FROM orders o

JOIN items i ON o.item\_id = i.item\_id

WHERE i.price >= 3;

```

**3.** Сумму покупок клиента 1

```SQL

SELECT SUM(i.price) as total\_purchases

FROM orders o

JOIN items i ON o.item\_id = i.item\_id

WHERE o.user\_id = 1;

```

**4.** Сумма всех покупок до 2020-05-01 включительно

```SQL

SELECT SUM(i.price) as total\_purchases

FROM orders o

JOIN items i ON o.item\_id = i.item\_id

WHERE o.order\_date <= '2020-05-01';

```

**5.** Сумму всех заказов и среднее цена заказа поквартально

```SQL

SELECT

EXTRACT(QUARTER FROM o.order\_date) as quarter,

SUM(i.price) as total\_purchase,

AVG(i.price) as average\_purchase

FROM orders o

JOIN items i ON o.item\_id = i.item\_id

GROUP BY EXTRACT(QUARTER FROM o.order\_date);

```

**6.** Объясните, как вы будете оптимизировать запросы для больших объемов данных.

**Для оптимизации запросов можно использовать следующие подходы:**

* **Индексация**: Создание индексов на часто используемых столбцах, таких как **item\_id**, **user\_id**, **order\_date**, **update\_date**.
* **Избегание избыточных данных**: выбирать только те столбцы, которые действительно нужны для анализа.
* **Использование подзапросов и временных таблиц**: для сложных запросов можно использовать подзапросы или временные таблицы, чтобы разбить запрос на более простые части.
* **Оптимизация соединений**: использовать соединения, которые минимизируют количество обрабатываемых данных.
* **Кэширование результатов**: если данные не часто обновляются, можно кэшировать результаты запросов.
* **Параллелизация**: использовать параллельные вычисления для ускорения обработки больших объемов данных.

## **Задание №2**

У нас есть поток заказов, примерно **100 000 заказов в день**, которые записываются в таблицу `orders`.

**Эти данные переливаются в DWH, но есть сложность:**

- **В первые 6 месяцев** заказы активно изменяются (например, обновляется статус, состав заказа, стоимость и т. д.).

- **Далее в течение 2-5 лет** возможны редкие изменения.

- **Мы не можем использовать метку времени обновления** (`updated\_at` или аналоги).

Вопросы не подразумевают единственно верного варианта решения задачи.

**Ожидаем, что вы изложите ход мысли и подробно опишете решения, ответив на следующие вопросы:**

1. Как можно построить **архитектуру DWH** на основе **Data Vault 2.0**, чтобы:

    - **Обеспечить актуальность данных** (иметь "актуальную" таблицу заказов).

    - **Эффективно хранить изменения**, учитывая разные паттерны обновлений.

    - **Не терять историчность** (возможность "путешествовать" во времени).

2. Какие **сущности** вы выделите и какую роль они будут играть?

3. Как можно **обновлять актуальное состояние заказов** без использования `updated\_at`?

4. Как можно оптимизировать работу с данными такого рода?

**❏ ❐ ❑ ❒ ❏ ❐ ❏ ❐ ❑ ❒ ❏ ❐ ❑ ❒ ❑ ❒ ❑ ❐ ❑ ❒ ❏ ❐ ❏ ❐ ❑ ❒ ❏ ❐ ❑ ❒ ❑ ❒ ❑ ❏**

**1. Архитектура DWH на основе Data Vault 2.0**

**Data Vault 2.0** — это методология проектирования хранилищ данных, которая фокусируется на гибкости, масштабируемости и аудите данных.

**Для решения вашей задачи можно использовать следующую архитектуру:**

* **Hubs:** Центральные таблицы, которые хранят уникальные бизнес-ключи. Для заказов это будет таблица **order\_hub**, содержащая уникальные идентификаторы заказов.
* **Links:** Таблицы, которые хранят связи между бизнес-ключами. Например, **order\_customer\_link** для связи заказов с клиентами.
* **Satellites:** Таблицы, которые хранят атрибуты и изменения данных. Для заказов это будет **order\_satellite**, содержащая атрибуты заказа (например, статус, стоимость, состав заказа).
* **Effective Satellites:** Таблицы, которые хранят текущие значения атрибутов. Это позволит иметь "актуальную" таблицу заказов.

**2. Сущности и их роли**

* **Order Hub:** хранит уникальные идентификаторы заказов.
  + Роль: Основная сущность, обеспечивающая уникальность заказов.
* **Order Satellite:** хранит изменения атрибутов заказов.
  + Роль: Хранение истории изменений заказов.
* **Order Effective Satellite:** хранит текущие значения атрибутов заказов.
  + Роль: Обеспечение актуальности данных.
* **Order-Customer Link:** хранит связи между заказами и клиентами.
  + Роль: Обеспечение связей между сущностями.

**3. Обновление актуального состояния заказов**

Для обновления актуального состояния заказов без использования **updated\_at** можно использовать следующий подход:

* **Инкрементальная загрузка данных:** периодически (например, раз в день) загружать новые и измененные заказы в DWH.
* **Сравнение данных:** сравнивать новые данные с текущими в **order\_satellite**. Если данные изменились, обновлять **order\_satellite** и **order\_effective\_satellite**.
* **Использование триггеров:** настроить триггеры в базе данных, которые будут отслеживать изменения в таблице **orders** и обновлять соответствующие записи в DWH.

**4. Оптимизация работы с данными**

* **Партиционирование таблиц:** разделить таблицы на партиции по дате или другим критериям для ускорения доступа к данным.
* **Индексация:** создать индексы на часто используемых столбцах, таких как **order\_id**, **customer\_id**.
* **Архивирование старых данных:** перемещать старые данные (например, старше 5 лет) в архивные таблицы для уменьшения объема активных данных.
* **Кэширование:** использовать кэширование для часто запрашиваемых данных.
* **Параллельная обработка:** использовать параллельную обработку данных для ускорения загрузки и обновления данных.

Эти подходы помогут обеспечить актуальность данных, эффективно хранить изменения и сохранять историчность данных в **DWH**.

* **ФИО:** Дуплей Максим Игоревич
* **TG:** @quadd4rv1n7
* **Дата:** 06.03.2025