Министерство науки и высшего образования Российской Федерации

ФГАОУ ВО «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий-РТФ

ОТЧЕТ

По лабораторной работе №1

«Основы работы с Docker и PostgreSQL»

По дисциплине «Разработка приложений»

Выполнил: Гуламов А.С.

Группа: РИМ-150950

Проверил преподаватель: Кузьмин Д.

Екатеринбург

2025

**Цель работы**: Освоить фундаментальные концепции и базовые операции Docker: создание образов, запуск контейнеров, управление ими, работа с сетями и томами. На практике закрепить навыки, запустив изолированную базу данных PostgreSQL и подключившись к ней извне.

**Задачи:**

1. Установить и проверить работу Docker.
2. Изучить базовые команды Docker.
3. Запустить контейнер с PostgreSQL в изолированном режиме.
4. Запустить контейнер с pgAdmin и подключить его к контейнеру с БД через сеть Docker.
5. Подключиться к БД из pgAdmin, создать схему и выполнить запросы.
6. Обеспечить сохранность данных БД с помощью томов Docker.

**Ход работы:**

**1. Установка и проверка Docker**

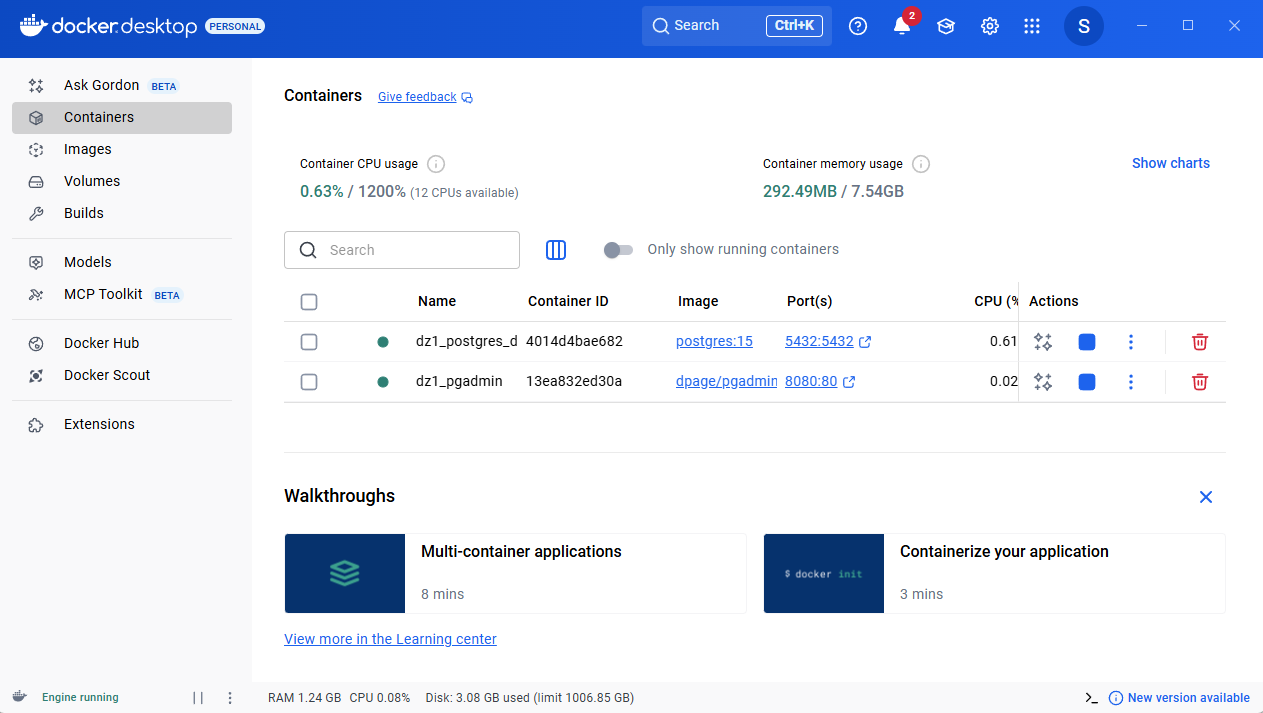
С официального сайта был установлен Docker Desktop (Рисунок 1).

Рисунок 1 – Docker Desktop

Далее, запустив Visual Studio Code и вписав в терминале команды «docker version», «docker compose version» и «docker hello-world», убедились в правильной работе Docker (Рисунки 2-3).

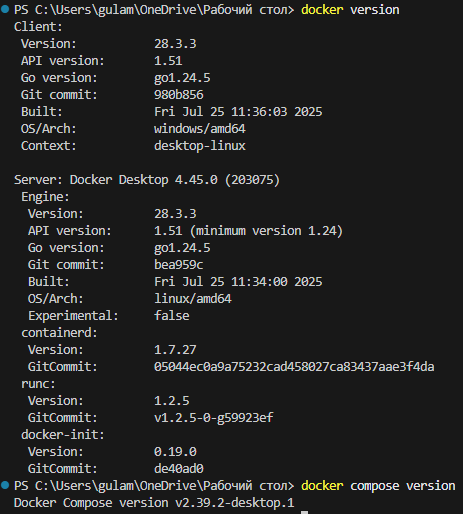


Рисунок 2 – Команды «docker version» и «docker compose version»

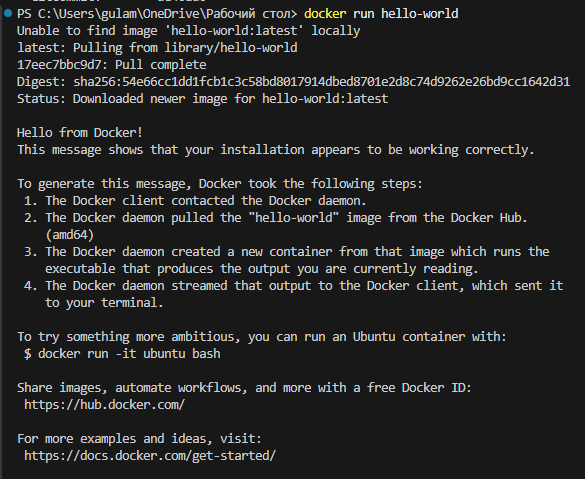


Рисунок 3 – Команда «docker hello-world»

**2) Базовые команды Docker. Работа с образами и контейнерами**

Воспользуемся базовыми командами для просмотра информации (Рисунок 4).

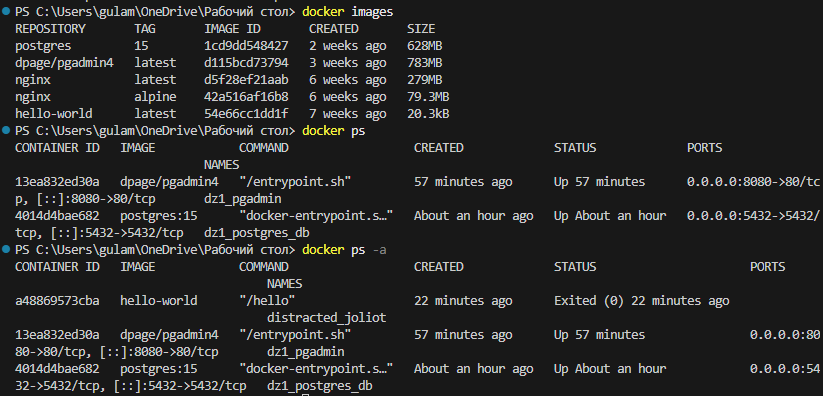


Рисунок 4 – Ввод команд «docker images», «docker ps» и «docker ps -a»

Создадим простейший контейнер и проверим его работу, набрав в браузере «localhost:8080» (Рисунки 5-6).

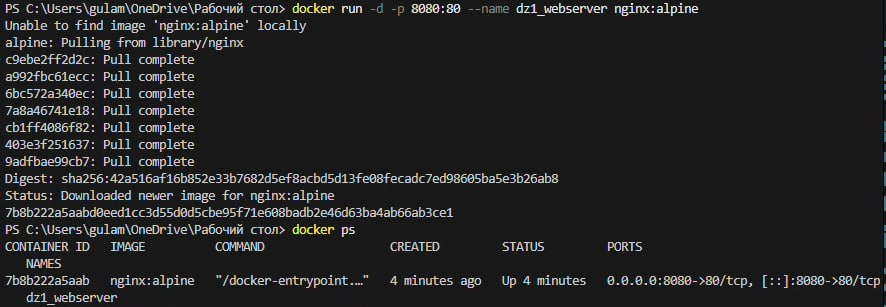


Рисунок 5 – Создание контейнера

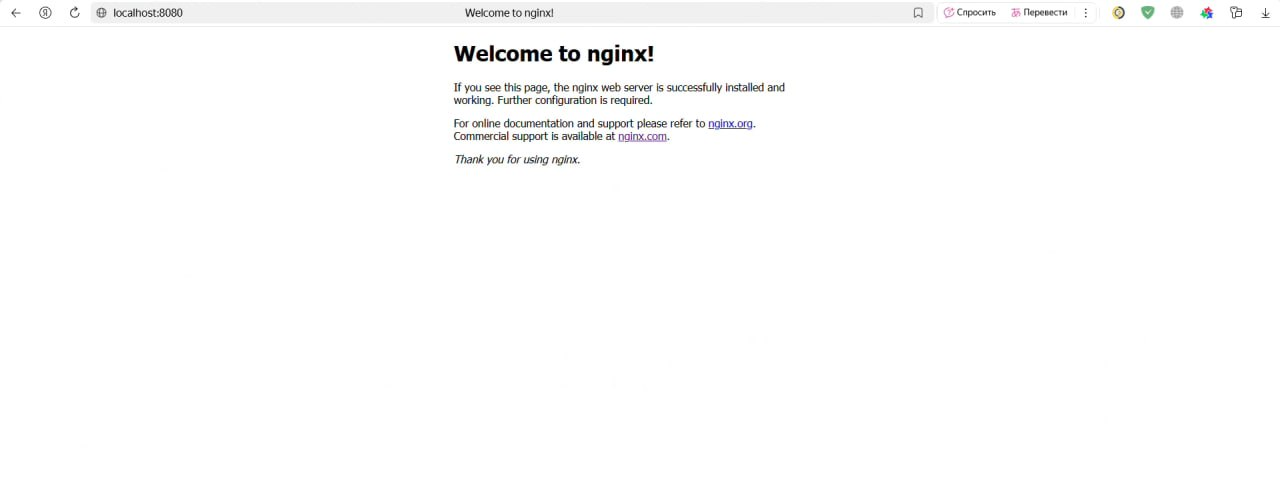


Рисунок 6 – Переход на «localhost:8080»

**3) Запуск PostgreSQL в контейнере**

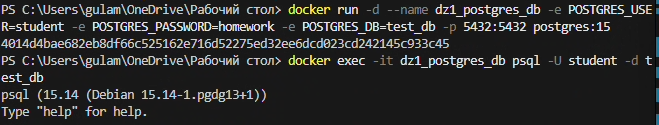
Запустим контейнер с PostgreSQL и подключимся к БД из контейнера (Рисунок 7). Создадим таблицу с помощью команды «CREATE TABLE users (id SERIAL PRIMARY KEY, name VARCHAR(50));», вставим данные с помощью «INSERT INTO users (name) VALUES (‘Alice’), (‘Bob’);» и проверим ее с помощью команды «SELECT \* FROM users;» (Рисунок 8).

Рисунок 7 – Запуск контейнера с PostgreSQL

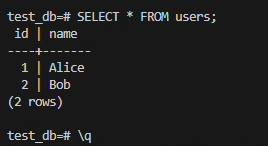


Рисунок 8 – Проверка созданной таблицы и выход из консоли

**4) Подключение к БД через pgAdmin из второго контейнера**

Создадим сеть Docker и проверим, что она появилась в списке сетей (Рисунок 9, случайно была создана сеть «ls»)

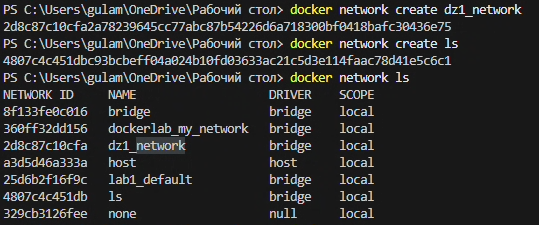


Рисунок 9 – Создание сети

Подключим контейнер с PostgreSQL к сети и запустим pgAdmin в той же сети (Рисунок 10).

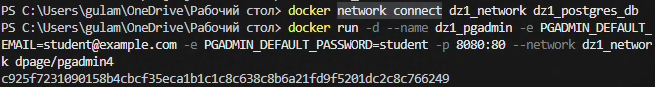


Рисунок 10 – Подключение к сети и запуск pgAdmin

Откроем «localhost:8080» и попадем в pgAdmin и подключим сервер с нашей созданной базой данных, далее проверим, что все создано (Рисунок 11).

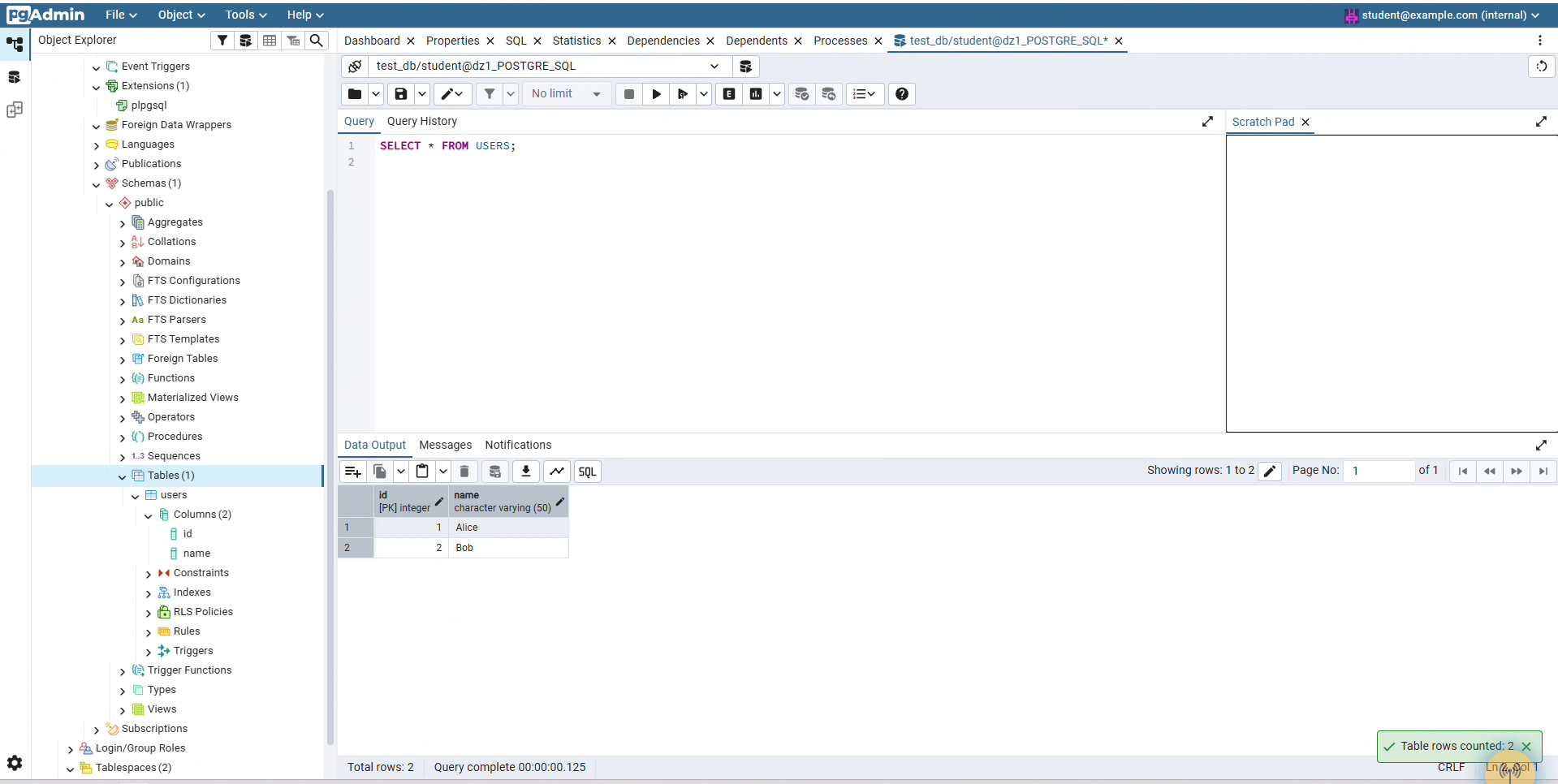


Рисунок 11 – Окно с pgAdmin

**5) Сохранение данных с помощью Томов (Volumes)**

Остановим текущий контейнер с БД, создадим том и запустим новый контейнер (Рисунок 12).

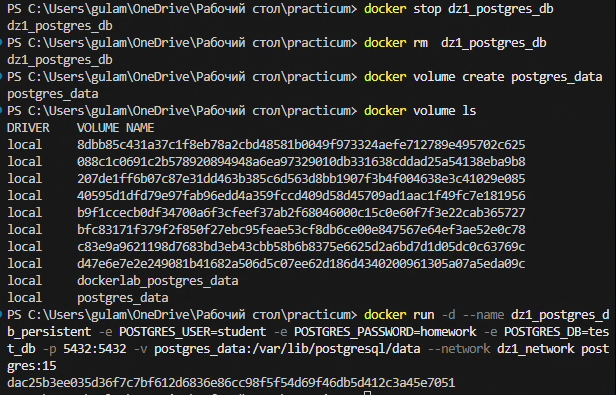


Рисунок 12 – Создание тома

Через pgAdmin подключимся к нашей БД, создадим таблицу и введем данные (Рисунок 13).

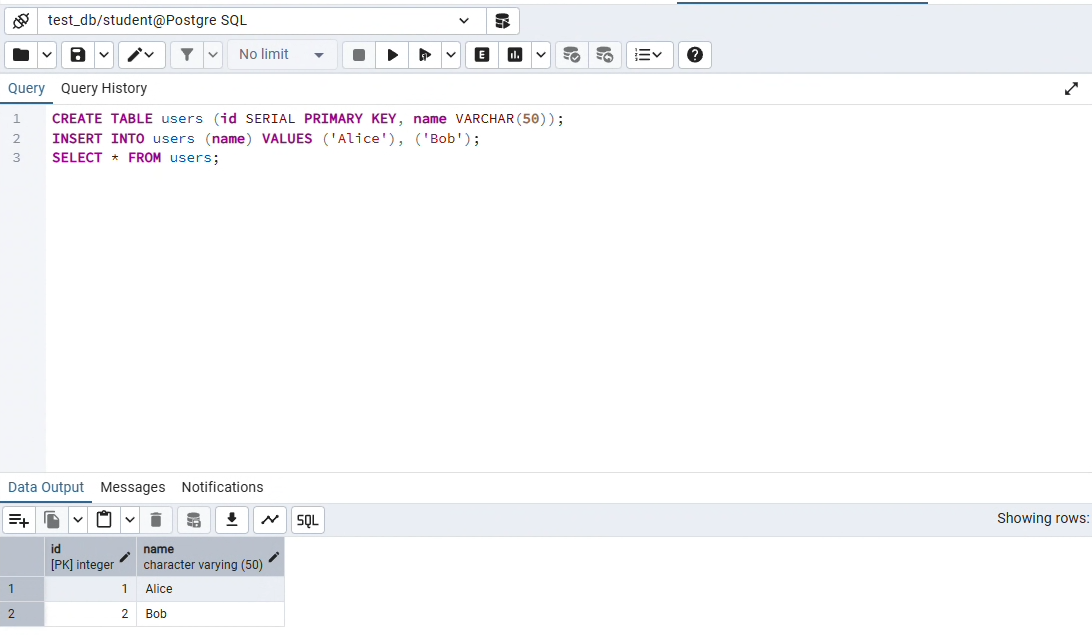


Рисунок 13 – Создание таблицы через pgAdmin

Далее отключим контейнеры, включим заново и проверим сохранность данных (Рисунки 14-15).

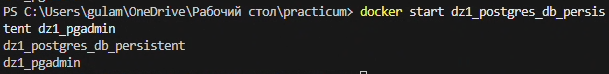


Рисунок 14 – Запуск контейнеров

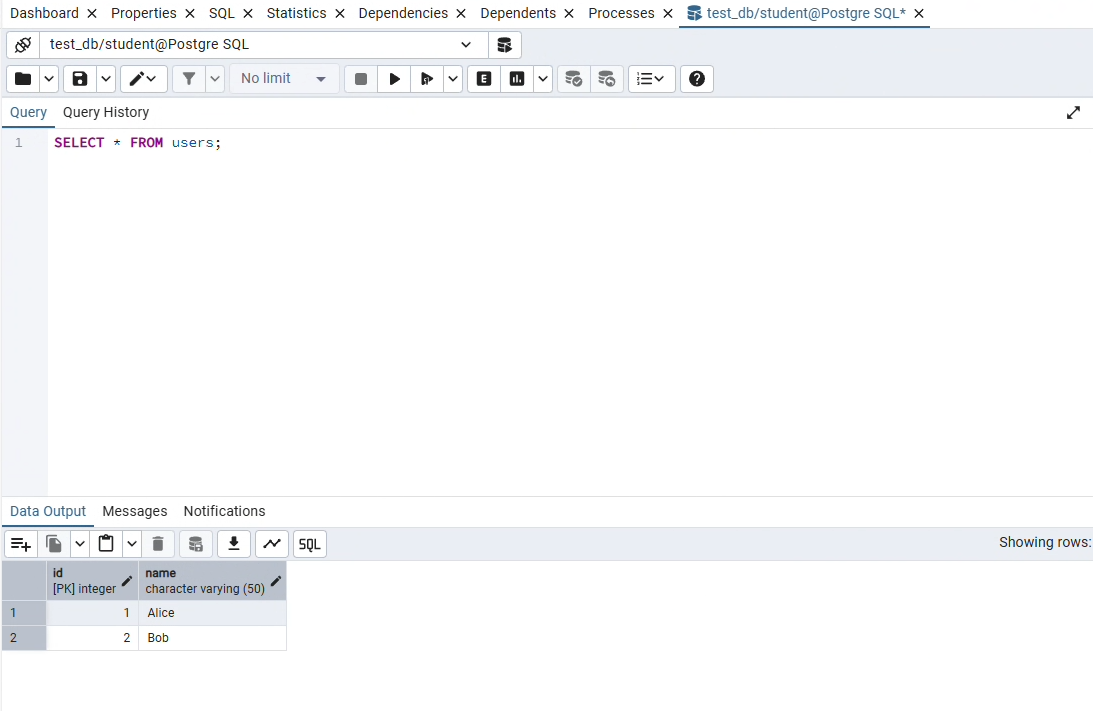


Рисунок 15 – Проверка данных в pgAdmin

Данные сохранились, всё работает.

**6) Перенос конфигурации контейнеров в docker-compose.yaml**

Создадим файл docker-compose.yaml (файл приложен в репозитории на github). «Поднимем» его командой «docker-compose up -d». Проверим работоспособность, перейдя на localhost:5050 (Рисунок 16).

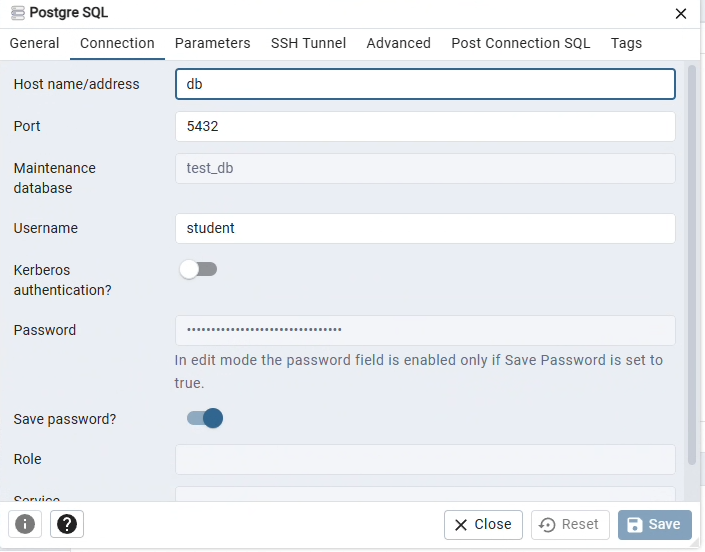


Рисунок 16 – Подключение БД к pgAdmin

Создадим там таблицу с помощью команды «CREATE TABLE users (id SERIAL PRIMARY KEY, name VARCHAR(50));», вставим данные с помощью «INSERT INTO users (name) VALUES (‘Alice’), (‘Bob’);» и проверим ее с помощью команды «SELECT \* FROM users;». Затем отключим с помощью команды в терминале «docker-compose down» и «поднимем» заново (Рисунок 17), данные сохранились.

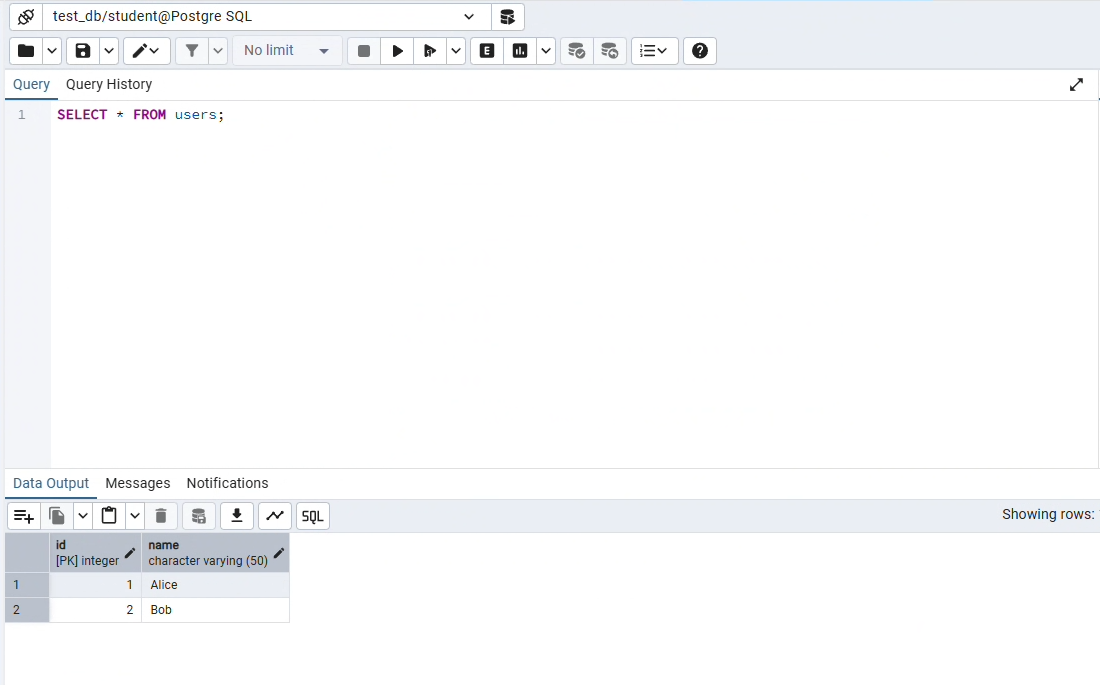


Рисунок 17 – Проверка данных после второго «поднятия»

**Вопросы:**

1. Что такое docker?

Docker – инструмент, позволяющий упаковывать приложения и всё, что им нужно для работы (код, библиотеки, настройки), в контейнеры. Например, применимо к нашей работе, мы можем запустить веб-сервер NGINX, СУБД PostgreSQL.

2. Для чего нужны тома и сети docker?

Том нужен для сохранения данных. Если подключить том, данные сохранятся даже после перезапуска или удаления контейнера. Сети нужны для обмена данными между контейнерами. По умолчанию контейнеры не видят друг друга, но если подключить их к одной созданной сети, они смогут общаться по имени

3. Как подключится к контейнеру и выполнить в нём команды?

Мы можем подключиться к контейнеру через терминал, используя команду «docker exec -it ***имя\_контейнера*** ***команда***». В нашей работе мы использовали docker exec -it dz1\_postgres psql -U student -d test\_db для подключения к базе данных.

4. Для чего нужен pgAdmin?

PgAdmin – веб-интерфейс для управления PostgreSQL, он позволяет подключаться к базе данных через браузер, просматривать таблицы, схемы, индексы, выполнять SQL-запросы и создавать/удалять пользователей, базы, таблицы – без командной строки.

**Вывод:**

В ходе выполнения лабораторной работы были освоены основные концепции и инструменты Docker, необходимые для развертывания и управления приложениями в контейнерах.

Удалось успешно запустить контейнеры с веб-сервером NGINX и СУБД PostgreSQL, настроить тома (volumes) для сохранения данных базы данных даже после удаления контейнера, создать пользовательскую Docker-сеть, обеспечивающую взаимодействие между контейнерами по имени, развернуть веб-интерфейс pgAdmin и подключить его к PostgreSQL-контейнеру для удобного управления базой данных, автоматизировать запуск всей инфраструктуры с помощью файла docker-compose.yml.

В процессе работы были преодолены ошибки, связанные с отсутствием опыта работы с Docker, например, конфликты портов, некорректный синтаксис yaml, проблемы с аутентификацией и кавычками в SQL. Это позволило глубже понять особенности работы с Docker и важность внимательности при настройке конфигураций.

В результате создана полностью изолированная, воспроизводимая и переносимая среда разработки, состоящая из базы данных и инструмента для её администрирования, что подтверждает практическую применимость Docker в реальных проектах.

**Ссылка на репозиторий:**

**https://github.com/bepis-art/application-development-2025-urfu.git**