Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное  
образовательное учреждение высшего образования

«Комсомольский-на-Амуре государственный университет»

Факультет компьютерных технологий

Кафедра «Проектирование, управление и разработка информационных систем»

Направление 09.03.01 – «Информатика и вычислительная техника»

ОТЧЁТ

по учебной (ознакомительной) практики

Разработка Telegram-бота для тестирования знаний по SQL и работе с реляционными базами данных на примере PostgreSQL

Студент группы 3ИТб-1 Т. Е. Акименко

Преподаватель А.Н. Петрова

2025

**Лист задания**

1. Изучить основные принципы построения тестовых систем для проверки знаний по работе с реляционными базами данных (СУБД PostgreSQL).
2. Разработать Telegram-бота, реализующего тестирование пользователей по SQL-запросам и работе с базами данных PostgreSQL.
3. Реализовать в боте два режима работы:
   * Обучение – после каждого ответа бот сообщает, правильный ли он, и, если ответ неверный, выводит пояснение (текст, изображение или ссылку).
   * Контроль – обратная связь и итоговая оценка предоставляются только после завершения теста.
4. Организовать выбор раздела теста (если разделов несколько) и формирование теста из вопросов с вариантами ответов (3–4 на каждый вопрос).
5. Для каждого вопроса предусмотреть правильный вариант и пояснение к нему.
6. Все вопросы, варианты ответов, правильные ответы и пояснения реализовать во внешнем файле (например, .json) по самостоятельно разработанной структуре.
7. Реализовать сохранение результатов тестирования в заранее согласованном формате (.json, .csv и т.п.).
8. Оформить исходный код Telegram-бота и файл с вопросами и ответами, опубликовать работу на GitHub.
9. Сформировать отчет согласно РД013-2016.

**Содержание**

[Введение 3](#_Toc201668574)

[Ход работы 5](#_Toc201668575)

[1 Анализ задания и проектирование структуры теста 5](#_Toc201668576)

[2 Формирование базы вопросов 5](#_Toc201668577)

[3 Регистрация Telegram-бота через BotFather 7](#_Toc201668578)

[4 Реализация логики тестирования в Python. 8](#_Toc201668579)

[5 Тестирование и отладка. 12](#_Toc201668580)

[6 Публикация на Github 15](#_Toc201668581)

[Заключение 18](#_Toc201668582)

[Список использованных источников 19](#_Toc201668583)

Введение

В современном мире информационные технологии играют ключевую роль во всех сферах деятельности, а эффективное управление данными становится одним из важнейших факторов успешной работы организаций. Реляционные базы данных и язык структурированных запросов SQL являются стандартом для хранения, обработки и анализа больших объемов информации. Владение навыками работы с реляционными базами данных, а также умение формировать корректные SQL-запросы, востребовано среди специалистов в области информационных технологий, программирования и анализа данных.

Одним из эффективных способов проверки и совершенствования знаний по работе с базами данных является автоматизированное тестирование. Использование современных средств коммуникации, таких как мессенджеры, позволяет сделать процесс обучения и самопроверки более доступным и удобным для пользователей. Telegram-боты, благодаря своей популярности и простоте интеграции, широко применяются для автоматизации различных задач, в том числе образовательных.

Целью данной работы является разработка Telegram-бота для тестирования знаний по SQL и основам работы с реляционными базами данных на примере СУБД PostgreSQL. В ходе выполнения работы реализуются различные режимы тестирования, формируется база вопросов с вариантами ответов и пояснениями, а также обеспечивается сохранение и анализ результатов тестирования. Такой подход способствует не только объективной оценке уровня знаний, но и формированию практических навыков работы с современными системами управления базами данных.

Ход работы

1. Анализ задания и проектирование структуры теста

В ходе анализа задания было определено, что целью работы является создание системы тестирования знаний по SQL на примере таблицы 1 «Музыканты» (Muz). Данная таблица содержит четыре поля: идентификатор музыканта, имя, дату рождения и страну рождения, что позволяет охватить основные типы данных и типовые SQL-операции.

Таблица 1 – Музыканты

| Имя поля | Тип данных | Описание |
| --- | --- | --- |
| Num\_muz | int | Номер музыканта |
| Name\_muz | nvarchar(50) | Имя музыканта |
| Birthday | datetime2(7) | Дата рождения |
| Country | nvarchar(50) | Страна рождения |

На основе данной структуры были разработаны вопросы, проверяющие умение формировать запросы SELECT, использовать фильтрацию, сортировку, группировку, функции для работы со строками и датами. Такой подход обеспечивает комплексную проверку знаний по работе с реляционными базами данных в СУБД PostgreSQL

1. Формирование базы вопросов

База вопросов для тестирования была сформирована на основе структуры таблицы Музыканты (Muz) и примеров SQL-запросов, приведённых в методических материалах [1, с 14-28)]. Основная задача — охватить ключевые аспекты языка SQL, используемые при работе с реляционными базами данных.

Темы, по которым составлены вопросы:

* + Выборка всех столбцов и записей из таблицы (SELECT \*)
  + Переименование столбцов и использование псевдонимов (AS)
  + Отбор уникальных значений (DISTINCT)
  + Ограничение количества выводимых строк (LIMIT)
  + Применение строковых и арифметических функций (например, UPPER, CONCAT, арифметические операции)
  + Сортировка результатов (ORDER BY)
  + Группировка данных и агрегатные функции (GROUP BY, COUNT)
  + Фильтрация по условиям (WHERE)
  + Работа с датами (EXTRACT, AGE, NOW)

Структура файла с вопросами:

Вопросы хранятся во внешнем файле формата JSON. Каждый вопрос включает:

* + Текст вопроса
  + 3–4 варианта ответа
  + Индекс правильного варианта
  + Пояснение к правильному ответу

Пример структуры одного вопроса показан листинге 1.

Листинг 1: Код файла questions.json

|  |
| --- |
| {  "id": 1,  "question": "Как выбрать все столбцы и записи из таблицы музыкантов?",  "options": [  "SELECT \* FROM muz;",  "SELECT ALL FROM muz;",  "SELECT columns FROM muz;"  ],  "correct\_option": 0,  "explanation": "В PostgreSQL для вывода всех столбцов и записей используется SELECT \* FROM table\_name;."  } |

Вопросов:

1. Как выбрать все столбцы и записи из таблицы музыкантов?
2. Как в PostgreSQL задать псевдоним столбцу Name\_muz в запросе?
3. Как вывести уникальные страны из таблицы музыкантов?
4. Как выбрать первых двух музыкантов из таблицы?
5. Как вывести имена музыкантов в верхнем регистре?
6. Как отсортировать музыкантов по дате рождения по убыванию?
7. Как сгруппировать музыкантов по стране и посчитать их количество?
8. Как выбрать музыкантов, родившихся после 1900 года?
9. Как получить год рождения музыканта?
10. Как вычислить возраст музыканта в годах на текущий момент?
11. Регистрация Telegram-бота через BotFather

Для взаимодействия с пользователями в Telegram и автоматизации процесса тестирования был создан специальный Telegram-бот. Регистрация и получение токена доступа осуществляются через официальный бот Telegram –

BotFather.

В Telegram был найден и запущен официальный бот [@BotFather](https://t.me/botfather), предназначенный для управления и создания новых ботов. С помощью команды /newbot был инициирован процесс создания бота. В ходе диалога с BotFather были указаны:

* + Имя бота (например, SQLTestBot).
  + Уникальное имя пользователя (username), оканчивающееся на bot (например, sql\_test\_bot).

После успешного создания BotFather выдал уникальный токен (строку символов), необходимый для интеграции Telegram API с программным кодом. Полученный токен был вставлен в файл main.py в соответствующую строку для авторизации бота при запуске программы (рисунок 1).

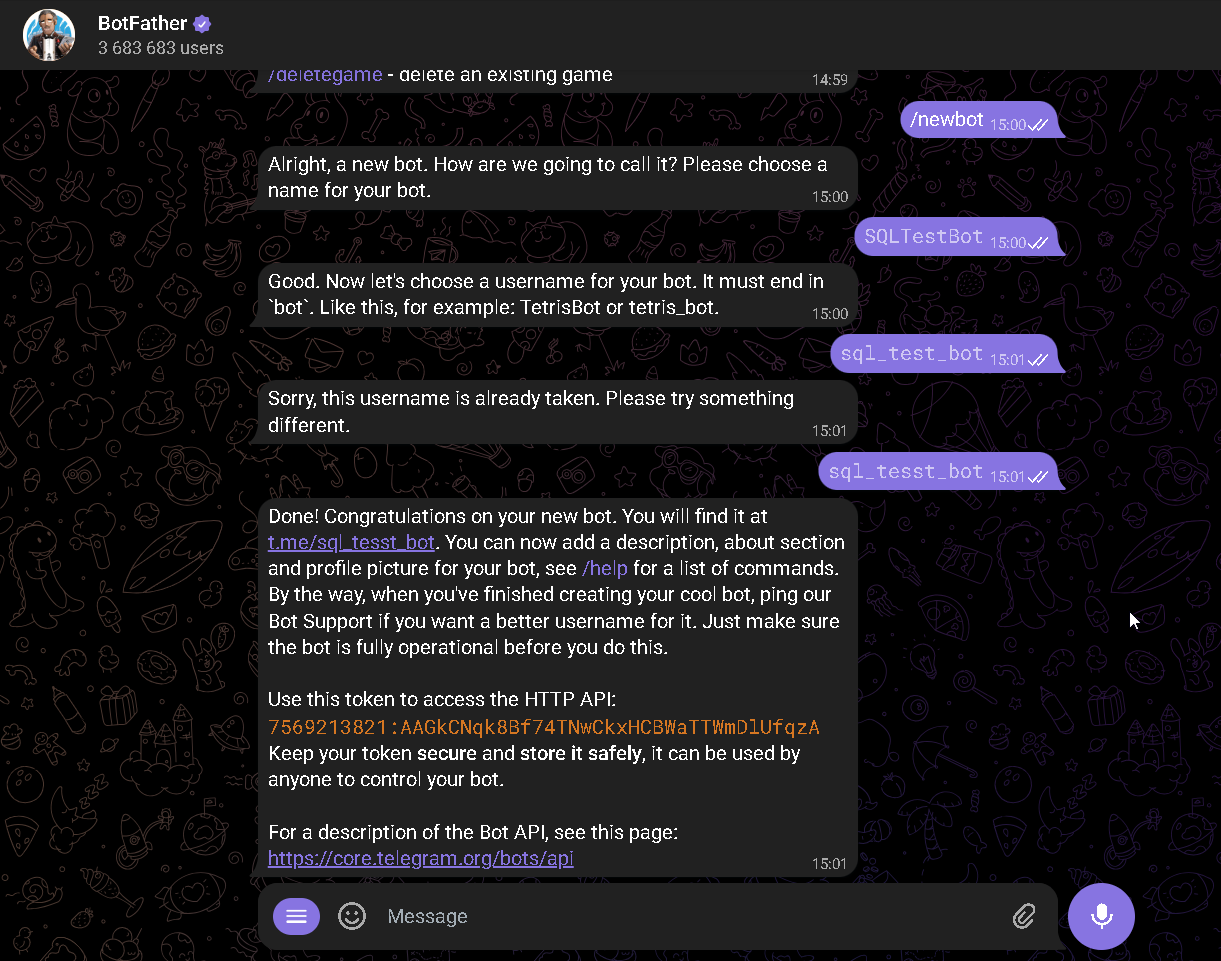


Рисунок 1 – Создание Telegram-бота

1. Реализация логики тестирования в Python.

Для реализации Telegram-бота использовался язык программирования Python и библиотека python-telegram-bot, обеспечивающая удобный интерфейс для взаимодействия с Telegram API.

Основные этапы реализации логики тестирования:

1. Загрузка базы вопросов

Вопросы и варианты ответов хранятся во внешнем файле questions.json в формате JSON. При запуске бота этот файл загружается в память для последующего использования.

1. Обработка команд и сообщений пользователей

Бот реагирует на команду /start, предлагая пользователю выбрать режим работы: «Обучение» или «Контроль». Выбор режима сохраняется в сессии пользователя.

1. Показ вопросов и вариантов ответов

В зависимости от текущего номера вопроса бот отправляет пользователю текст вопроса и варианты ответов в виде кнопок клавиатуры. Пользователь выбирает один из вариантов.

1. Обработка ответов и подсчёт результатов

После выбора ответа бот проверяет его правильность, сохраняет результат и, в режиме обучения, сразу выводит пояснение к ответу. В режиме контроля пояснения выводятся после завершения теста.

1. Завершение теста и вывод итогов

После ответа на все вопросы бот выводит пользователю количество правильных ответов и, при необходимости, детальные пояснения по ошибочным вопросам.

1. Сохранение результатов

Результаты тестирования каждого пользователя сохраняются в файл results.json для последующего анализа.

Текст программы показан в Листинг 2.

Листинг 1: Код файла main.py

|  |
| --- |
| import json from telegram import Update, ReplyKeyboardMarkup from telegram.ext import Application, CommandHandler, MessageHandler, filters, ContextTypes  # Загрузка вопросов with open("questions.json", "r", encoding="utf-8") as f:  questions = json.load(f)  # Сессии пользователей: user\_id -> состояние user\_sessions = {}  # Загрузка и сохранение результатов def load\_results():  try:  with open("results.json", "r", encoding="utf-8") as f:  return json.load(f)  except:  return []  def save\_results(results):  with open("results.json", "w", encoding="utf-8") as f:  json.dump(results, f, ensure\_ascii=False, indent=2)  async def start(update: Update, context: ContextTypes.DEFAULT\_TYPE):  user\_sessions[update.effective\_user.id] = {}  await update.message.reply\_text(  "Привет! Выберите режим теста:",  reply\_markup=ReplyKeyboardMarkup([["Обучение", "Контроль"]], one\_time\_keyboard=True)  )  async def mode\_choice(update: Update, context: ContextTypes.DEFAULT\_TYPE):  mode = update.message.text.strip().lower()  if mode not in ["обучение", "контроль"]:  await update.message.reply\_text("Пожалуйста, выберите режим: Обучение или Контроль.")  return  user\_id = update.effective\_user.id  user\_sessions[user\_id] = {  "mode": mode,  "current": 0,  "score": 0,  "answers": []  }  await send\_question(update, context, user\_id)  async def send\_question(update, context, user\_id):  session = user\_sessions[user\_id]  idx = session["current"]  if idx >= len(questions):  await finish\_test(update, context, user\_id)  return  q = questions[idx]  reply\_markup = ReplyKeyboardMarkup([[opt] for opt in q["options"]], one\_time\_keyboard=True)  await update.message.reply\_text(f"Вопрос {idx + 1}/{len(questions)}:\n{q['question']}", reply\_markup=reply\_markup)  async def handle\_answer(update: Update, context: ContextTypes.DEFAULT\_TYPE):  user\_id = update.effective\_user.id  if user\_id not in user\_sessions or "mode" not in user\_sessions[user\_id]:  await update.message.reply\_text("Сначала выберите режим теста: /start")  return  session = user\_sessions[user\_id]  idx = session["current"]  q = questions[idx]  user\_answer = update.message.text  try:  user\_choice = q["options"].index(user\_answer)  except ValueError:  await update.message.reply\_text("Пожалуйста, выберите один из предложенных вариантов.")  return   is\_correct = (user\_choice == q["correct\_option"])  session["answers"].append({"question\_id": q["id"], "answer": user\_choice, "correct": is\_correct})  if is\_correct:  session["score"] += 1   # В режиме обучения сразу выводим пояснение  if session["mode"] == "обучение":  explanation = q["explanation"] if not is\_correct else "Верно!"  await update.message.reply\_text(explanation)   session["current"] += 1  if session["current"] < len(questions):  await send\_question(update, context, user\_id)  else:  await finish\_test(update, context, user\_id)  async def finish\_test(update, context, user\_id):  session = user\_sessions[user\_id]  total = len(questions)  score = session["score"]  result\_text = f"Тест завершён!\nВаш результат: {score} из {total}."  await update.message.reply\_text(result\_text)  # В режиме контроль выводим пояснения только сейчас  if session["mode"] == "контроль":  for i, ans in enumerate(session["answers"]):  q = questions[i]  if not ans["correct"]:  await update.message.reply\_text(  f"Вопрос: {q['question']}\nВаш ответ: {q['options'][ans['answer']]}\nПравильный ответ: {q['options'][q['correct\_option']]}\nПояснение: {q['explanation']}"  )  # Сохраняем результат  results = load\_results()  results.append({  "user\_id": user\_id,  "username": update.effective\_user.username,  "score": score,  "total": total,  "mode": session["mode"],  "answers": session["answers"]  })  save\_results(results)  # Сброс сессии  del user\_sessions[user\_id]  def main():  app = Application.builder().token("7569213821:AAGkCNqk8Bf74TNwCkxHCBWaTTWmDlUfqzA").build()  app.add\_handler(CommandHandler("start", start))  app.add\_handler(MessageHandler(filters.Regex("^(Обучение|Контроль)$"), mode\_choice))  app.add\_handler(MessageHandler(filters.TEXT & ~filters.COMMAND, handle\_answer))  app.run\_polling()  if \_\_name\_\_ == "\_\_main\_\_":  main() |

1. Тестирование и отладка.

После реализации логики Telegram-бота был проведён комплексный этап тестирования и отладки для обеспечения корректной работы всех функций (рисунки 2-6).

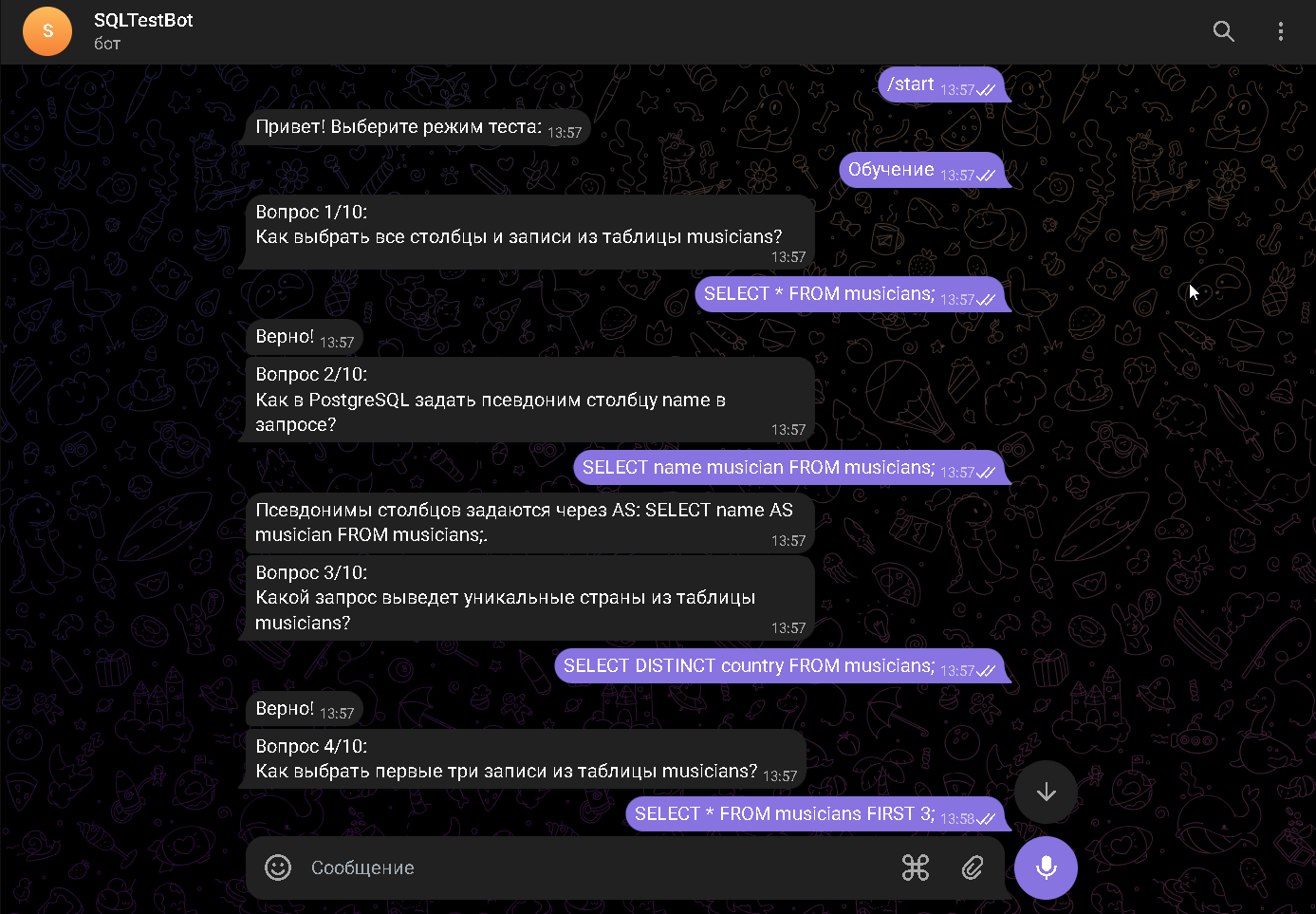


Рисунок 2 – Режим «Обучение»

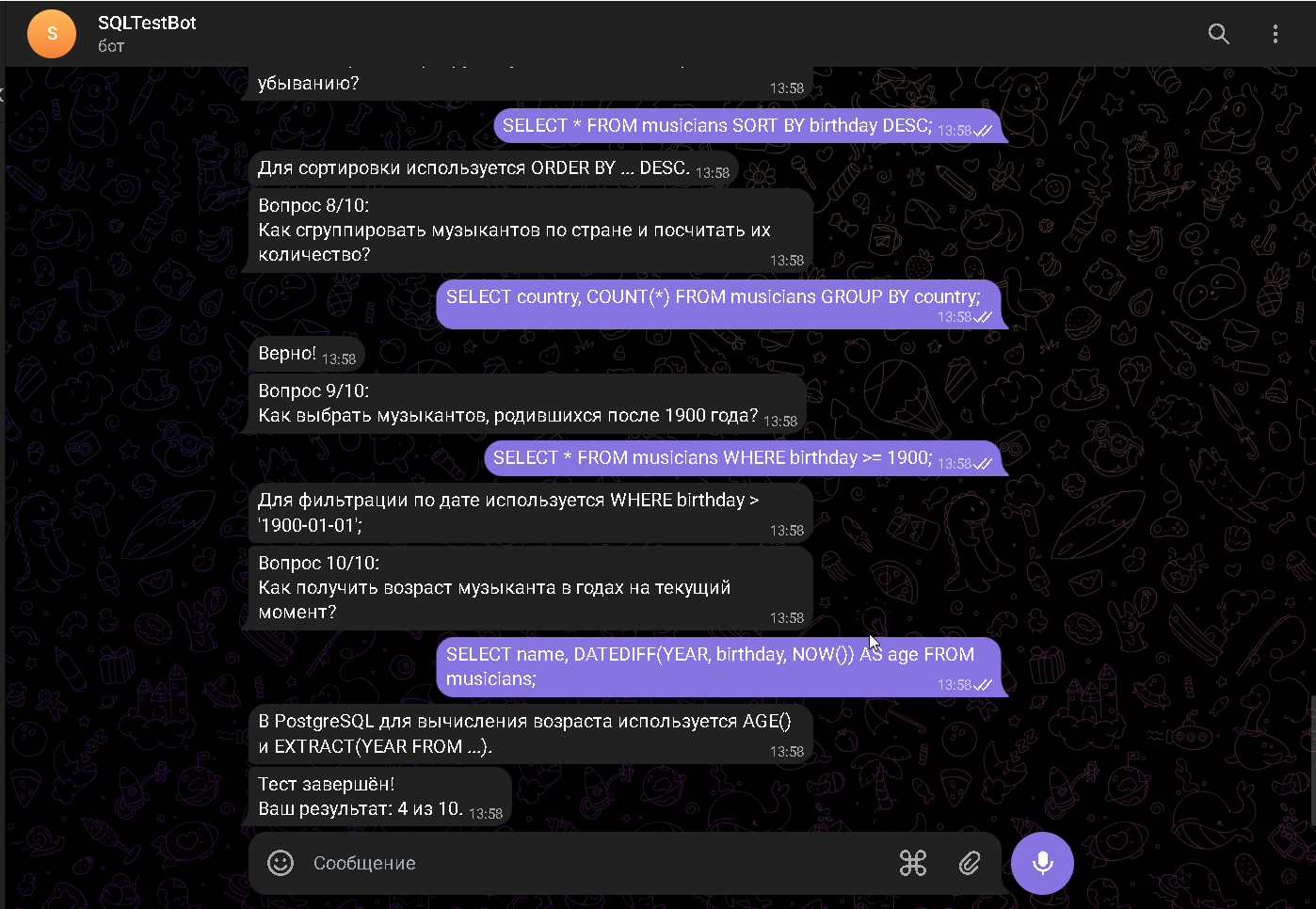


Рисунок 3 – Режим «Обучение»

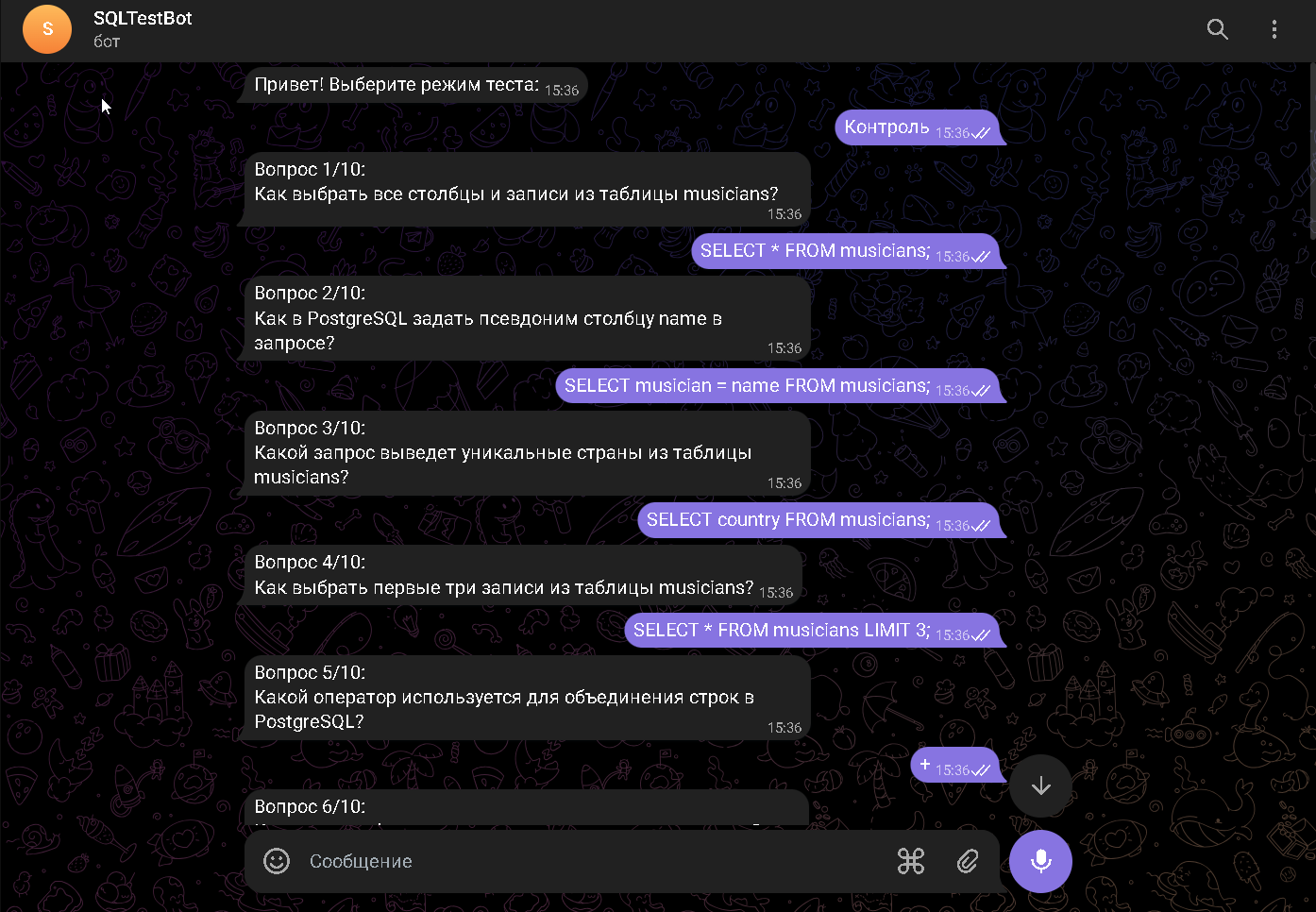


Рисунок 4 – Режим «Контроль»

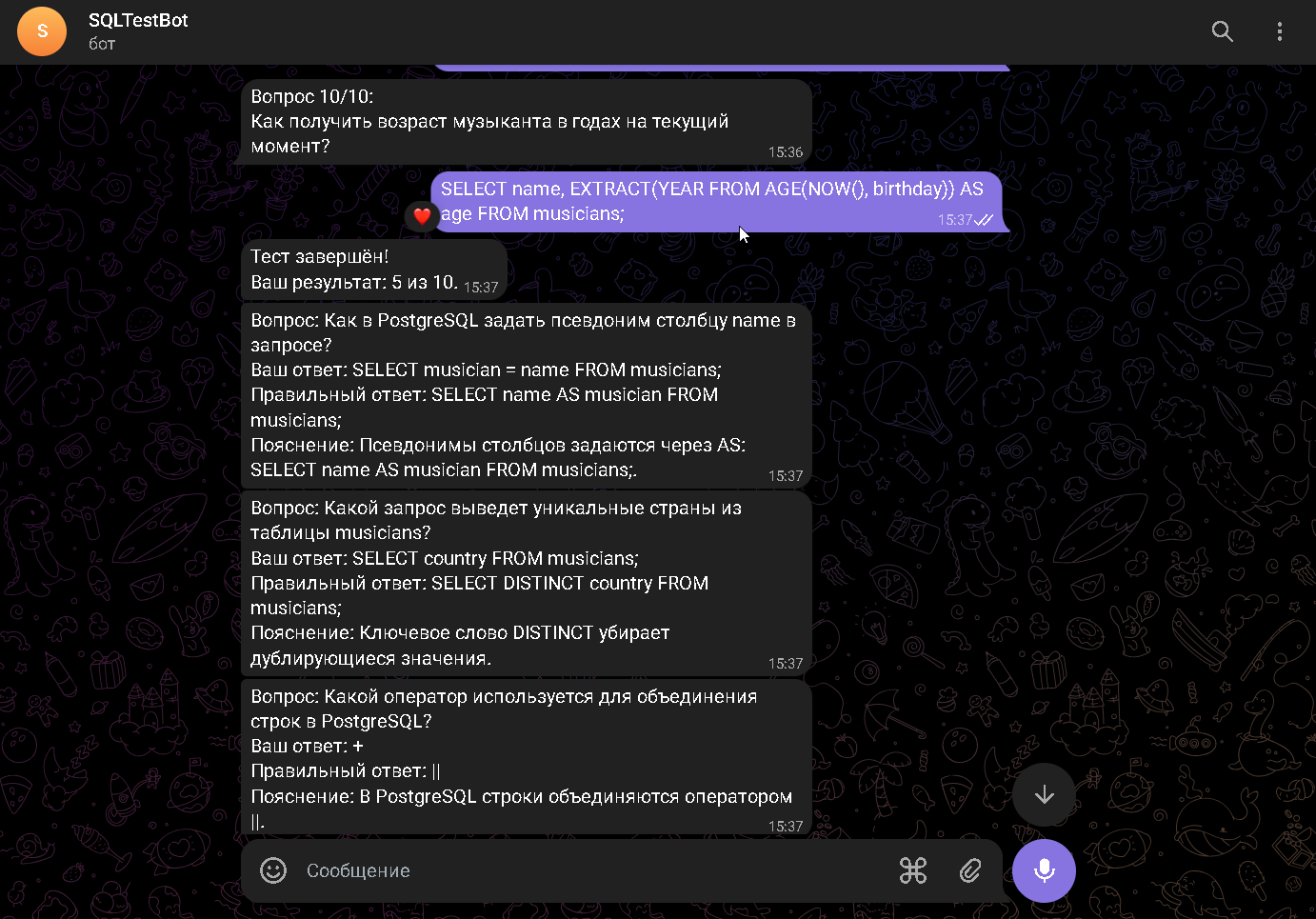


Рисунок 5 – Режим «Контроль»

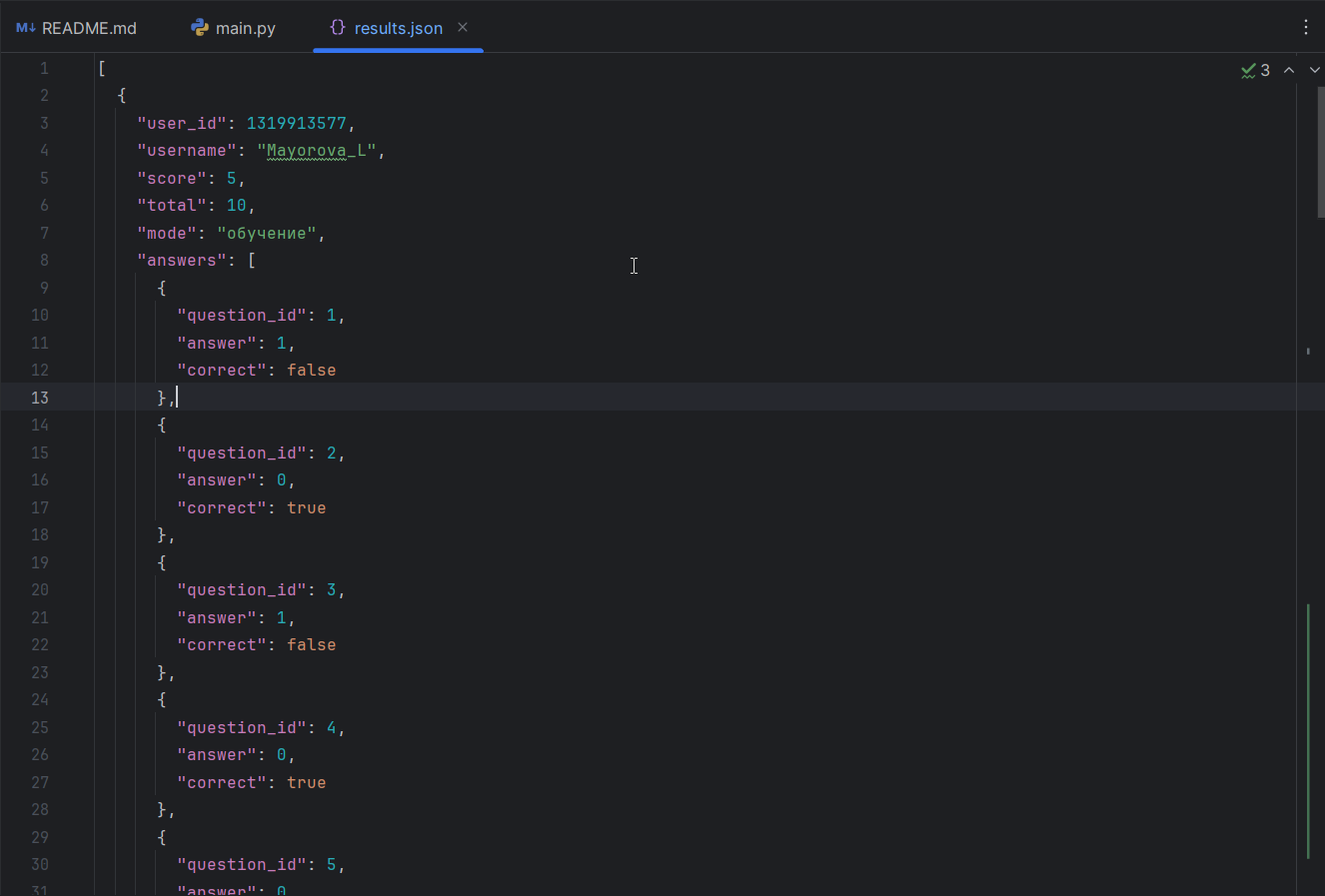


Рисунок 6 – Результаты тестирования

1. Публикация на Github

Для обеспечения открытого доступа к исходному коду, базе вопросов и отчёту проект был опубликован на платформе GitHub. Это позволяет преподавателю и другим заинтересованным лицам ознакомиться с реализацией работы, а также упростить проверку и возможное повторное использование материалов (рисунки 7-12).

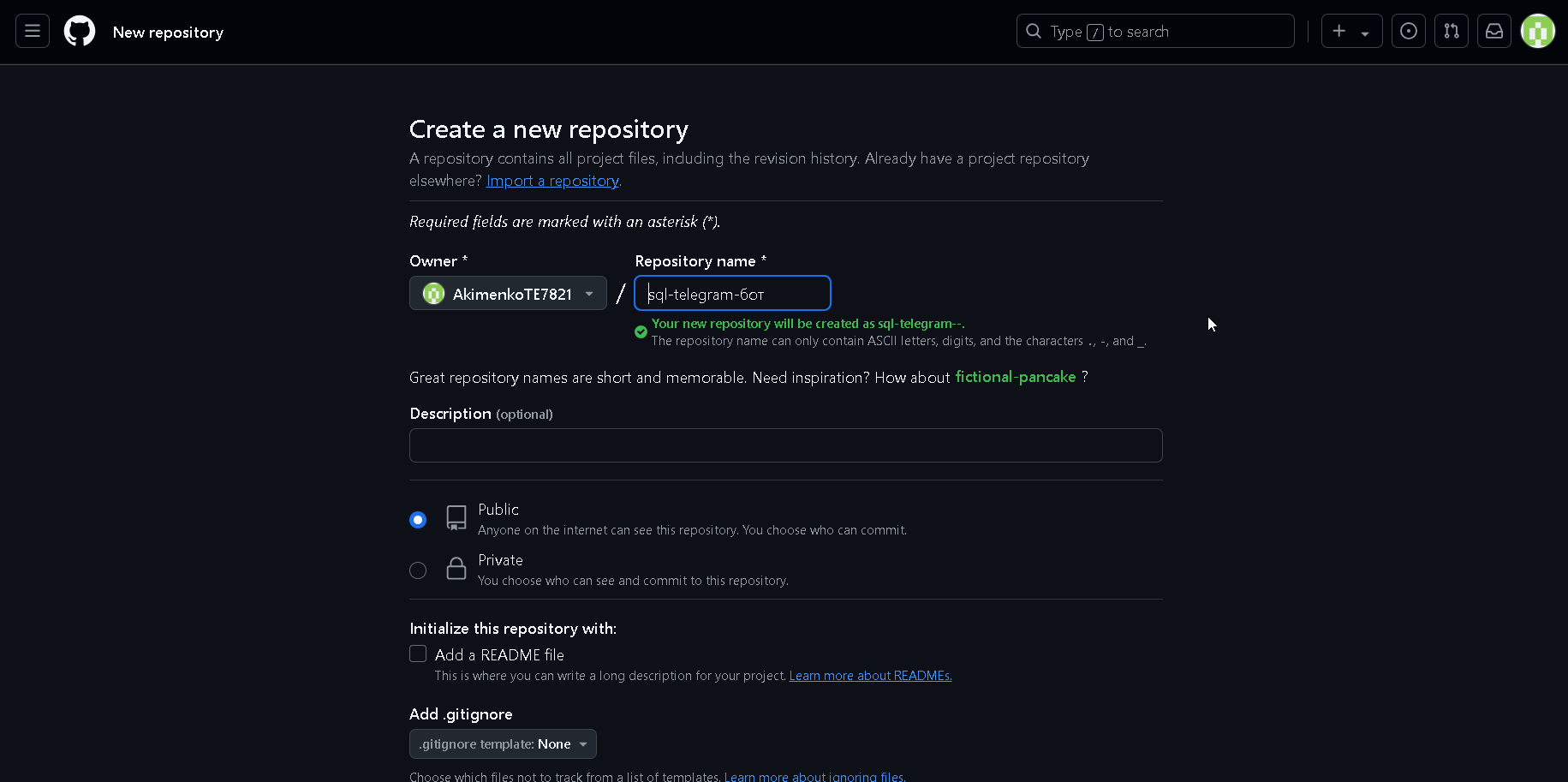


Рисунок 7 – Создание репозитория

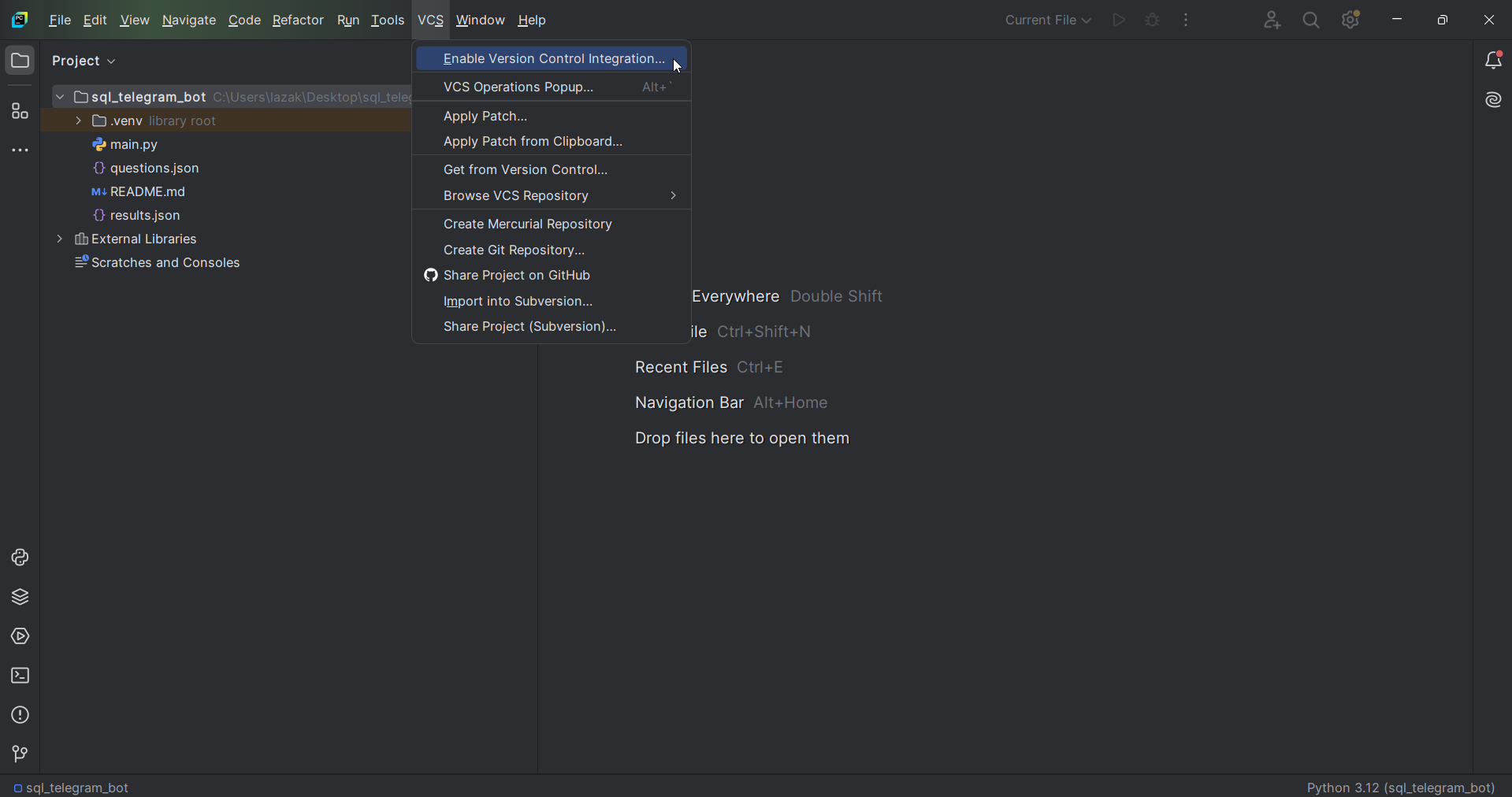


Рисунок 8 – Инициализация git-репозитория

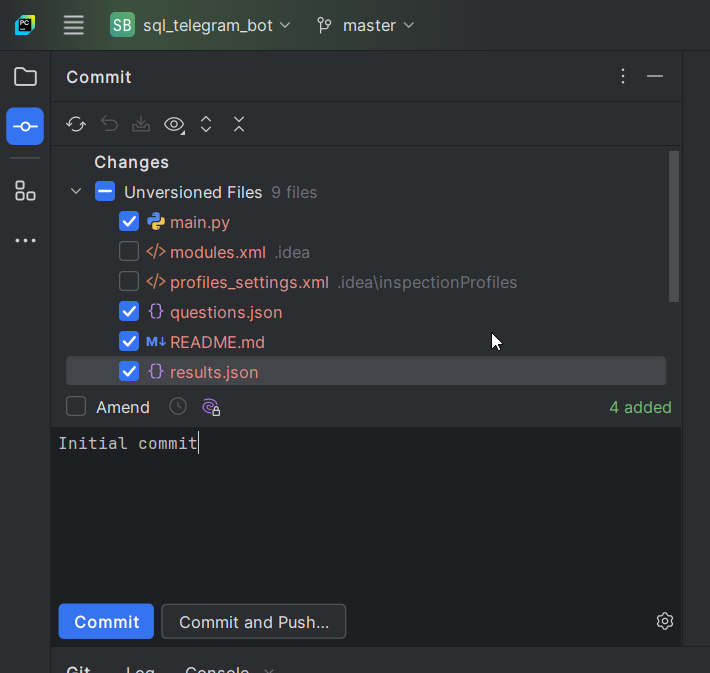


Рисунок 9 – Первый коммит

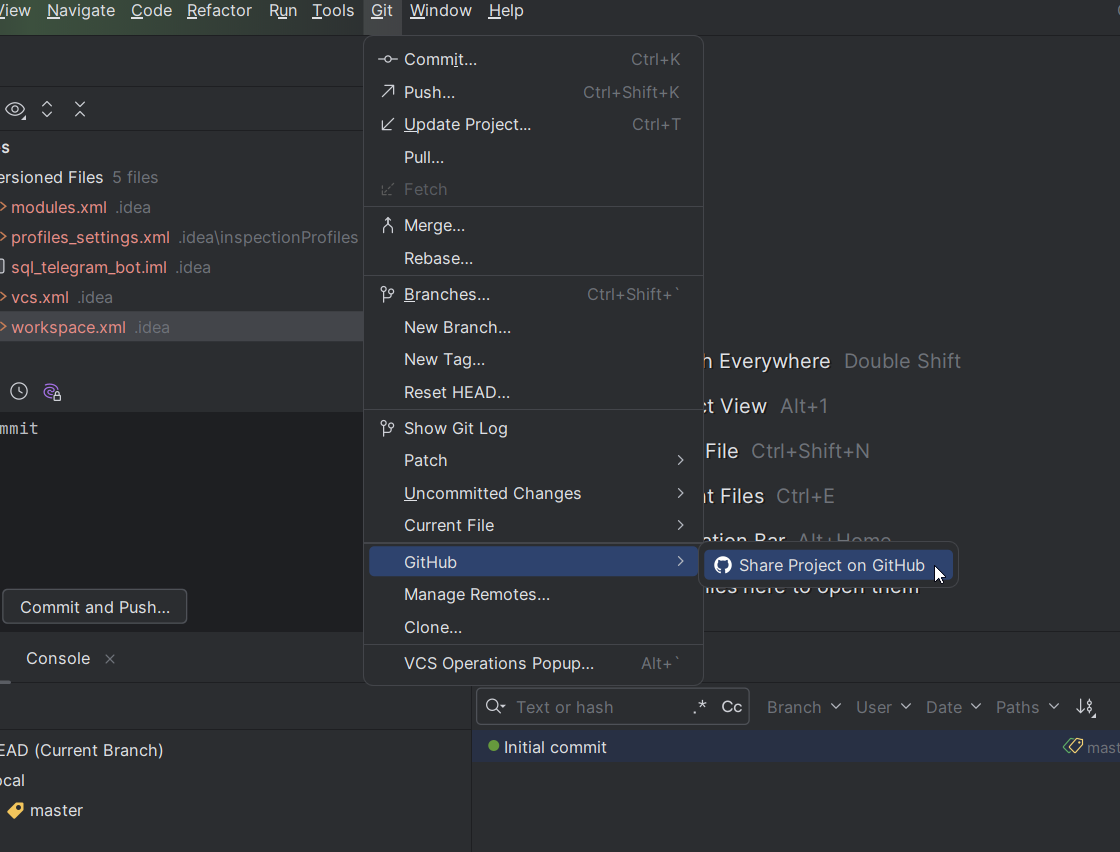


Рисунок 10 – Публикация на GitHub

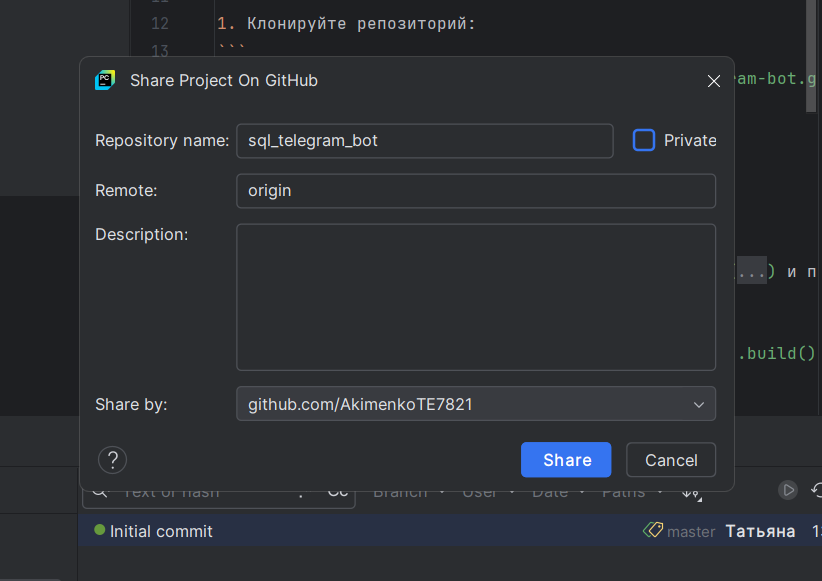


Рисунок 11 – Публикация на GitHub

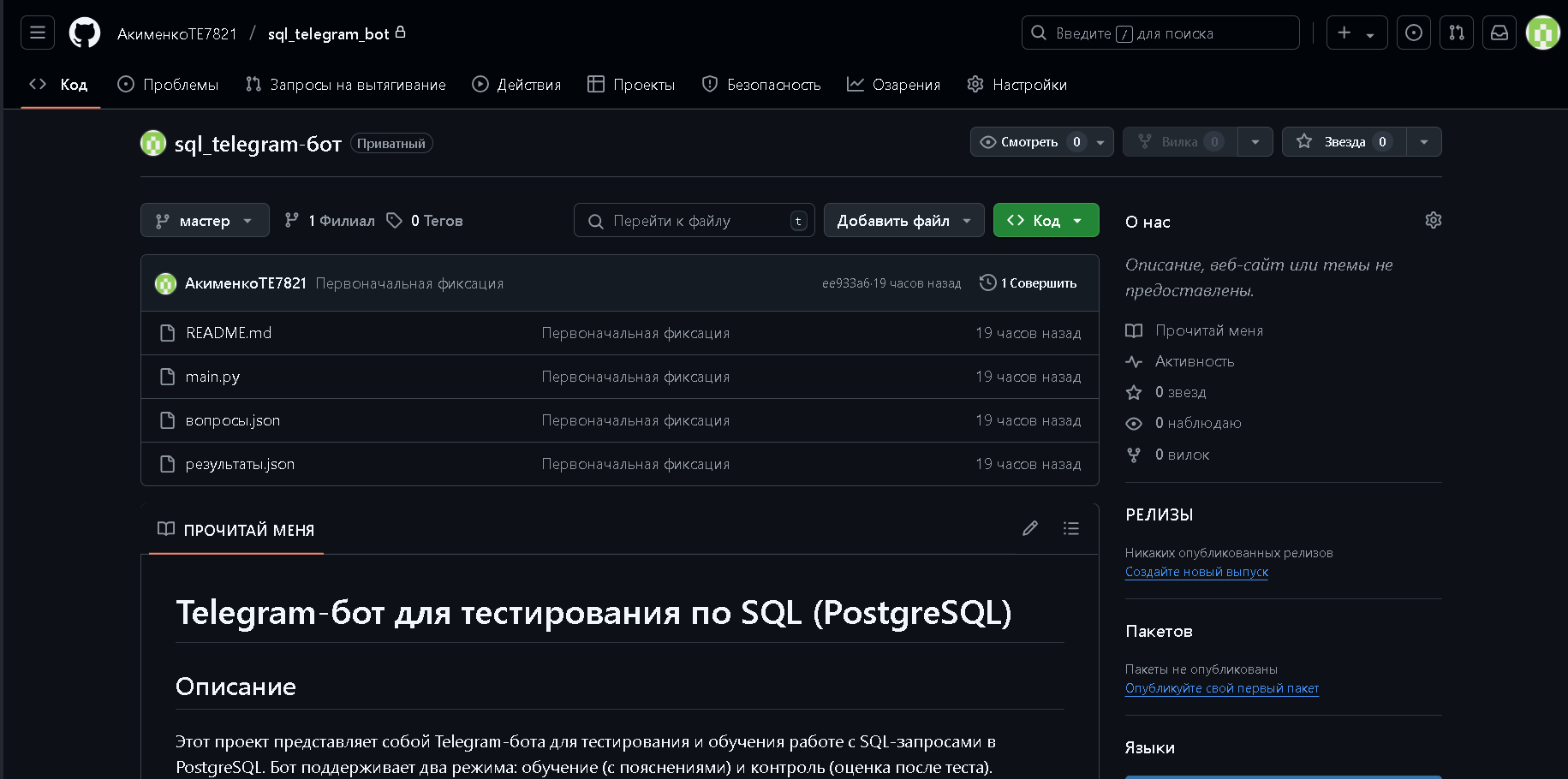


Рисунок 12 – Опубликованный проект

В результате все материалы лабораторной работы – исходный код, база вопросов, результаты тестирования, инструкция и отчёт – были опубликованы на GitHub и доступны для проверки по ссылке:

<https://github.com/AkimenkoTE7821/sql_telegram_bot>

Заключение

В ходе выполнения лабораторной работы была реализована система автоматизированного тестирования знаний по SQL с использованием Telegram-бота. В качестве предметной области была выбрана таблица «Музыканты», что позволило охватить широкий спектр SQL-операций: выборку данных, фильтрацию, сортировку, группировку, работу с функциями для строк и дат.

В процессе работы были выполнены все этапы проектирования: проведён анализ задания, спроектирована структура теста, сформирована база вопросов, реализована логика тестирования на языке Python с использованием библиотеки python-telegram-bot. Особое внимание уделялось удобству пользователя: бот поддерживает обучающий и контрольный режимы, выводит пояснения к ответам и сохраняет результаты прохождения теста.

Проведено тестирование и отладка, что позволило выявить и устранить ошибки, а также убедиться в корректной работе всех функций бота. Проект был опубликован на GitHub, что обеспечивает его доступность для проверки и дальнейшего использования.

В результате работы получен удобный инструмент для самостоятельной подготовки и контроля знаний по SQL, который может быть использован как в учебных целях, так и для самопроверки.

Список использованных источников

1. Петрова, А. Н. Реализация баз данных : учебное пособие / А. Н. Петрова, В. Е. Степаненко. — Комсомольск-на-Амуре : КНАГУ, 2020. — 144 с. — Текст : электронный. — URL: <https://e.lanbook.com/book/151716> (дата обращения: 24.06.2025). — Режим доступа: для авториз. пользователей.
2. Официальная документация PostgreSQL. — Текст : электронный. — URL: <https://www.postgresql.org/docs/> (дата обращения: 24.06.2025). — Режим доступа: свободный.
3. Официальная документация python-telegram-bot. — Текст : электронный. — URL: <https://docs.python-telegram-bot.org/> (дата обращения: 24.06.2025). — Режим доступа: свободный.
4. Telegram Bot API. — Текст : электронный. — URL: <https://core.telegram.org/bots/api> (дата обращения: 24.06.2025). — Режим доступа: свободный.