Министерство науки и высшего образования Российской Федерации   
Федеральное государственное автономное образовательное  
учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

**Отчет по лабораторной работе №1**

**Разработка базы данных**

**для приложения “Планировщик задач”**

Выполнила

Студент группы 6305-010302D

Радаева Евгения Денисовна

**Самара 2025**

Цель работы

Развернуть базу данных, разработать ORM-модели и реализовать скрипты для управления данными.

Основные задачи

1. Настройка базы данных

* Развернуть PostgreSQL в Docker;
* Подключиться к БД через DBeaver.

Для развертывания PostgreSQL в Docker был создан файл docker-compose.yml (правая часть рисунка 1).

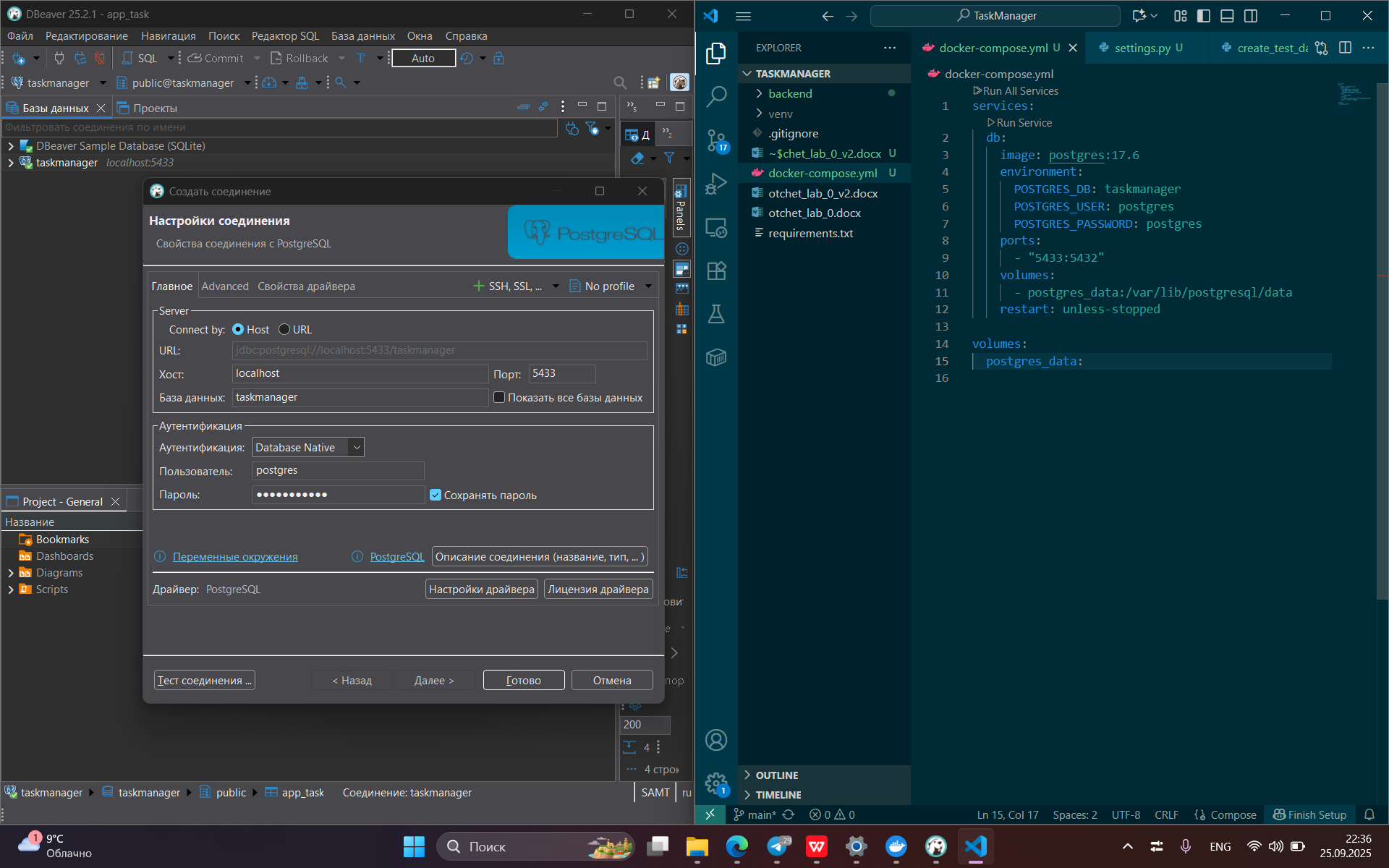


Рисунок 1 – Создание docker-compose.yml (справа) и подключение к базе данных PostgreSQL через DBeaver (слева)

Затем в командной строке с помощью команды

*docker compose up -d*

был запущен контейнер с образом PostgreSQL (верхняя часть рисунка 2). С помощью команды

*docker compose ps*

было проверено, произошел ли запуск и работает ли контейнер (нижняя часть рисунка 2), о чем говорит значение “Up” в столбце “STATUS” в строке с нужной БД “taskmanager”.

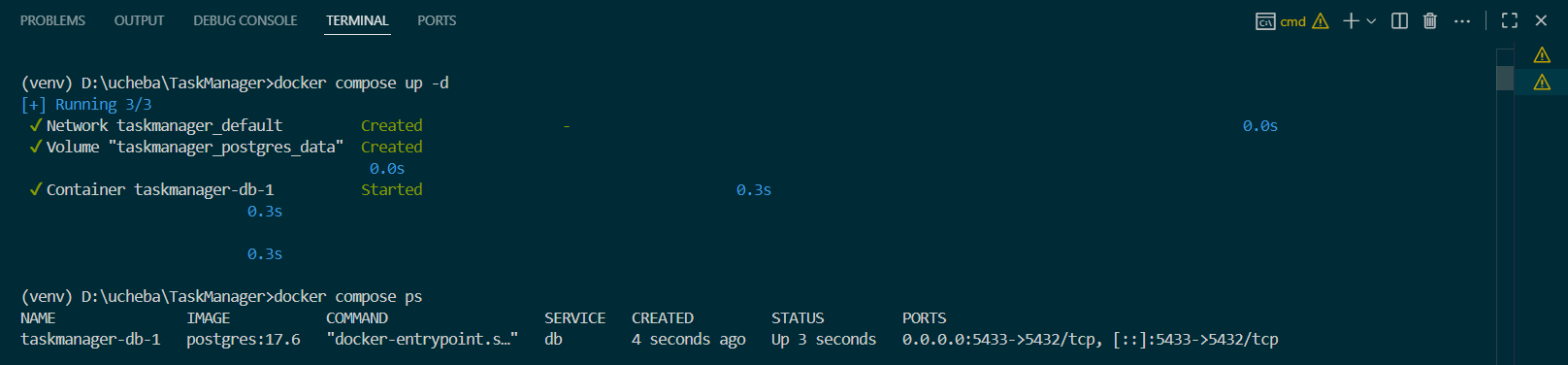


Рисунок 2 – Запуск контейнера с БД и проверка успешности запуска

После запуска контейнера с PostgreSQL было налажено подключение к БД с помощью DBeaver (левая часть рисунка 1).

Параметры подключения:

Host: localhost

Port: 5433

Database: taskmanager

User: postgres

Password: postgres

2. Разработка ORM-моделей

* Определить сущности и связи между ними:

В рамках проекта были разработаны 3 основные сущности:

* User;
* Task;
* Category.

А также установлены связи между ними:

* User 1:N Task (один пользователь имеет много задач)
* User 1:N Category (один пользователь имеет много категорий)
* Task N:M Category (многие-ко-многим)
* Реализовать модели с использованием Django ORM:

Для представления данных в БД была использована Django ORM, модели с нужными полями и связями были определены в файле models.py приложения.

* Настроить миграции (для Python-проектов):

Для создания таблиц в БД была выполнена настройка миграций (рисунок 3) с помощью команд

*python manage.py makemigrations*

*python manage.py migrate*

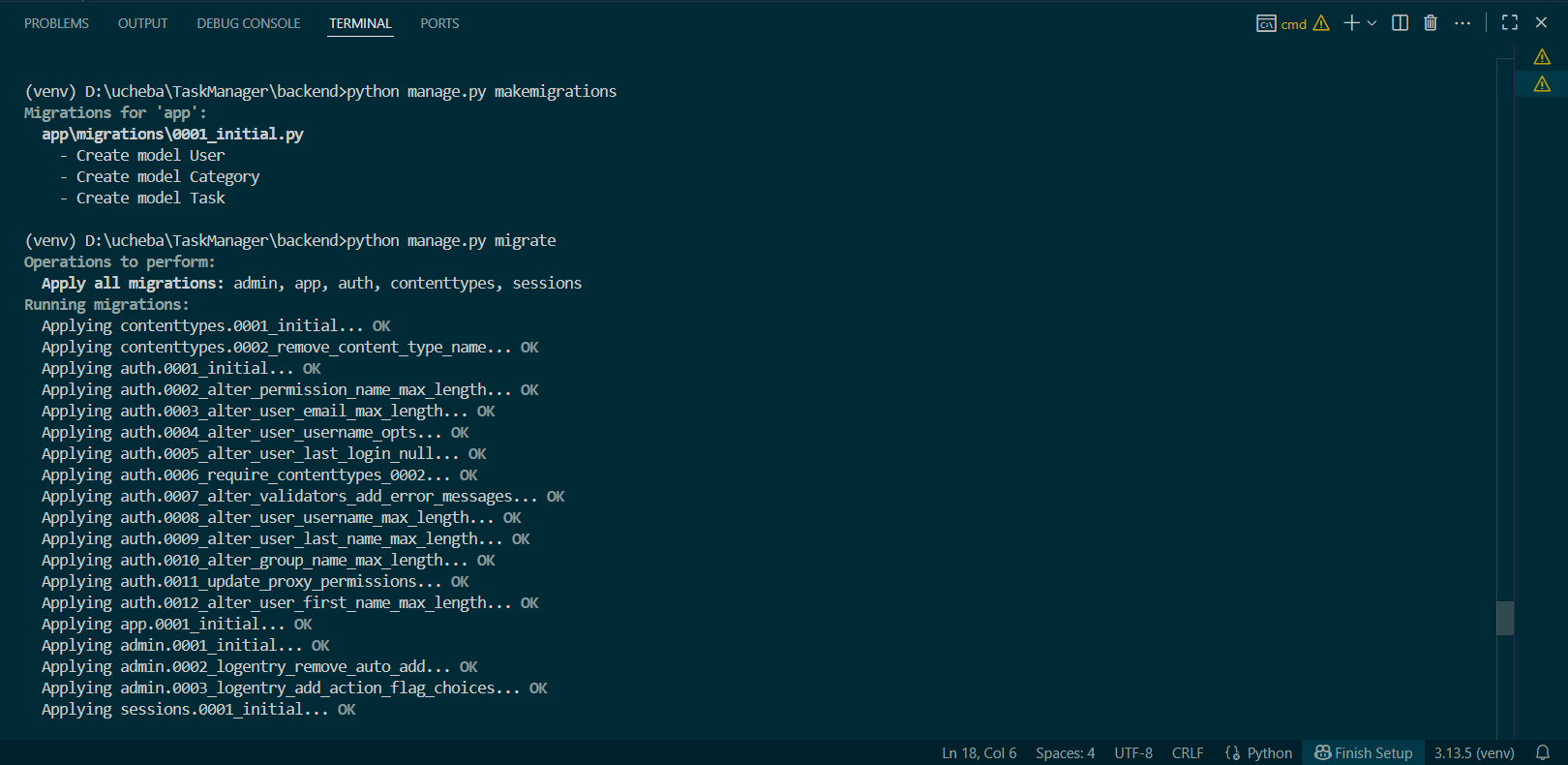


Рисунок 3 – Настройка и запуск миграций

После выполнения миграций в БД были созданы 12 таблиц (рисунок 4).

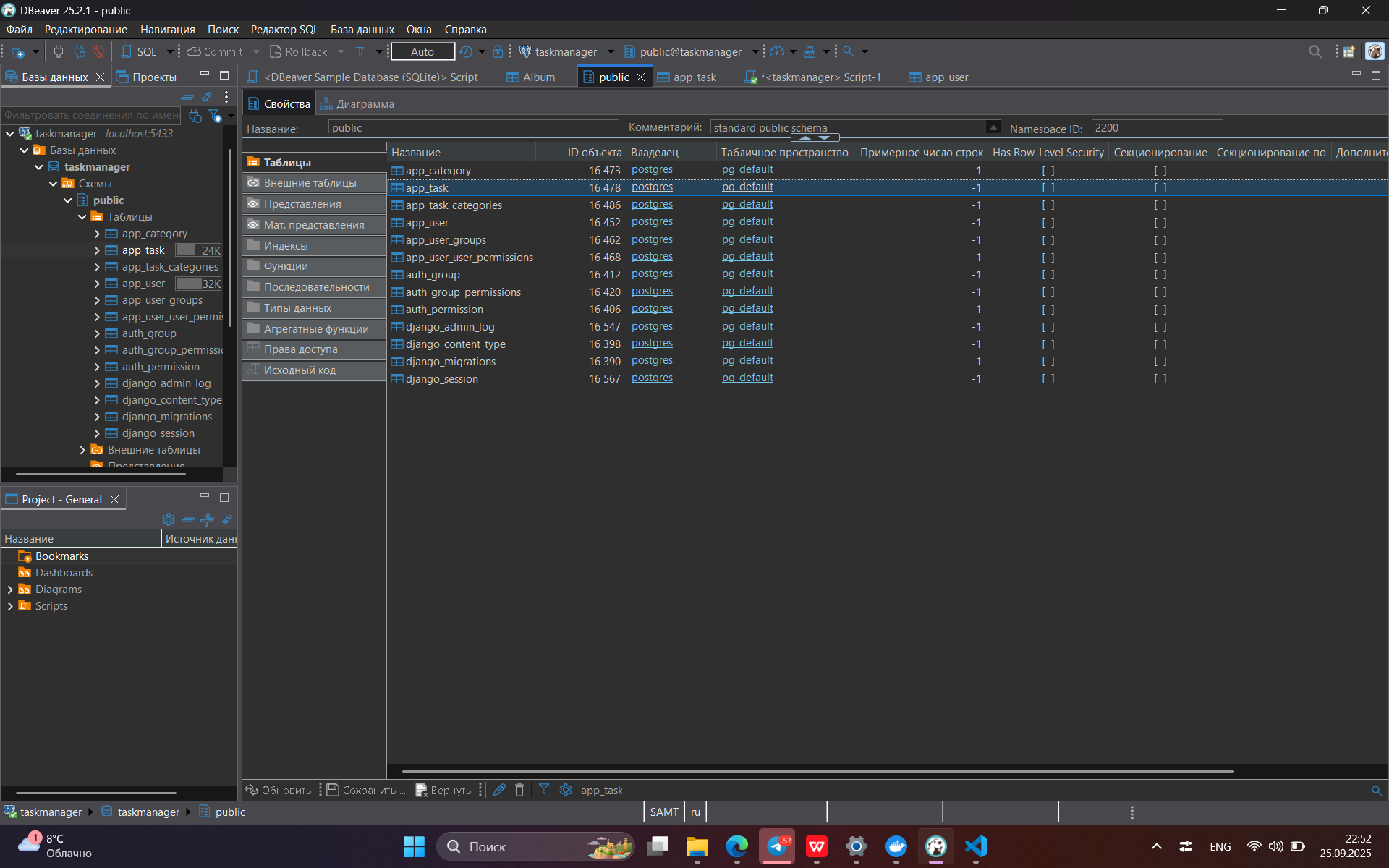


Рисунок 4 – Список таблиц после миграций

Получившаяся структура БД:

* app\_category – категории задач;
* app\_task – задачи;
* app\_task\_categories – связь многие-ко-многим для задач и категорий;
* app\_user – пользователи;
* app\_user\_groups – группы пользователей;
* app\_user\_user\_permissions – права пользователей;
* auth\_\* – системные таблицы аутентификации Django;
* django\_\* – системные таблицы Django.

3. Разработка модели пользователя

* Определить структуру модели (email, пароль, имя и т. д.):

Была создана модель пользователя User, унаследованная от AbstractUser, которая включает следующие поля:

* Username;
* Email;
* Password;
* First\_name;
* Last\_name;
* Time\_zone.
* Настроить хеширование паролей:

Хеширование паролей было настроено с помощью встроенных средств Django, которые автоматически хешируют пароль с использованием алгоритма PBKDF2 с SHA256.

4. Функционал для работы с моделями

* Написать скрипты для заполнения базы данных тестовыми данными;
* Реализация функционала для работы с данными в соответствии с тематикой выбранного приложения:

Для заполнения БД тестовыми данными был создан скрипт create\_test\_data.py. Его запуск (рисунок 5) осуществлялся с помощью команды

*python create\_test\_data.py*

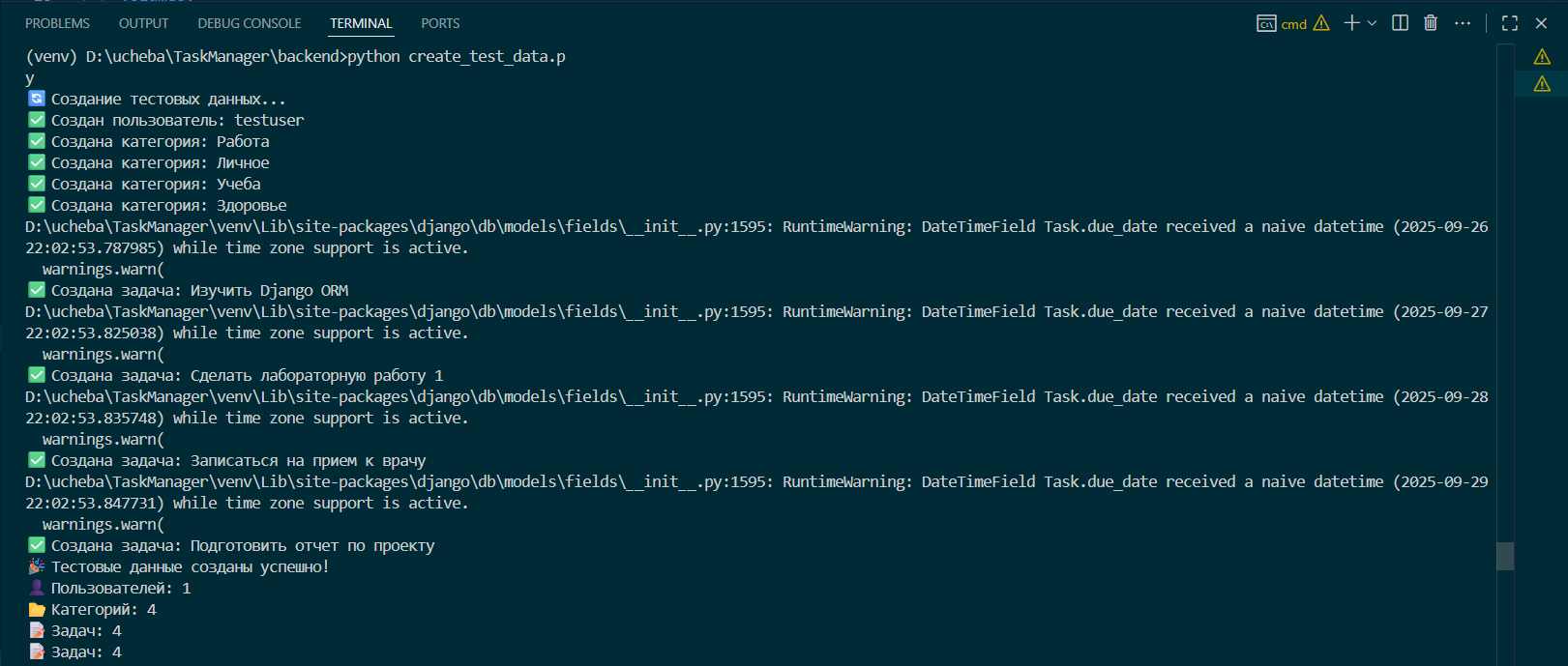


Рисунок 5 – Работа скрипта по заполнению БД тестовыми данными

В результате работы скрипта в БД произошли изменения:

* В таблице с пользователями появился 1 новый пользователь (рисунок 6):

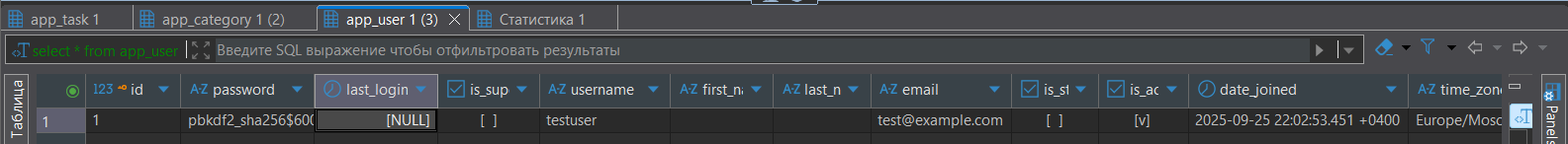


Рисунок 6 – Таблица с пользователями после выполнения скрипта

* В таблице с категориями появилось 4 новых категории (рисунок 7):

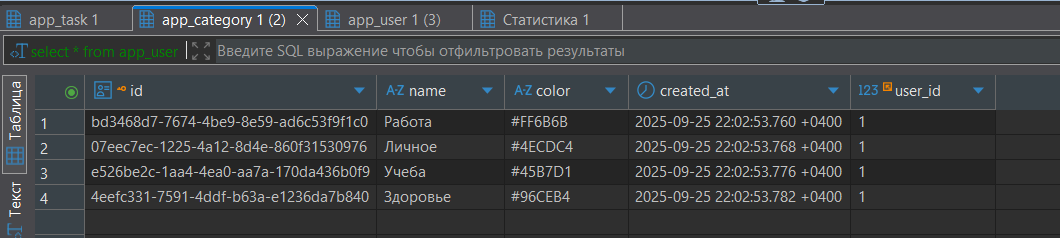


Рисунок 7 – Таблица с категориями после выполнения скрипта

* В таблице задач появилось 4 новые задачи (рисунок 8):

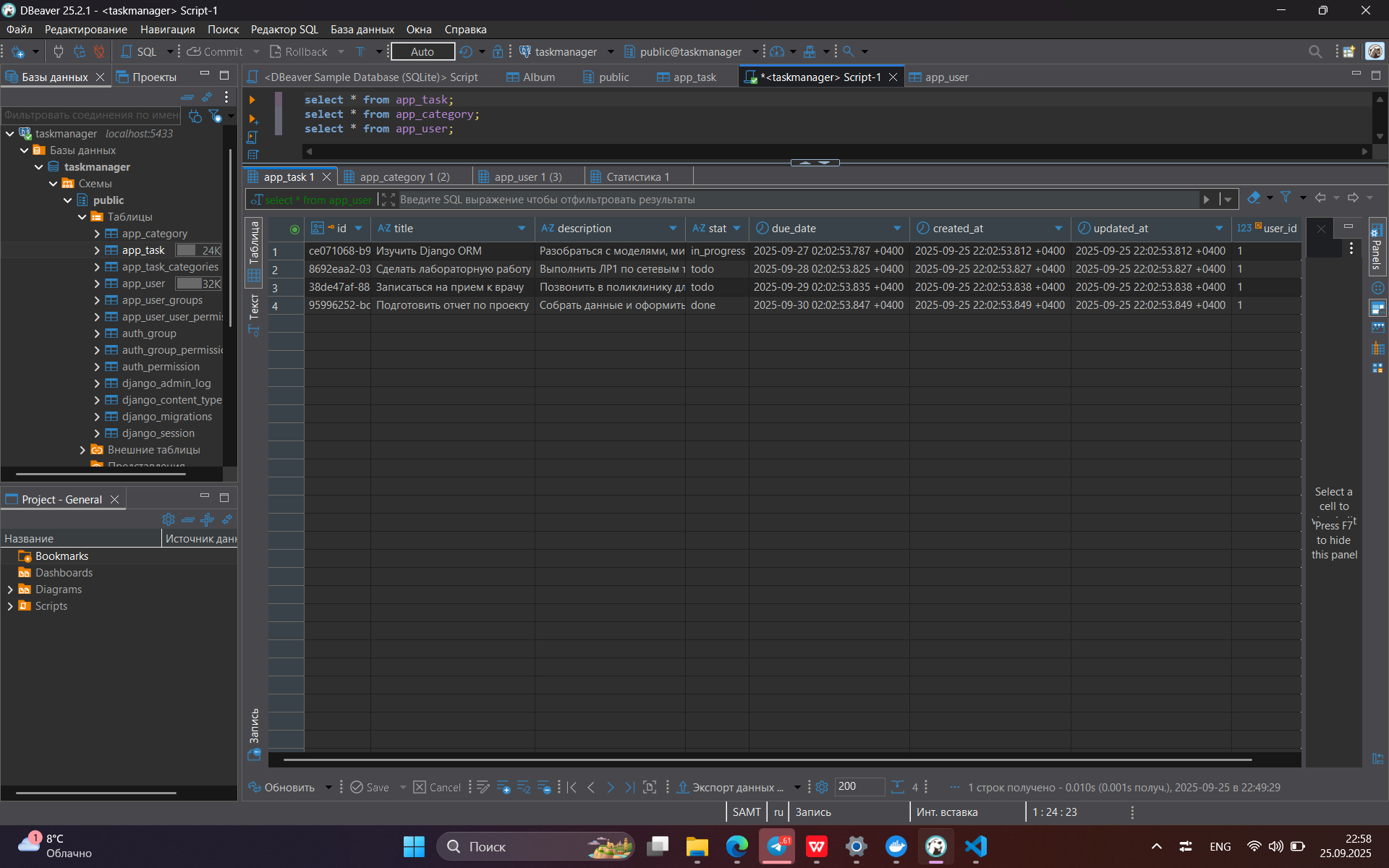
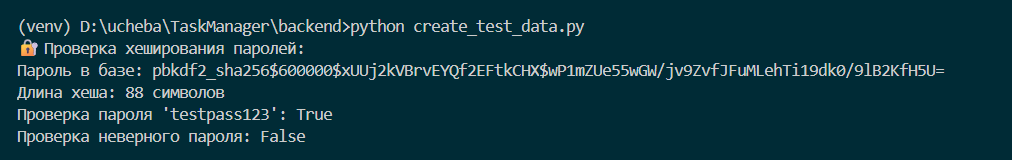


Рисунок 8 – Таблица с задачами после выполнения скрипта

Итоги

В ходе лабораторной работы была успешно разработана и развернута база данных PostgreSQL в Docker для приложения “планировщик задач”. Было выполнено подключение к БД с помощью DBeaver, были разработаны и реализованы с помощью DjangoORM ORM-модели, выполнены миграции. Реализован скрипт для заполнения БД тестовыми данными.