**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ**

**ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» (ДГТУ)

Факультет «Информатика и вычислительная техника»

Кафедра «Кибербезопасность информационных систем»

**Лабораторная работа № 4**

на тему «работа с темпоральными БД в СУБД PostgreSQL»

|  |
| --- |
| Выполнил: студент группы ВКБ43 |
| Ковалев Данил Петрович |
| (Фамилия, имя, отчество) |
| Проверил: |
|  |
| (Фамилия, имя, отчество) |

Оглавление

[Цель 3](#_Toc179639729)

# **Цель**: исследовать и получить навыки в создании, изменении и применении темпоральных баз данных в СУБД PostgreSQL.

# **Ход работы**: создать темпоральную БД и приложение реализующее отображение данных на текущую или на любую другую временную отметку:

Для создания базы данных я использовал Docker compose, конфигурация запуска бд и работы заданий представлено на рисунке 1.

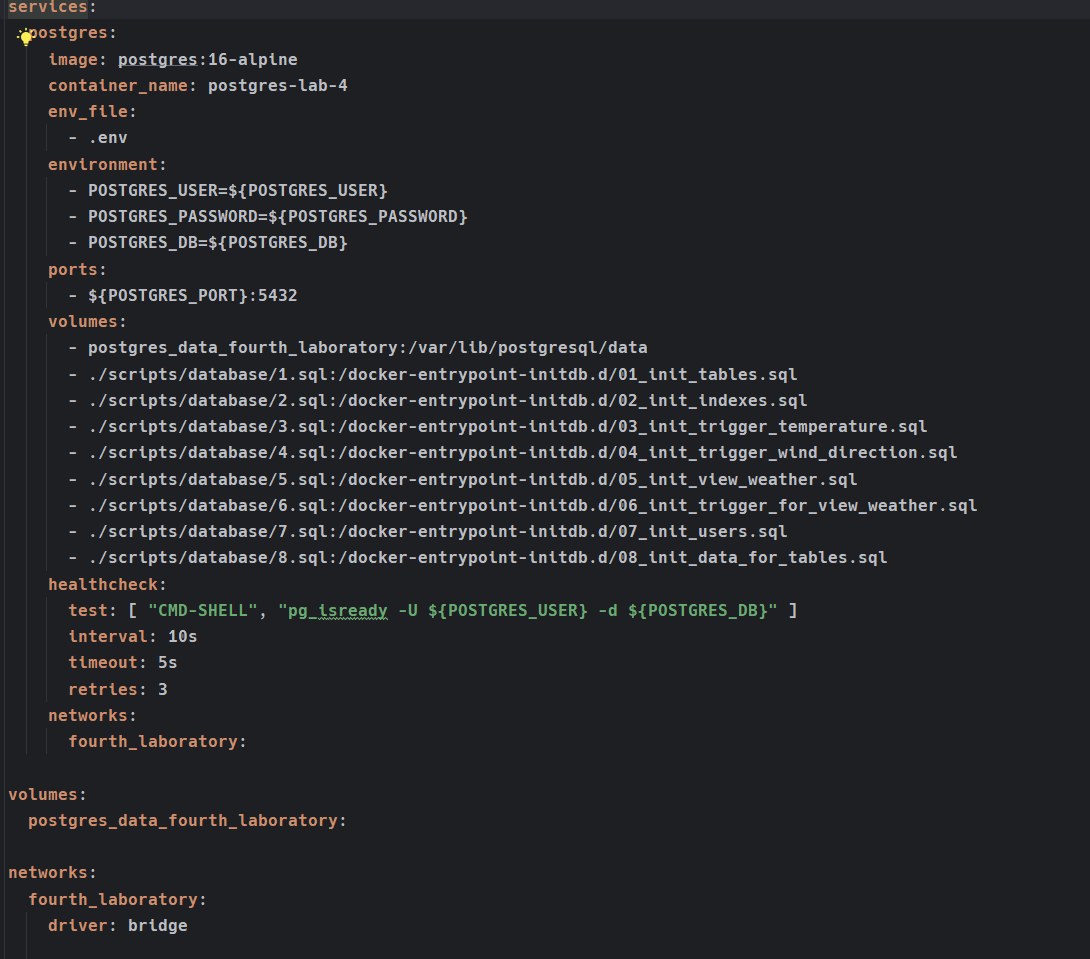


Рисунок 1 – docker compose для создания базы данных

**Задание 1.** Реализовать базовое отношение temporary\_object, содержащее атрибуты: id – идентификатор записи, тип: integer; time\_create – время создания записи (время рождения), тип timestamp; time\_dead – время смерти записи, тип timestamp.

При реализации данного задания был использован такой скрипт, который представлен на рисунке 2.

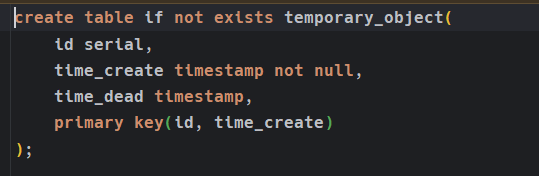


Рисунок 2 – реализация объекта temporary\_object

**Задание 2**. Создать таблицы, для которых планируется реализовать свойство темпоральности, объявив их наследниками temporary\_object;

Были реализованы такие вот таблицы, которые представлены ниже на рисунке 3. Предметная область – метеостанция для сохранения информации о температуре, направления ветра и города.

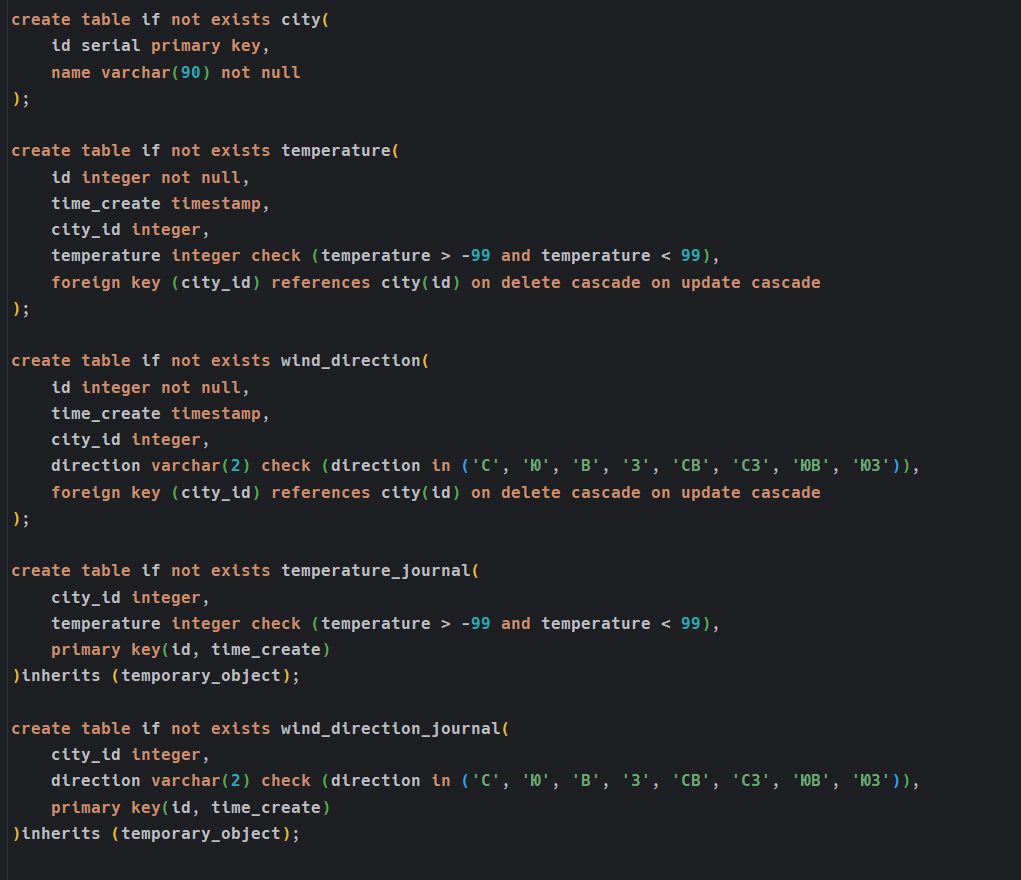


Рисунок 3 – таблицы базы данных

**Задание 3**. Для всех ключевых полей (как первичных ключей, так и внешних) создать индексы командой CREATE INDEX.

Для создания индексов использовался следующий скрипт, который представлен ниже на рисунке 4. Здесь мы используем обычные BTree индексы, специфических данных не содержится, чтобы использовать иные типы индексов.

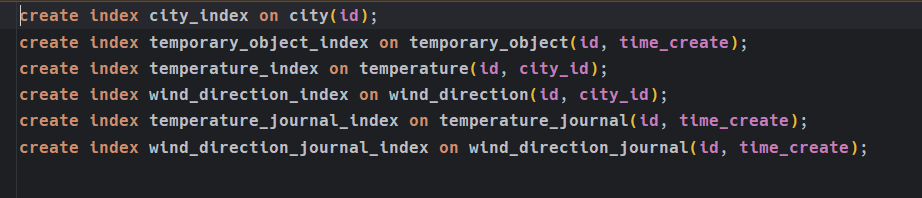


Рисунок 4 – создание индексов для базы данных

**Задание 4**. Определить триггерную функцию(-и), которая(-ые) при добавлении новой записи в отношение, поддерживающее свойство темпоральности, проверяет наличие такой записи среди существующих объектов (сравнение по id). Если запись с таким id найдена, то для этой записи устанавливается время смерти, а добавляемая запись содержит в себе соответственные значения атрибутов найденной записи (кроме тех, которые заданы для новой записи), пустое значение времени смерти и новое значение времени рождения, равное текущему моменту. Таким образом, у новой записи время рождения равно, времени смерти ее предка. Если запись с таким id на найдена, то происходит ее добавление в БД с установлением в качестве времени рождения текущего момента времени.

Был реализовано несколько триггеров. Первый триггер, представленный на рисунке 5, позволяет отслеживать прошлое состояние температур и если были изменения, то вносить во временную таблицу базы данных.

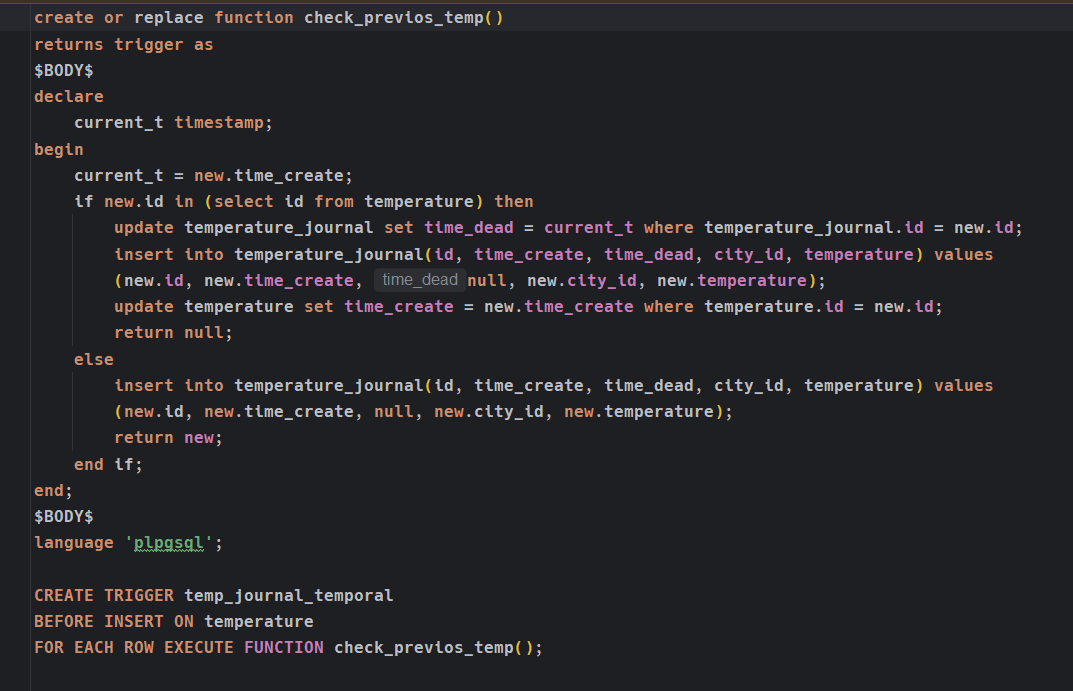


Рисунок 5 – триггерная функция для проверки изменения температуры

Рассмотрим ещё один триггер, который проверяет изменилась ли запись ветра. Если были изменения, то записывает в темпоральную таблицу. Сниппет программы представлен на рисунке 6.

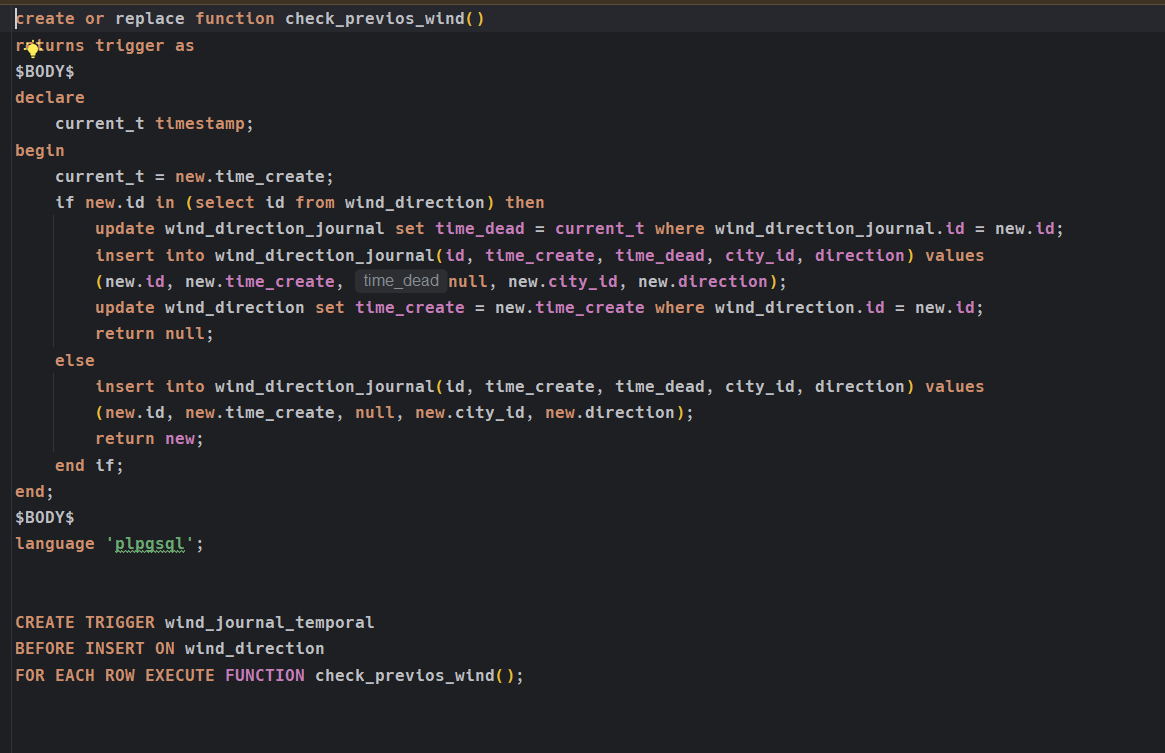


Рисунок 6 – триггерная функция для проверки изменения направления ветра

**Задание 5.** Создание нескольких пользователей в базе данных.

Для создания нескольких пользователей был использован скрипт, который представлен ниже на рисунке 7. Можно отметить, что мы создали нескольких пользователей, выдали права на подключение к базе данных. Для читателя был выдан доступ к операциям SELECT, а для editor были выданы права на операции UPDATE, SELECT.

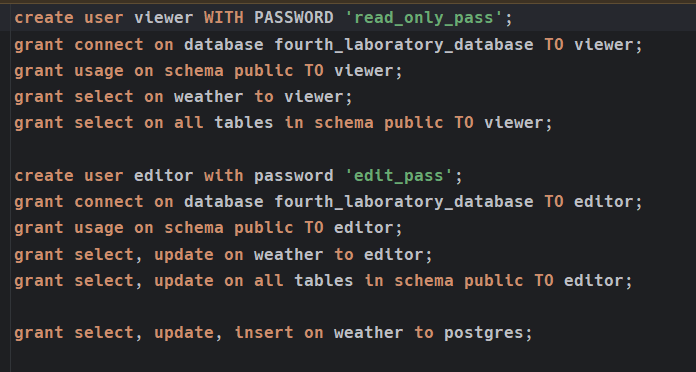


Рисунок 7 – создание пользователей и выдача прав

**Задание 6**. Создать представление, объединяющее и представляющее данные из нескольких темпоральных таблиц.

Было создано представление weather, которое предоставляет возможность отображать данные из нескольких таблиц. View представлен на рисунке 8.

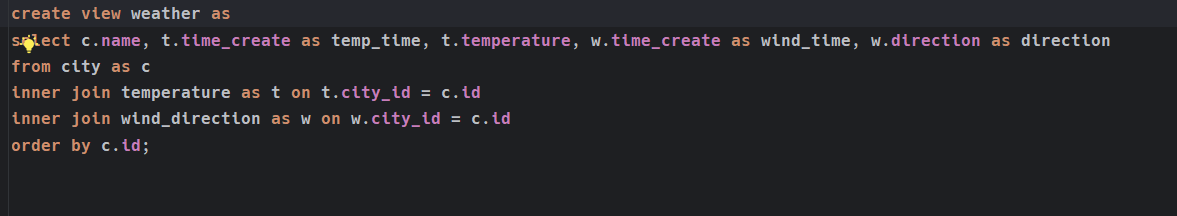


Рисунок 8 – представление для отображения данных

**Задание 7**. Реализовать разграничение прав доступа для пользователей с использованием представления: один пользователь только читает; другой может читать и изменять данные, но только в одной из таблиц представления (для реализации изменения данных через представление использовать INSTEAD OF триггер); администратор (postgres) может читать и изменять данные в любой из таблиц представления.

Для реализации обновление View использовался следующий триггер, который представлен на рисунке 9.

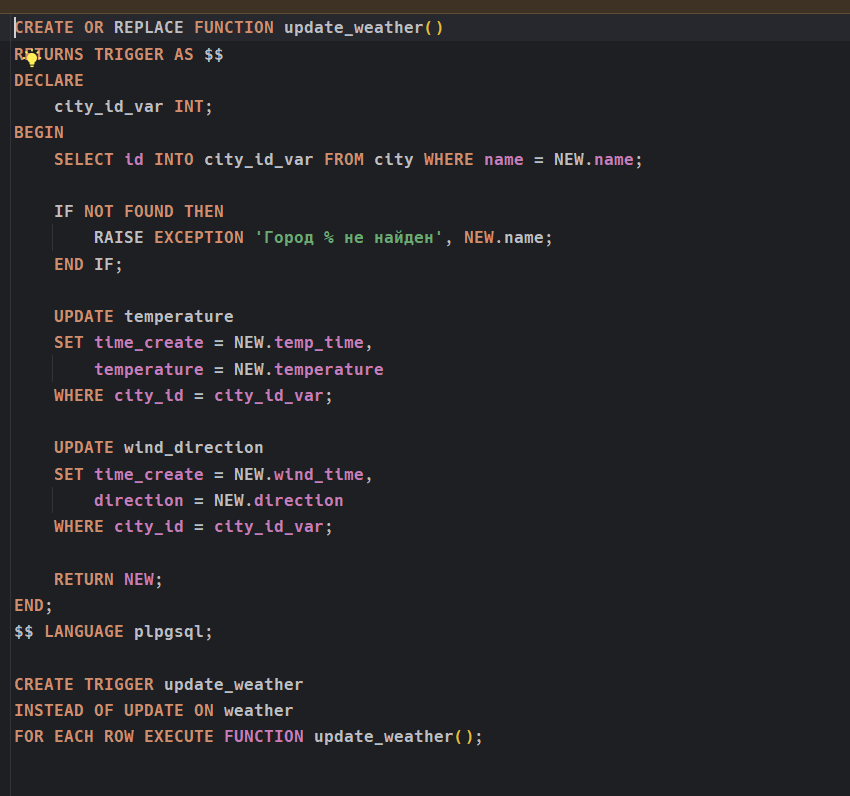


Рисунок 9 – триггер для обновления данных в View

Для реализации приложения, которое умеет отображать данные использовалось несколько языков программирования Python и Javascript. Python выступает в качестве бэкенд клиента и взаимодействия с базой данных, javascript используется для отображения UI. На рисунке 10 представлены библиотеки и фреймворки, которые используются на стороне backend.

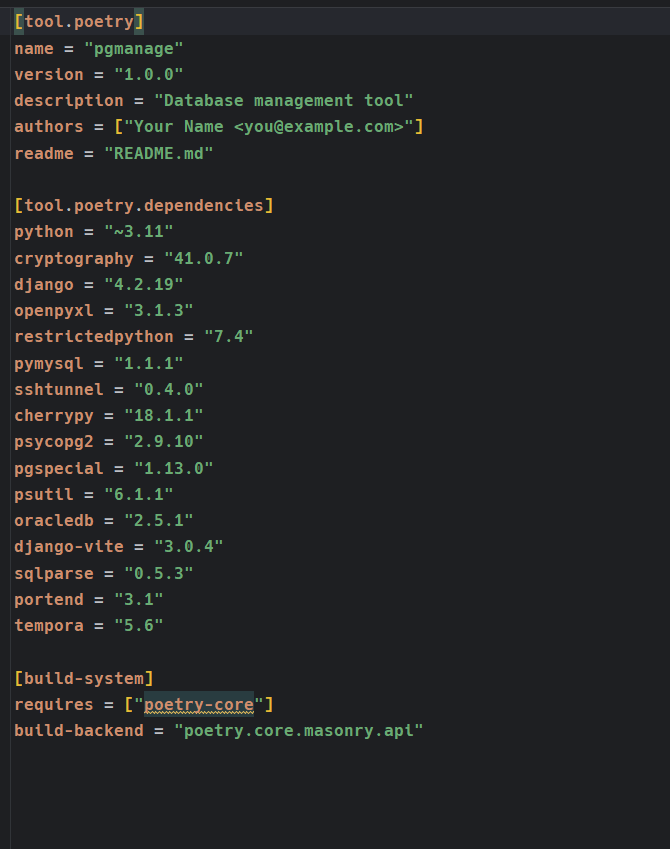


Рисунок 10 – используемые библиотеки для написания backend

При написании frontend используется следующие библиотеки, которые представлены ниже на рисунке 11.

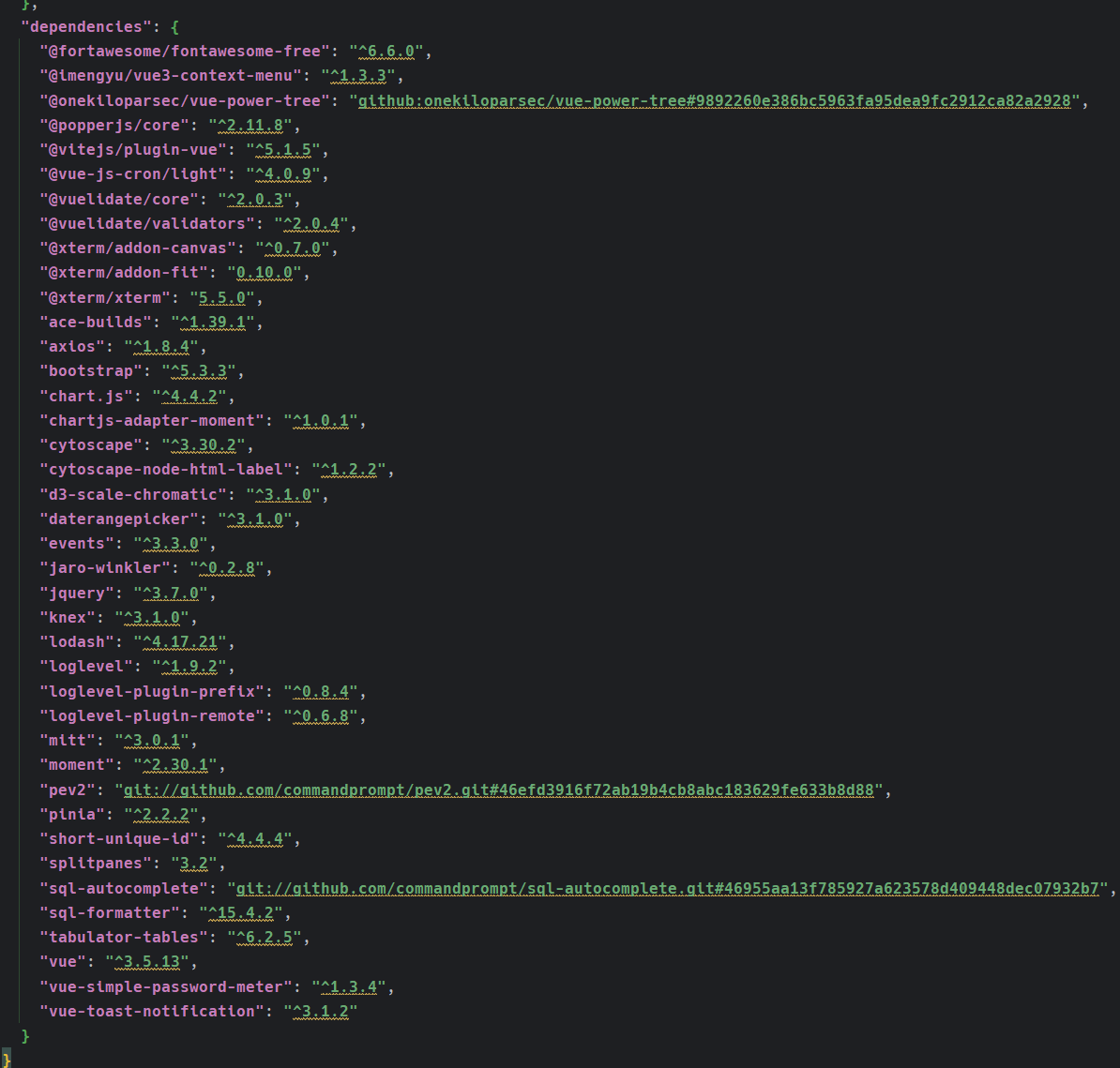


Рисунок 11 – библиотеки и фреймворки для frontend

Данное приложение является Web приложением для взаимодействия с базой данных. Ниже представлю рисунок 12, где показана страница аутентификации.

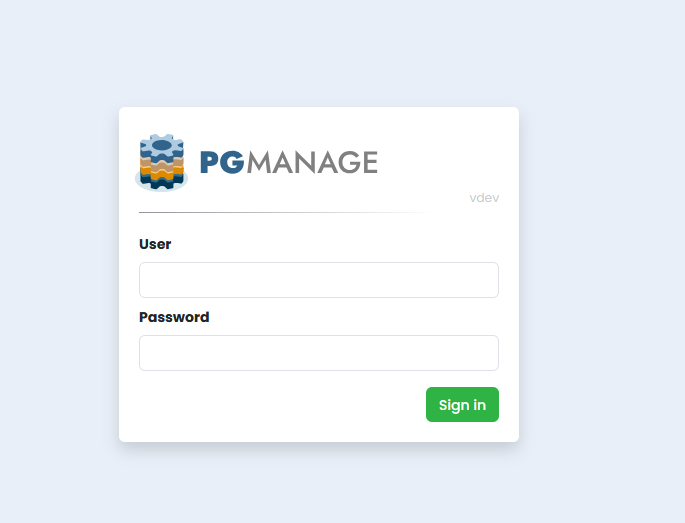


Рисунок 12 – страница аутентификации

Далее при работе нам нужно зарегистрировать нашу базу данных, чтобы создался WebSocket, который будет держать соединение с нашей базой данных. На рисунке 13 представлен интерфейс для регистрации базы данных.

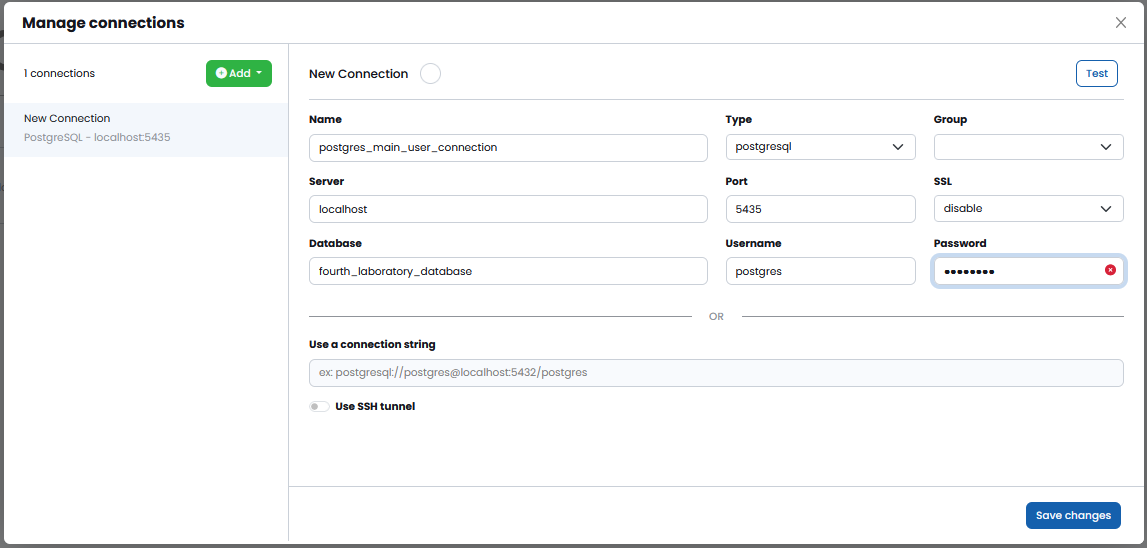


Рисунок 13 – страница для подключения к базе данных

Все действия, проделанных с прошлых заданий применяются, как миграции к базе данных, в следствии чего они уже были применены и вручную ничего не придется делать.

Попробуем войти за пользователя viewer, показав, что у него есть только возможность просмотра данных таблиц, но нет возможностей вносить правки в таблицы. На рисунке 14 показано, что у него есть доступ на просмотр, а на рисунке 15 видно, что база данных вернула Permission denied.

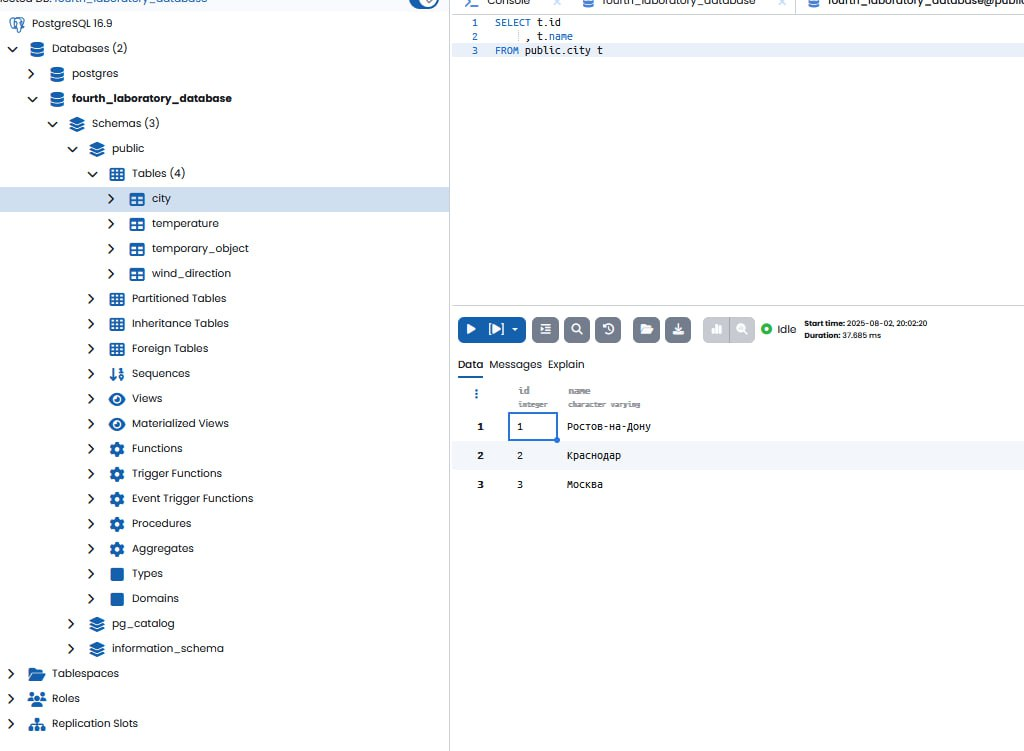


Рисунок 14 – результат SELECT запроса для просмотра содержимого базы данных



Рисунок 15 – попытка вставки данных из-под пользователя viewer

Попробуем теперь просмотреть, что пользователь может просматривать отображение weather.

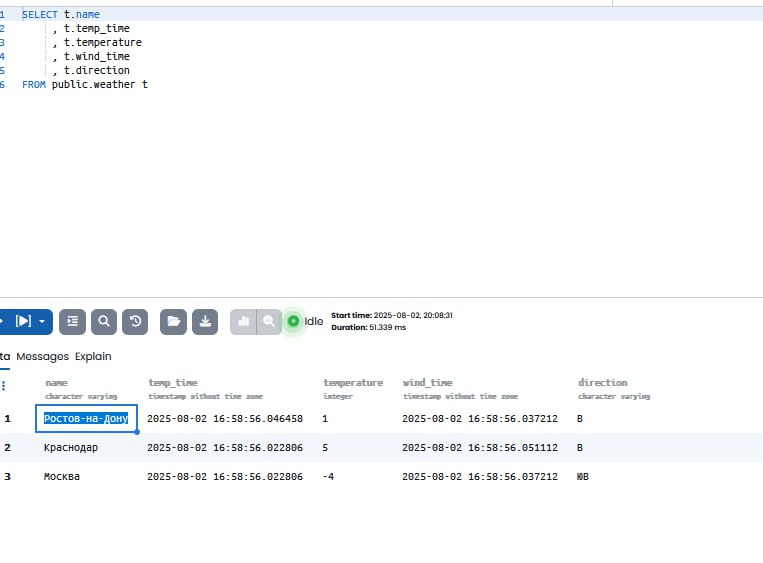


Рисунок 16 – отображение данных в View

Важно учитывать тот момент, что нельзя разрешить пользователю просматривать только View. Если запретить просмотр хотя бы 1 из таблиц, которые используется во View, то сразу же будет и блокировка View для пользователя. Такая особенность была замечена, но вполне логична. Так что нельзя сделать идеальное разграничение прав только с помощью View.

Попробуем теперь зайти за пользователя editor и удостовериться, что у нас есть возможность изменять данные во View и других местах. Выполним запрос, который представлен на рисунке 17.



Рисунок 17 – запрос на обновление View

В результате теперь View отображает наши обновленные данные, которые представлены ниже на рисунке 18.

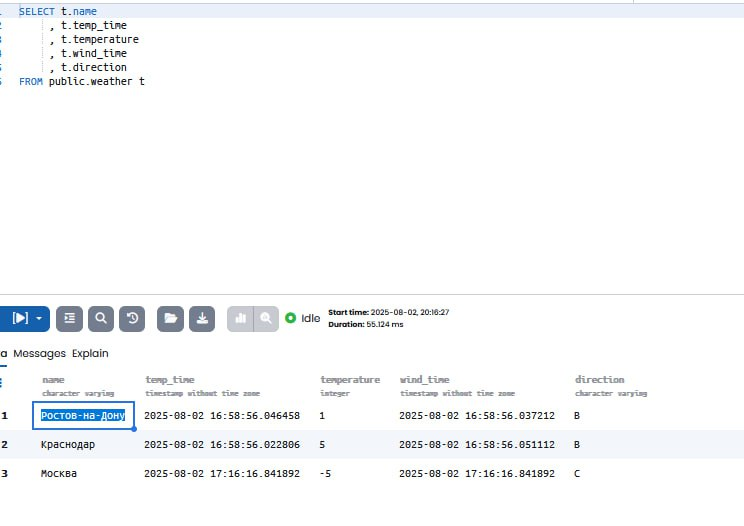


Рисунок 18 – обновление данных в View