Национальный исследовательский ядерный университет «МИФИ»

(Московский Инженерно-Физический Институт)

Кафедра №42 «Криптология и кибербезопасность»

**Лабораторная работа №4**

**«Переход на PostgreSQL»**

Тимин Александр Б21-515 (2024г.)

**1 Заполнение**

Учитывая специфику предметной области и возможности PostgreSQL создадим базу данных и необходимый набор таблиц (листинг запросов приведен в разделе [«Приложение»](#application)).

Также заполним таблицы информацией о нескольких известных сериалах (листинг запросов приведен в разделе [«Приложение»](#application)). На рисунке 1.1 приведена часть данных, которые в дальнейшем будут использоваться в запросах.

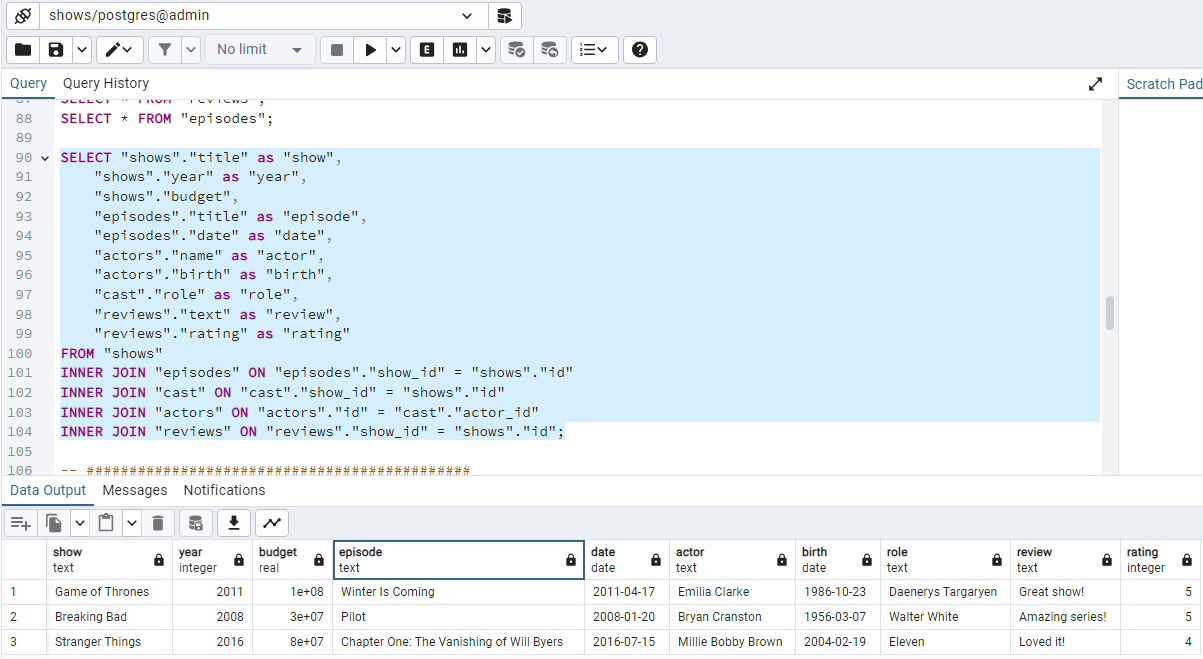


Рисунок .1 - данные.

Здесь и далее при демонстрации базы данных используется GUI pgAdmin 4.

**2 Запросы**

Предположим, что сотрудники компании, которая предоставляет информацию о сериалах, хотят знать ответы на следующие вопросы:

1. Какие актеры снялись в сериале "Game of Thrones"?
2. Какие сериалы были выпущены после 2010 года и их бюджеты?
3. Какие отзывы получили сериалы с рейтингом выше 4?
4. Какова общая сумма бюджета всех сериалов?
5. Какие актеры родились после 1980 года?
6. Какие сериалы имели эпизоды, вышедшие после 2015 года?
7. Какие актеры сыграли роли в сериале с наивысшим бюджетом?
8. Сколько эпизодов было в каждом сериале?

На рисунках 1-8 представлены соответствующие запросы к базе данных и ответы на них.

Ссылка на соответствующий листинг запросов приведен в разделе [«Приложение»](#application).

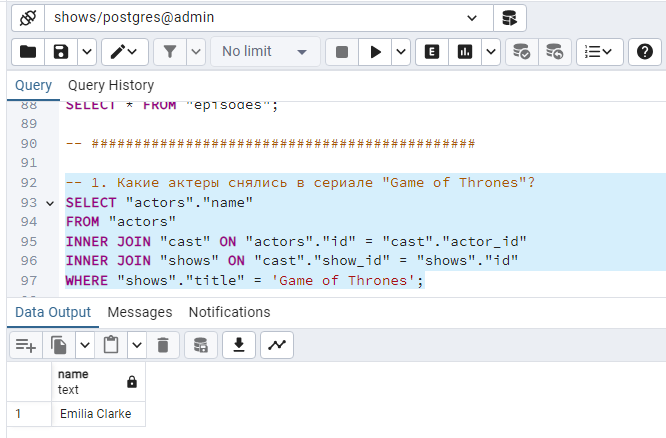


Рисунок 2.1 - получение списка актеров, снявшихся в сериале "Game of Thrones".

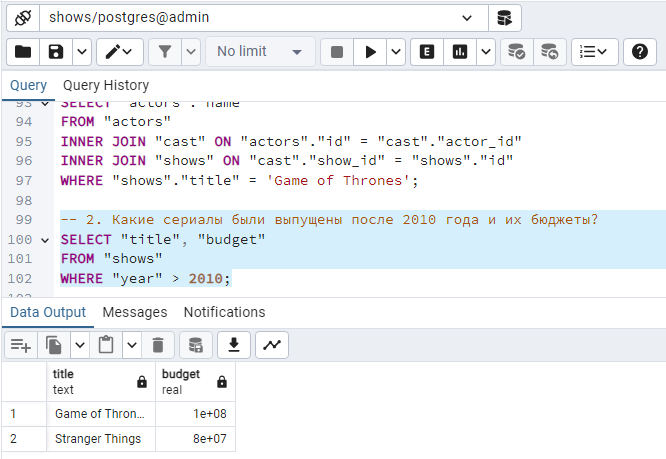


Рисунок 2.2 - получение списка сериалов, выпущенных после 2010 года и их бюджеты.

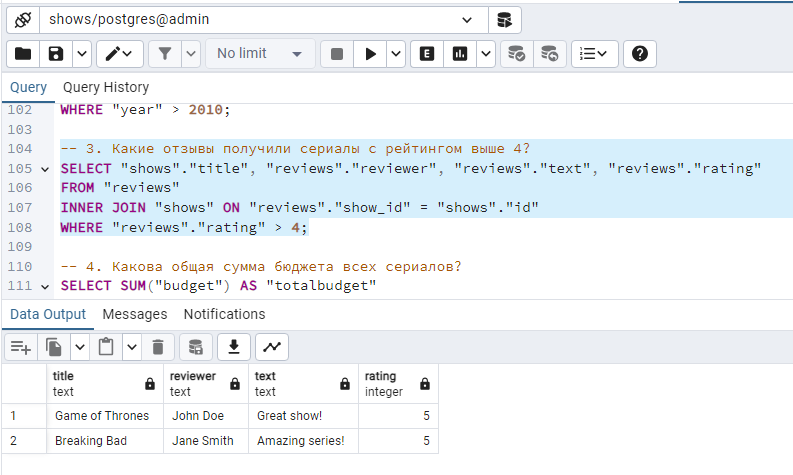


Рисунок 2.3 - получение отзывов, полученных сериалами с рейтингом выше 4.

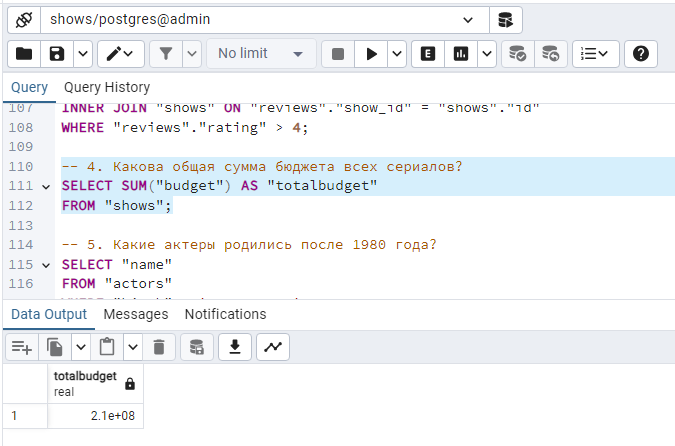


Рисунок 2.4 - получение суммы бюджета всех сериалов.

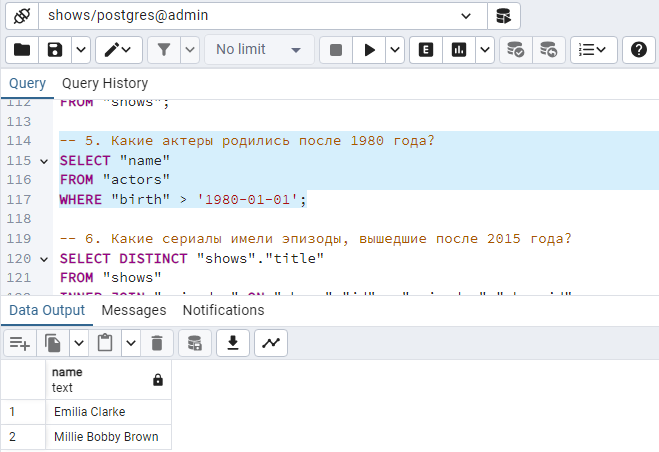


Рисунок 2.5 - получение списка актеров, рожденных после 1980 года.

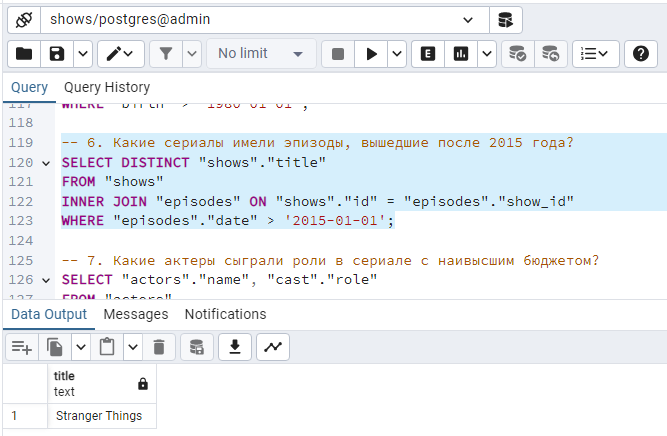


Рисунок 2.6 - получение списка сериалов, имеющих эпизоды, вышедшие после 2015 года.

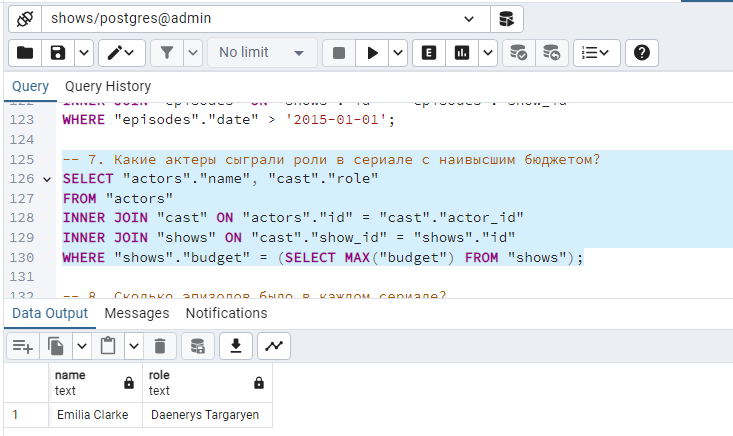


Рисунок 2.7 – получение списка актеров, сыгравших роли в сериале с наивысшим бюджетом.

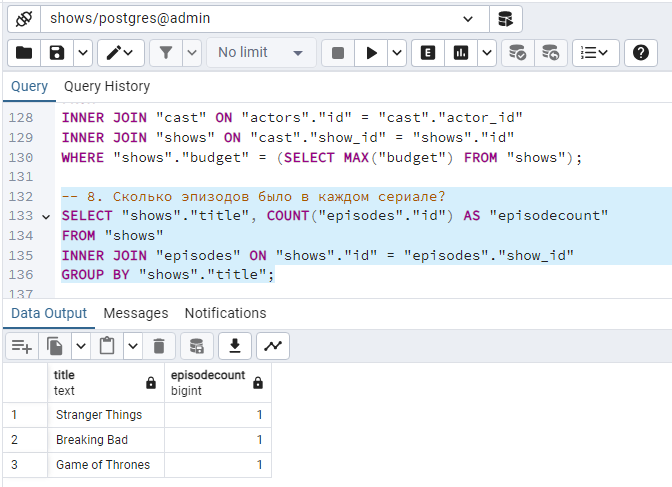


Рисунок 2.8 – получение количества эпизодов в каждом сериале.

**3 Сравнение PostgreSQL и SQLite**

1. *Жесткая типизация данных.* PostgreSQL не позволяет хранить в одном столбце данные разных типов. SQLite позволяет хранить данные разных типов в одном столбце.
2. *Поддержка сложных запросов.* PostgreSQL поддерживает более сложные запросы и функции, такие как оконные функции и рекурсивные CTE.
3. *Совместный доступ к базе данных.* PostgreSQL поддерживает многопользовательский доступ и управление ролями, в то время как SQLite больше ориентирована на встраиваемые приложения и поддерживает только сериализованный доступ.
4. *COLLATION.* PostgreSQL поддерживает коллации и позволяет задавать порядок сортировки строк в соответствии с выбранной локалью, что позволяет корректно сортировать строки на различных языках. SQLite ограничивается побайтовой сортировкой, что не всегда корректно для всех языков.
5. *Транзакции.* PostgreSQL поддерживает сложные транзакции с возможностью отката (ROLLBACK) и фиксации (COMMIT) в рамках транзакционных блоков, что позволяет обеспечивать целостность данных на высоком уровне. SQLite также поддерживает транзакции, но имеет меньше возможностей для их настройки и управления.
6. *Агрегатные функции и оконные функции.* PostgreSQL поддерживает широкий набор агрегатных и оконных функций, которые могут быть использованы для анализа данных, таких как вычисление скользящего среднего, ранжирование и т.д. SQLite поддерживает ограниченный набор таких функций.
7. *Работа с датами и временем.* PostgreSQL имеет мощные встроенные функции для работы с датами и временем, включая поддержку часовых поясов, интервалов и сложных операций с датами. SQLite имеет ограниченные возможности для работы с датами и временем.

**Заключение**

В ходе работы в соответствии с предметной областью была создана и заполнена база данных сериалов, к ней было выполнено несколько составных запросов. Также было проведено сравнение PostgreSQL и SQLite.

**На защиту**

На рисунке 1 – скриншот сообщения с заданиями на защиту.

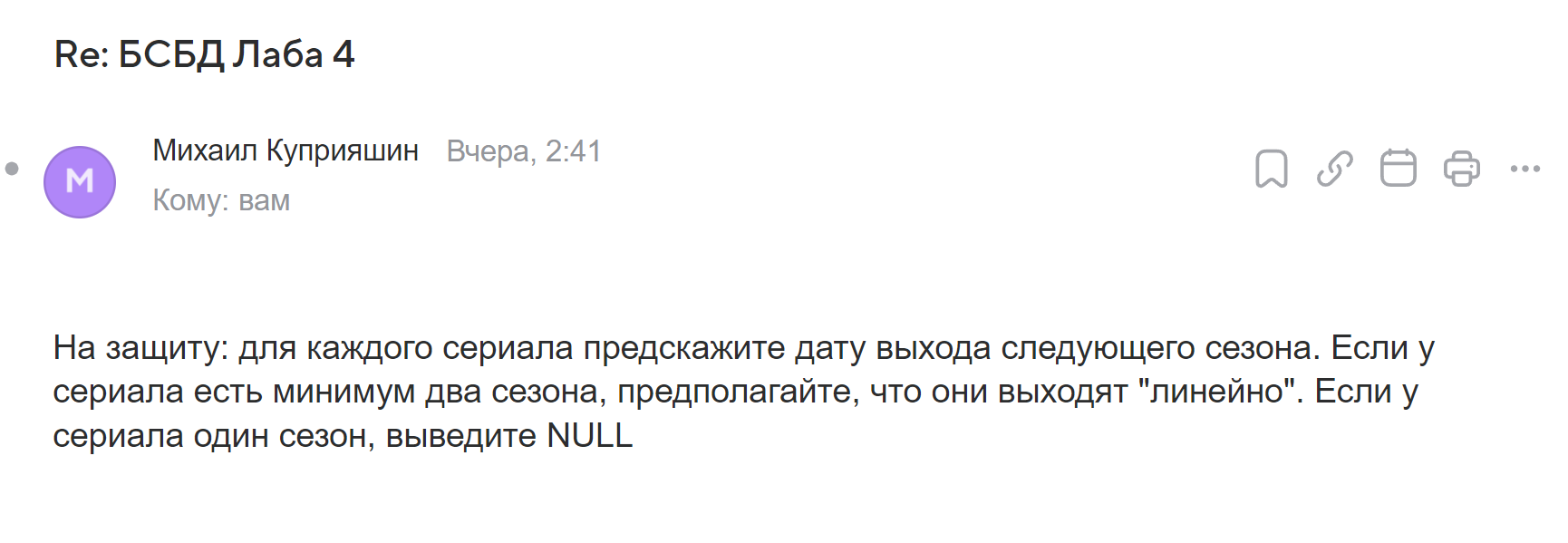


Рисунок 1 - задания на защиту.

Для начала необходимо обновить структуру таблиц. На рисунке 2 – ERD новой структуры.

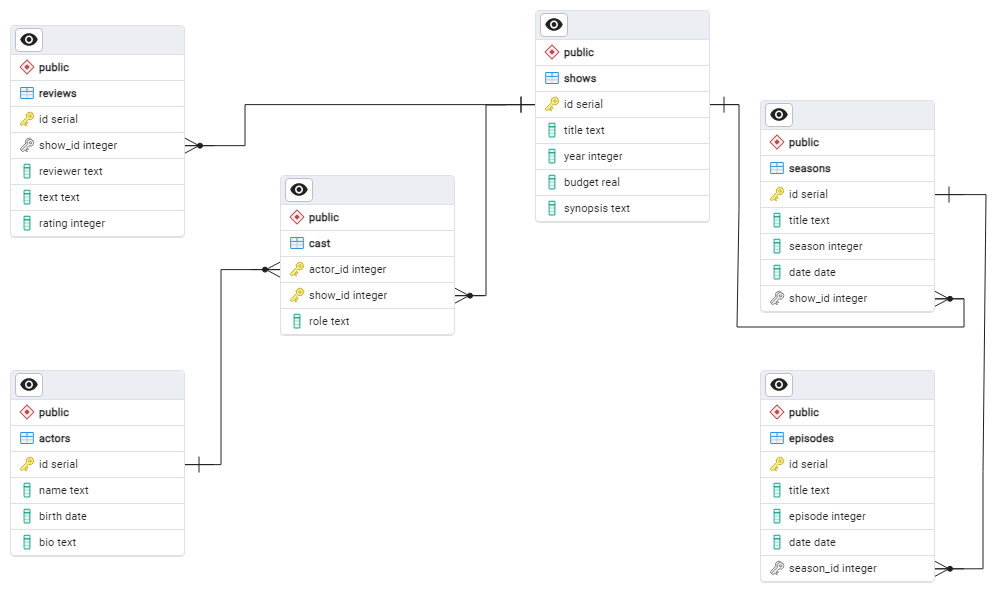


Рисунок - ERD обновленной структуры таблиц.

Основная идея запроса – использование функции LEAD. Оконная функция LEAD позволяет нам получить значение следующей строки в упорядоченном наборе данных. На рисунке 3 – данный запрос и ответ на него.

Ссылка на соответствующие листинг запросов приведен в разделе [«Приложение»](#application).

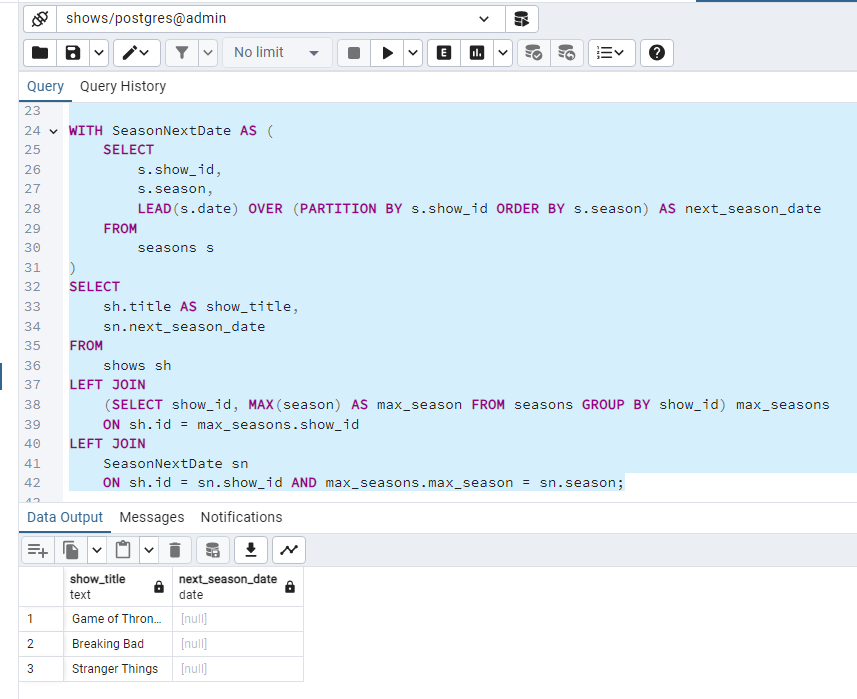


Рисунок - определение даты выхода следующего сезона.

**На защиту**

На рисунке 4 – скриншот сообщения с заданиями на защиту.

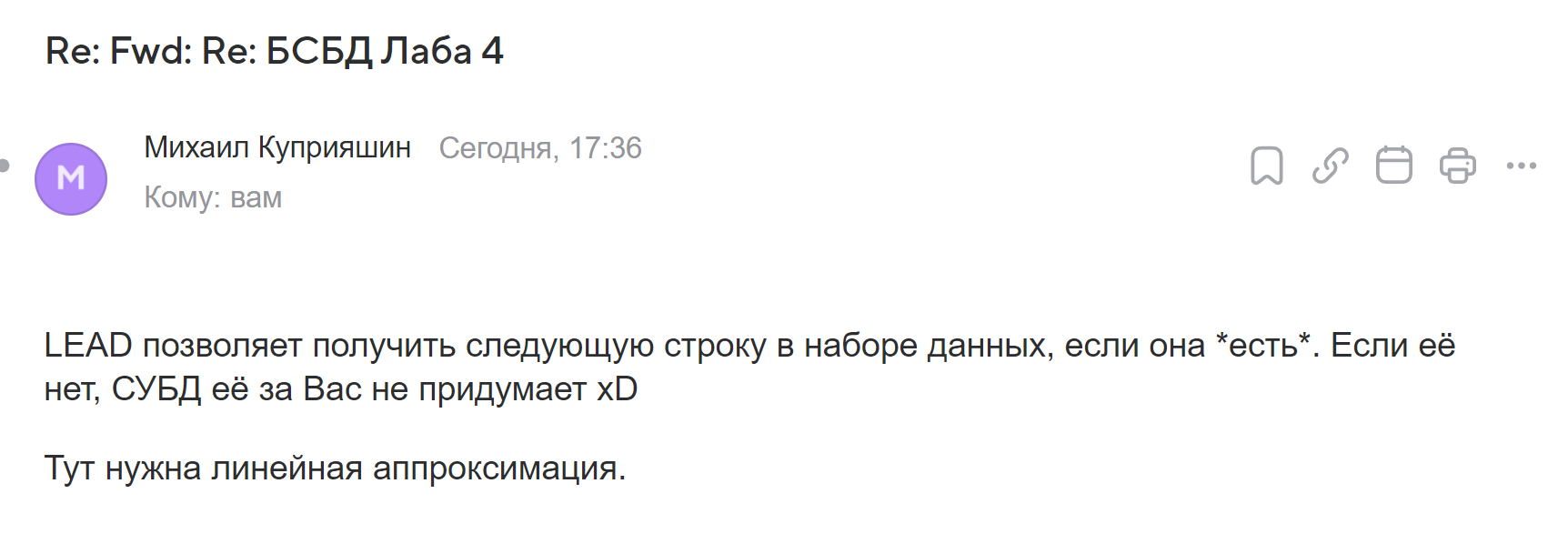


Рисунок - задание на защиту.

Не без помощи всем известной нейросети вспомнил, как делать линейную аппроксимацию.

Изменил данные в таблицах (так как в порыве научных изысканий, я там что-то сломал).

Касаемо самого запроса данных: воспользовавшись каскадом временных таблиц и объединением двух запросов сконструировал квазимодо, результат которого представлен на рисунке 5.

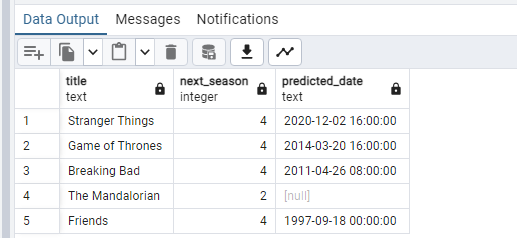


Рисунок - результат запроса.

Приложение

* [SQL-скрипт создания таблиц](https://github.com/KATEHOK/DBS_labs-2024/blob/main/4/create.sql);
* [SQL-скрипт заполнения таблиц](https://github.com/KATEHOK/DBS_labs-2024/blob/main/4/new_insert.sql);
* [SQL-скрипт обновления структуры таблицы](https://github.com/KATEHOK/DBS_labs-2024/blob/main/4/alter.sql);
* [SQL-скрипт получения данных из таблиц](https://github.com/KATEHOK/DBS_labs-2024/blob/main/4/new_new_requests.sql);
* [ERD обновленной структуры таблиц](https://github.com/KATEHOK/DBS_labs-2024/blob/main/4/pics/ERD.png);
* [обновленный отчет (docx)](https://github.com/KATEHOK/DBS_labs-2024/blob/main/4/report.docx);
* [обновленный отчет (pdf)](https://github.com/KATEHOK/DBS_labs-2024/blob/main/4/report_altered_altered.pdf).