МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра математического обеспечения ЭВМ

**Разработка приложения для управления базой данных в формате PostgreSQL с помощью чат-бота Telegram**

Курсовая работа

Направление 02.03.02 - Фундаментальная информатика   
и информационные технологии

Профиль – «Инженерия программного обеспечения»

Зав. кафедрой \_\_\_\_\_\_\_\_\_ д. т. н., проф. Г.В. Абрамов

Обучающийся \_\_\_\_\_\_\_\_\_ студ. 3 к. 61 гр. И. Ю. Смехнев

Руководитель \_\_\_\_\_\_\_\_\_ к. ф.-м. н., доц. М. К. Чернышов

Воронеж 2024

Содержание

[Введение 3](#_Toc177555364)

[1. Постановка задачи 3](#_Toc177555365)

[2. Обоснование выбора работы 5](#_Toc177555366)

[2.1. Актуальность проблемы разработки информационных систем 5](#_Toc177555367)

[2.2. Сущность исследуемой проблемы и подходы к её решению 6](#_Toc177555368)

[2.3. Оценка различных подходов и обоснование выбранного метода 6](#_Toc177555369)

[2.4. Теоретическое обоснование исследований 7](#_Toc177555370)

[3. Алгоритмы и классы, используемые в проекте 8](#_Toc177555371)

[3.1. Обработка команд и событий в Telegram 8](#_Toc177555372)

[3.2. Выполнение SQL-запросов к базе данных 9](#_Toc177555373)

[3.3. Сохранение и удаление SQL-запросов 10](#_Toc177555374)

[3.4. Класс настроек 11](#_Toc177555375)

[4. Разработка и тестирование 12](#_Toc177555376)

[4.1. Выбор программного средства для реализации Telegram-бота 12](#_Toc177555377)

[4.2. Реализация 13](#_Toc177555378)

[4.3. Тестирование и результаты 18](#_Toc177555379)

[Заключение 22](#_Toc177555380)

[Список литературы 24](#_Toc177555381)

[Приложение 1 25](#_Toc177555382)

[Приложение 2 26](#_Toc177555383)

Введение

В условиях стремительного развития цифровых технологий и возрастания объёмов данных актуальной задачей становится разработка информационных систем, обеспечивающих удобный и безопасный доступ к управлению данными. В условиях удаленной работы и распределенных команд востребованы системы, которые позволяют удаленно взаимодействовать с базой данных, обеспечивая доступ к информации в реальном времени.

Основная проблема, которая решается в данной работе, заключается в организации удаленного и безопасного взаимодействия пользователей с базой данных PostgreSQL через мессенджер Telegram. Эта проблема актуальна для компаний, работающих с географически распределенными командами, и требует реализации системы, способной выполнять SQL-операции удаленно.

Проект направлен на создание информационной системы, интегрированной с Telegram, что позволяет пользователям работать с базой данных PostgreSQL без необходимости прямого физического подключения.

Структура работы включает введение, три главы и заключение. В первой главе рассматриваются теоретические аспекты разработки информационных систем, а также анализ применяемых технологий и алгоритмов. Во второй главе подробно описаны этапы реализации Telegram-бота, включая проектирование системы, реализацию алгоритмов и работу с базой данных. Третья глава посвящена тестированию системы и анализу её эффективности на различных этапах разработки. В заключении подведены итоги работы и обозначены перспективы для её дальнейшего развития.

1. Постановка задачи

Целью работы является создание Telegram-бота для удаленного управления базой данных PostgreSQL, обеспечивающего удобное и безопасное взаимодействие с данными для пользователей. Основная проблема заключается в организации эффективного удалённого взаимодействия с базой данных через мессенджер, что актуально для компаний, работающих в распределённых командах и требующих доступа к информации в реальном времени.

Для достижения поставленной цели необходимо решить следующие задачи:

1. **Проектирование архитектуры системы**: требуется спроектировать архитектуру взаимодействия между Telegram-ботом и базой данных PostgreSQL, обеспечив её надёжность и масштабируемость. Важно определить, как будет осуществляться обработка команд, передача данных и обратная связь с пользователем.
2. **Реализация взаимодействия Telegram-бота и базы данных**: основной задачей является создание программного кода, который будет обрабатывать пользовательские запросы через Telegram и выполнять соответствующие SQL-операции в базе данных. Это включает такие действия, как выборка данных (SELECT), вставка (INSERT), обновление (UPDATE) и удаление данных (DELETE).
3. **Реализация выполнения произвольных запросов**: также необходимо реализовать возможность ввода произвольных SQL-запросов пользователем. Это обеспечит гибкость и позволит пользователям выполнять более сложные операции над базой данных, выходящие за рамки стандартных CRUD-операций. Пользователь должен иметь возможность отправлять свои SQL-запросы через бот и получать результаты их выполнения. Также важно предусмотреть возможность сохранения часто используемых запросов. Эта функция позволит пользователям сохранять введённые SQL-запросы для их повторного использования в будущем, что значительно упростит выполнение рутинных операций. Сохранённые запросы должны быть легко доступны через команду бота, обеспечивая быстрое выполнение сложных SQL-операций без необходимости повторного ввода запроса.
4. **Настройка среды выполнения**: необходимо настроить среду для запуска Telegram-бота, чтобы он работал в фоновом режиме на сервере, обеспечивая бесперебойное функционирование. Это может потребовать использования таких инструментов, как pm2 или systemd, для управления процессом выполнения.
5. **Тестирование и оптимизация системы**: система должна быть протестирована на различных сценариях использования, включая типичные и нетипичные запросы пользователей. Необходимо оценить производительность, надёжность и безопасность системы, а также провести оптимизацию кода для повышения эффективности её работы.
6. **Разработка документации и рекомендаций по использованию**: завершающим этапом является создание документации, в которой будут описаны принципы работы системы, её функциональные возможности, а также инструкции по её установке и использованию.
7. Обоснование выбора работы
   1. Актуальность проблемы разработки информационных систем

В современном мире объем данных растет с огромной скоростью, что делает управление ими одной из важнейших задач для организаций. Эффективная работа с информацией позволяет компаниям повышать свою продуктивность и оперативность принятия решений. Информационные системы, автоматизирующие процессы управления данными, становятся неотъемлемой частью бизнес-процессов в условиях цифровизации.

Среди задач, стоящих перед разработчиками информационных систем, особое место занимает создание инструментов для удаленного доступа к данным. Это актуально для компаний, работающих в распределенных командах, а также для тех, кто нуждается в оперативном доступе к информации в условиях глобализации. Одним из решений данной проблемы является использование автоматизированных систем, таких как Telegram-боты, которые предоставляют удобный интерфейс для взаимодействия с базами данных.

* 1. Сущность исследуемой проблемы и подходы к её решению

Исследуемая проблема заключается в создании информационной системы для удаленного управления базой данных PostgreSQL. Разработка подобных систем может осуществляться различными подходами:

* **Веб-интерфейсы и API**. Классический подход заключается в создании веб-приложений или API для управления базами данных удаленно. Этот метод обеспечивает гибкий и мощный интерфейс, но требует специфических навыков работы с веб-технологиями от пользователя.
* **Интеграция с мессенджерами**. Мессенджеры становятся удобным средством взаимодействия с базами данных благодаря знакомому интерфейсу и простоте использования. Telegram-боты предоставляют возможность выполнения SQL-операций через текстовые команды, что делает их удобными для конечных пользователей.
* **Клиент-серверные приложения**. Специализированные клиентские приложения обеспечивают доступ к серверам баз данных, но их разработка и поддержка требуют дополнительных ресурсов, что не всегда оправдано.

Интеграция с мессенджерами, такими как Telegram, представляется наиболее сбалансированным подходом, сочетая простоту использования и достаточную функциональность для решения задач удаленного управления данными.

* 1. Оценка различных подходов и обоснование выбранного метода

Выбор метода разработки Telegram-бота для работы с базой данных PostgreSQL определяется несколькими ключевыми аспектами.

1. **Доступность и простота Telegram**: Telegram — это популярная и удобная платформа для создания ботов, предлагающая гибкое API. Пользователи могут взаимодействовать с ботом через знакомый интерфейс мессенджера, что снижает порог вхождения и упрощает работу с системой.
2. **Преимущества PostgreSQL**: PostgreSQL — мощная реляционная СУБД, которая обеспечивает высокую производительность, надежность и безопасность при работе с большими объемами данных. Благодаря поддержке сложных запросов и расширений, PostgreSQL является отличным выбором для систем, требующих гибкой работы с данными.
3. **Python как основной язык разработки**: Python — универсальный язык программирования с богатой экосистемой библиотек, таких как pyTelegramBotAPI и psycopg2, которые позволяют эффективно интегрировать бота с Telegram и PostgreSQL. Его простота и читаемость делают разработку быстрее, а наличие большого количества готовых решений ускоряет реализацию проекта.
4. **Гибкость системы**: использование динамически формируемых SQL-запросов и шаблонов позволяет системе адаптироваться под изменения в структуре базы данных, не требуя значительных изменений в коде. Это обеспечивает высокую гибкость и масштабируемость проекта, что важно для информационных систем, работающих в условиях динамично меняющихся требований.

Таким образом, выбор Telegram, PostgreSQL и Python как основных инструментов разработки информационной системы обусловлен их доступностью, гибкостью и производительностью, что делает их идеальными для решения задач удаленного управления данными через мессенджер.

* 1. Теоретическое обоснование исследований

Выбор платформы Telegram для создания информационной системы и использование Python для разработки обеспечивают необходимые условия для эффективного решения задачи удаленного управления базой данных. Анализ существующих подходов и реализация функциональных возможностей, основанного на динамическом формировании SQL-запросов, позволили создать удобную и безопасную систему для взаимодействия с данными через мессенджер.

Таким образом, Telegram-бот на базе Python и PostgreSQL представляет собой эффективное решение для удаленного управления базами данных, обеспечивая пользователям удобный доступ к информации из любой точки мира.

1. Алгоритмы и классы, используемые в проекте
   1. Обработка команд и событий в Telegram

Перед написанием кода необходимо спроектировать логику обработки взаимодействий с пользователями. Это можно изобразить в виде блок-схемы (Рис. 1) и описать так:

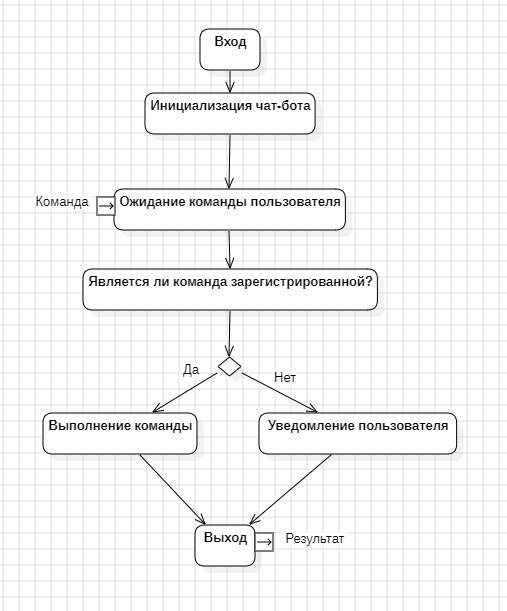


Рис. 1 Блок-схема алгоритма обработки команд и событий в Telegram

1. **Инициализация бота**: создание экземпляра бота для работы с API Telegram, настройка параметров.
2. **Обработка команд**: планирование возможных команд и их обработки (например, /start, /settings). Это может включать использование меню или текстовых сообщений.
3. **Взаимодействие с базой данных**: проектирование взаимодействия бота с базой данных для выполнения команд, передаваемых пользователями.
4. **Отправка ответов**: разработка системы обратной связи с пользователем в ответ на команды.

На данном этапе выбирается подход к реализации этих функций и рассматриваются возможные способы обработки команд через кнопки или текстовые сообщения.

* 1. Выполнение SQL-запросов к базе данных

Необходимо спроектировать систему выполнения запросов к базе данных. Алгоритм выполнения SQL-запросов можно изобразить в виде блок-схемы (Рис. 2) и описать так:

1. **Прием запросов**: пользователь должен ввести SQL-запрос в чат.
2. **Обработка данных**: продумывается система передачи и обработки SQL-запроса в базе данных, как бот будет отправлять запросы и получать результаты.
3. **Обработка ошибок**: планируется механизм обработки возможных ошибок выполнения SQL-запросов и уведомления пользователя.

Этот алгоритм может быть реализован для выполнения различных операций с базой данных, таких как запросы на выборку, вставку и обновление данных.

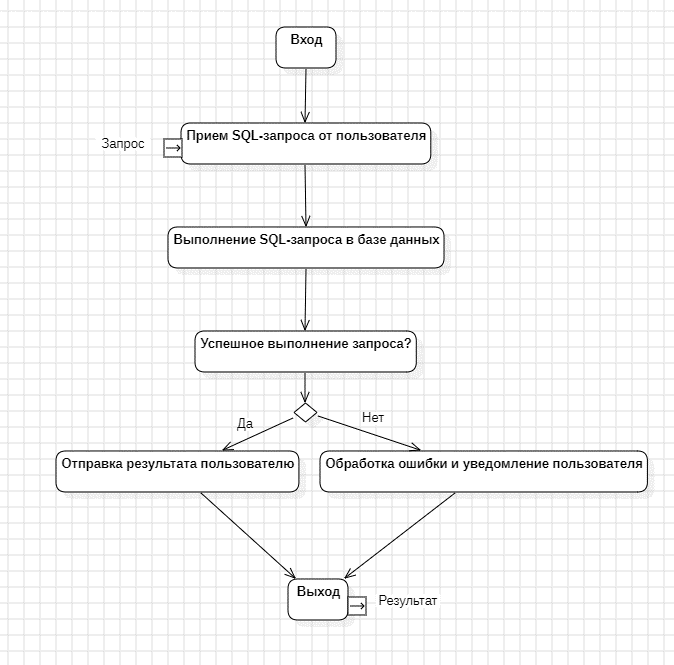


Рис.2 Блок-схема алгоритма выполнения SQL-запросов

* 1. Сохранение и удаление SQL-запросов

Проектирование алгоритма для сохранения и управления SQL-запросами может включать следующие этапы (Рис. 3):

1. **Сохранение запросов**: обдумывается, как запросы будут сохраняться в отдельном файле или базе данных для последующего использования.
2. **Удаление запросов**: разработка механизма удаления запросов пользователем, например, через команды или кнопки.

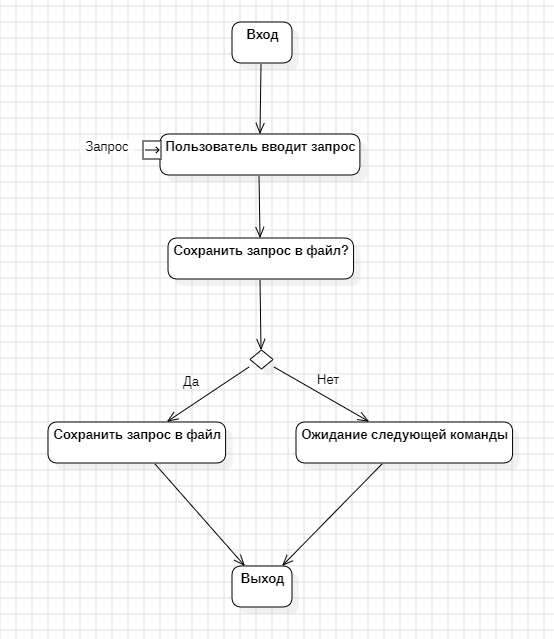


Рис. 2 Блок-схема алгоритма сохранения и удаления SQL-запросов

Этот алгоритм позволяет сохранять часто используемые запросы и управлять ими.

* 1. Класс настроек

Для более удобного взаимодействия с базой данных в программе должны быть предусмотрены настройки, такие как:

1. Формат вывода результата запроса SELECT (текстом в виде таблицы, либо файлом в формате .csv, .json и т.д.).
2. Опция вывода всех значений или только уникальных (DISTINCT).

Для управления этими параметрами необходимо создать отдельный класс, который будет хранить настройки. Важно, чтобы параметры были типа int, поскольку планируется добавлять возможность запрашивать эти настройки у пользователя после каждого запроса.

Примерная диаграмма класса представлена на рисунке 4.

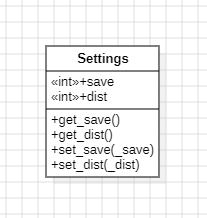


Рис. 3 Диаграмма класса настроек

1. Разработка и тестирование
   1. Выбор программного средства для реализации Telegram-бота

Программное обеспечение Telegram-бота функционирует в следующей среде:

* **Операционная система**: Linux (например, Ubuntu 20.04 и выше).
* **Язык программирования**: Python 3.8 и выше.
* **Среда разработки:** PyCharm;
* **Библиотеки**:
  + pyTelegramBotAPI — для взаимодействия с API Telegram;
  + psycopg2 — для работы с базой данных PostgreSQL;
  + python-dotenv – для хранения информации о подключении в защищенном файле
  + Prettytable – для создания генерации таблиц с помощью ASCII-символов
* **База данных:** PostgreSQL 13 и выше.

Программа запускается на сервере, где настроено фоновое выполнение процесса с использованием pm2 или systemd.

* 1. Реализация

Проект реализован в виде Telegram-бота на языке Python с использованием библиотеки telebot для взаимодействия с Telegram API и psycopg2 для работы с базой данных PostgreSQL. Основные компоненты проекта включают контроллер для обработки команд, сервисы для выполнения SQL-запросов, и слой базы данных.

* **Инициализация чат-бота**

Инициализация бота и подключение базы данных происходит при запуске программы, при этом создается экземпляр класса настроек

* **Реализация кастомных команд и кнопок**

Для работы с различными SQL-запросами в проекте были созданы кастомные команды, которые привязываются к кнопкам, используя InlineKeyboardMarkup и InlineKeyboardButton из библиотеки pyTelegramBotAPI. Эти команды управляют процессом выполнения SQL-запросов, взаимодействуя с пользователем через интерактивные кнопки и вызывая соответствующие callback-функции.

Основные этапы работы команд и кнопок:

1. **Создание команд**:
   * + Команды определяются в виде текстовых значений и используются для передачи информации о действии пользователя (например, выбор таблицы, столбца, тип операции).
     + Каждая команда связана с определенной функцией через callback, который обрабатывает конкретное действие.
2. **Привязка команд к кнопкам**:
   * + Для создания интерфейса с кнопками используется InlineKeyboardMarkup.
     + Например, при выборе таблицы генерируется список кнопок с названиями таблиц, и каждая кнопка содержит команду, передающую название выбранной таблицы в callback-функцию.
3. **Callback-функции**:
   * + Callback-функции обрабатывают команды, переданные через кнопки, и выполняют соответствующие действия.
     + Например, после выбора таблицы запускается функция, предлагающая пользователю выбрать столбцы, или программа переходит к следующему шагу, в зависимости от типа запроса.
4. **Динамическая работа с командами**:
   * + Все команды генерируются динамически в зависимости от действий пользователя. Например, после выбора таблицы генерируются команды для столбцов этой таблицы, что делает процесс гибким и удобным для пользователя.

Использование кастомных команд позволяет создавать удобный интерфейс для взаимодействия с базой данных через Telegram, где каждый шаг сопровождается интерактивными элементами, обеспечивая интуитивно понятный процесс выполнения запросов.

* **Реализация базовых SQL-операций**

Для взаимодействия с базой данных предусмотрены функции, выполняющие основные SQL-операции: вставка данных, обновление записей, удаление данных и выборка. Эти функции принимают параметры, необходимые для построения SQL-запросов, и выполняют их с помощью курсора PostgreSQL.

1. SELECT

При выборе пользователем команды SELECT, бот на pyTelegramBotAPI последовательно отправляет пользователю кнопки для выбора нужной таблицы из базы данных. Это реализуется с помощью кнопок InlineKeyboardMarkup и InlineKeyboardButton.

Порядок работы:

1. **Выбор таблицы**: после команды SELECT, бот отправляет кнопки с названием доступных таблиц. Пользователь выбирает нужную таблицу.
2. **Выбор столбцов**: после выбора таблицы бот отправляет кнопки с доступными столбцами этой таблицы, позволяя выбрать либо конкретные столбцы, либо все сразу.
3. **Формат вывода данных**: после выбора столбцов пользователю предоставляется выбор формата вывода данных — либо текстом в сообщении, либо сохранением в формате файла .csv.

Все шаги поддерживаются с помощью кнопок Inline, которые реагируют на действия пользователя через обработчиков событий.

1. INSERT

Процесс обработки запроса INSERT в боте на основе библиотеки pyTelegramBotAPI также осуществляется пошагово с использованием кнопок InlineKeyboardMarkup.

Порядок работы:

1. **Выбор таблицы**: как и в случае с запросом SELECT, бот отправляет пользователю кнопки с названием доступных таблиц после команды INSERT. Пользователь выбирает таблицу, в которую будет вставлена новая запись.
2. **Ввод значений**: после выбора таблицы бот запрашивает у пользователя ввод значений для каждого столбца таблицы. Это реализуется последовательным отправлением сообщений с запросами данных.
3. **Проверка данных**:
   * **Тип данных**: соответствуют ли введённые значения ожидаемым типам (например, целочисленным, строкам или датам).
   * **is\_nullable**: может ли поле принимать NULL значения. Если нет, пользователь обязан ввести значение.
   * **Ограничения**: такие как длина строки, диапазон чисел или уникальность значения.
4. **Подтверждение и вставка данных**: после ввода всех значений бот предлагает пользователю подтвердить ввод. Если пользователь подтверждает, данные отправляются в базу с использованием SQL-запроса INSERT.

Этот процесс также поддерживается обработчиками событий и кнопками Inline, что позволяет пользователю пошагово взаимодействовать с программой, обеспечивая удобное управление данными.

1. UPDATE

Обработка запроса UPDATE реализована аналогично процессам для SELECT и INSERT, с проверкой введенных пользователем данных.

Основные этапы:

1. **Выбор таблицы**: пользователь выбирает таблицу через кнопки InlineKeyboardMarkup.
2. **Выбор столбца**: после выбора таблицы программа предлагает выбрать столбцы, которые необходимо обновить.
3. **Ввод значений и проверка данных**:
   * **Тип данных**: проверка типа данных, чтобы соответствовать структуре столбца.
   * **is\_nullable**: проверяется, допускает ли столбец NULL.
4. **Подтверждение и обновление**: пользователь подтверждает изменения, и данные отправляются на выполнение запроса UPDATE в базу данных.
5. DELETE

Процесс выполнения команды DELETE реализован с несколькими возможностями: удаление строки, столбца или целой таблицы. Взаимодействие с пользователем осуществляется через кнопки InlineKeyboardMarkup.

Основные этапы:

1. **Опции удаления**:
   * **Удаление строки**: пользователь указывает условия для удаления строки (например, по определенному значению в столбце).
   * **Удаление столбца**: программа предлагает выбрать столбец для удаления.
   * **Удаление всей таблицы**: программа запрашивает подтверждение на удаление всей таблицы.
2. **Подтверждение удаления**: для каждого типа удаления программа запрашивает подтверждение от пользователя.
3. **Выполнение команды**: после подтверждения команда DELETE выполняется и данные удаляются из базы.

* **Реализация выполнения любого SQL запроса**

Помимо базовых операций, пользователь может выполнять любые запросы разной сложности. Это могут быть сложные выборки с использованием различных условий, сортировок, группировок и агрегатных функций. Функция поддерживает вложенные запросы, соединения таблиц, создание и управление представлениями.

Также для удобства пользователей бот позволяет сохранять часто используемые запросы и быстро их выполнять. Это особенно полезно для повторяющихся операций, таких как регулярные отчеты или часто используемые выборки данных.

1. Сохранение запросов

Пользователь может сохранить запрос с удобным для него именем. Это позволяет легко идентифицировать и выполнять запросы в будущем. Сохранение запроса осуществляется с помощью функции save\_request, которая записывает запрос и его имя в CSV файл.

1. Быстрое выполнение сохраненных запросов

Пользователь может быстро выполнить ранее сохраненный запрос, выбрав его из списка. Функция get считывает запрос из CSV файла по имени и возвращает его для выполнения.

* 1. Тестирование и результаты

Для тестирования работоспособности чат-бота были составлены специальная тестовая база данных (Приложение 1).

Тестирование проводилось для проверки корректности выполнения SQL-запросов, а также для проверки функциональности сохранения и быстрого выполнения запросов.

**Тестовые данные и сценарии**

Для тестирования были использованы следующие тестовые данные и сценарии:

* **Тестовые данные №1**:

**Описание**: проверка выборки данных из столбца date\_of\_birth таблицы students

**Ожидаемый** **результат**: список дат рождений.

* **Тестовые данные №2:**

**Описание**: проверка вставки нового пользователя в таблицу students.

**Ожидаемый** **результат**: успешная вставка записи и подтверждение выполнения.

* **Тестовые данные №3:**

**Описание**: проверка обновления данных пользователя в таблице students.

**Ожидаемый** **результат**: успешное обновление возраста пользователя.

* **Тестовые данные №4:**

**Описание**: проверка удаления пользователя из таблицы students.

**Ожидаемый** **результат**: успешное удаление записи и подтверждение выполнения.

* **Тестовые данные №5:**

**Описание:** проверка выполнения собственного запроса SELECT \* FROM students WHERE date\_of\_birth > ‘2000-01-01’;

**Ожидаемый результат:** успешное выполнение запроса

* **Тестовые данные №6:**

**Описание:** проверка выполнения тестового запроса с ошибкой SELECT \* FROM users;

**Ожидаемый результат:** сообщение ошибки: таблицы с таким названием не существует

**Результаты тестирования:**

* **Тестовые данные №1:**

**Результат**: запрос выполнен успешно. Получен список дат рождений

**SQL-запрос:** SELECT date\_of\_birth FROM students;

**Сообщение** **бота**: (Рис. 5)

* **Тестовые данные №2:**

**Результат**: запрос выполнен успешно. Новый пользователь добавлен в таблицу.

**SQL-запрос**: INSERT INTO students (first\_name, last\_name, date\_of\_birth, email) VALUES (‘Ivan’, ‘Ivanov’, ‘2001-09-11’, ‘i\_ivanov@gmail.com’);

**Сообщение** **бота**: “Запись успешно добавлена.”

* **Тестовые данные №3:**

**Результат**: запрос выполнен успешно. Возраст пользователя обновлен.

**SQL-запрос**: UPDATE students SET email = ‘iivanov@temp.com’ WHERE first\_name = ‘Ivan' AND last\_name = ‘Ivanov’;

**Сообщение** **бота**: “Запись успешно обновлена.”

* **Тестовые данные №4:**

**Результат**: запрос выполнен успешно. Пользователь удален из таблицы.

**SQL-запрос**: DELETE FROM students WHERE first\_name = 'Ivan' AND last\_name = ‘Ivanov’;

**Сообщение** **бота**: “Запись успешно удалена.”

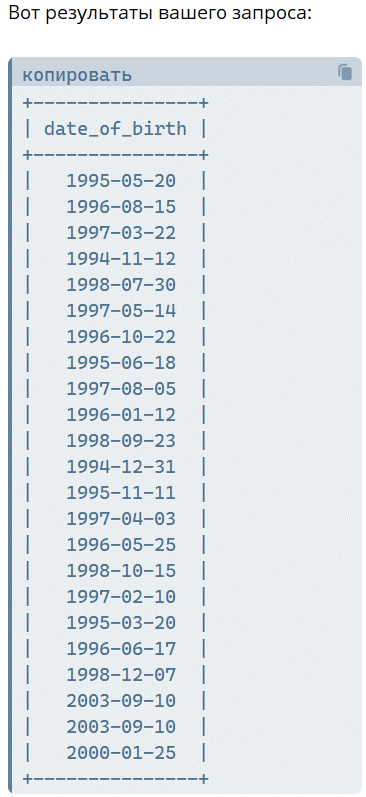


Рис. 5. Тест №1

* **Тестовые данные №5:**

**Результат:** запрос выполнен успешно. Получен список студентов

**Сообщение бота:** (Рис 6)

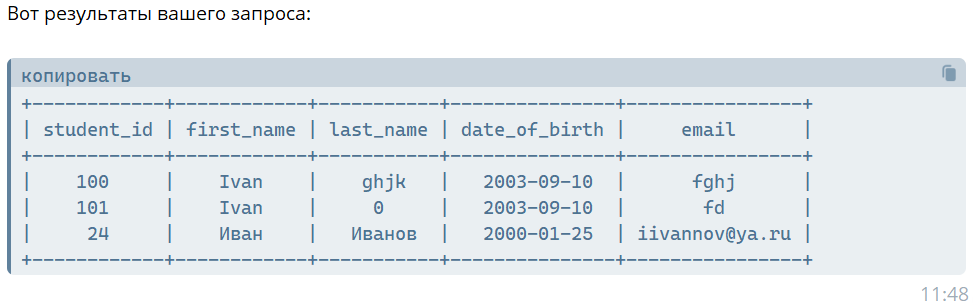


Рис. 6. Тест № 5

* **Тестовые данные №6:**

**Результат:** бот отправил сообщение ошибки

**Сообщение бота:** произошла ошибка: relation "users" does not exist

LINE 1: SELECT \* FROM users;

Заключение

В ходе выполнения данной работы была разработана информационная система в виде Telegram-бота для удаленного управления базой данных PostgreSQL. Основной целью работы было создание удобного и безопасного инструмента для выполнения различных SQL-операций без необходимости прямого физического подключения к серверам.

В процессе работы были решены следующие задачи:

* Исследованы возможности и ограничения API Telegram для реализации возможности удаленного управления данными.
* Разработана архитектура взаимодействия между ботом и базой данных.
* Реализованы механизмы безопасности для предотвращения несанкционированного доступа к базе данных.
* Проведено тестирование системы на различных сценариях использования и оценена её эффективность.
* Проанализированы возможности расширения бота для решения более сложных задач в будущем.

Результаты тестирования показали, что разработанный Telegram-бот успешно выполняет поставленные задачи, обеспечивая удобный и безопасный доступ к управлению базой данных PostgreSQL. Система продемонстрировала стабильную работу при выполнении различных операций с базой данных, а также возможность сохранения и быстрого выполнения часто используемых запросов.

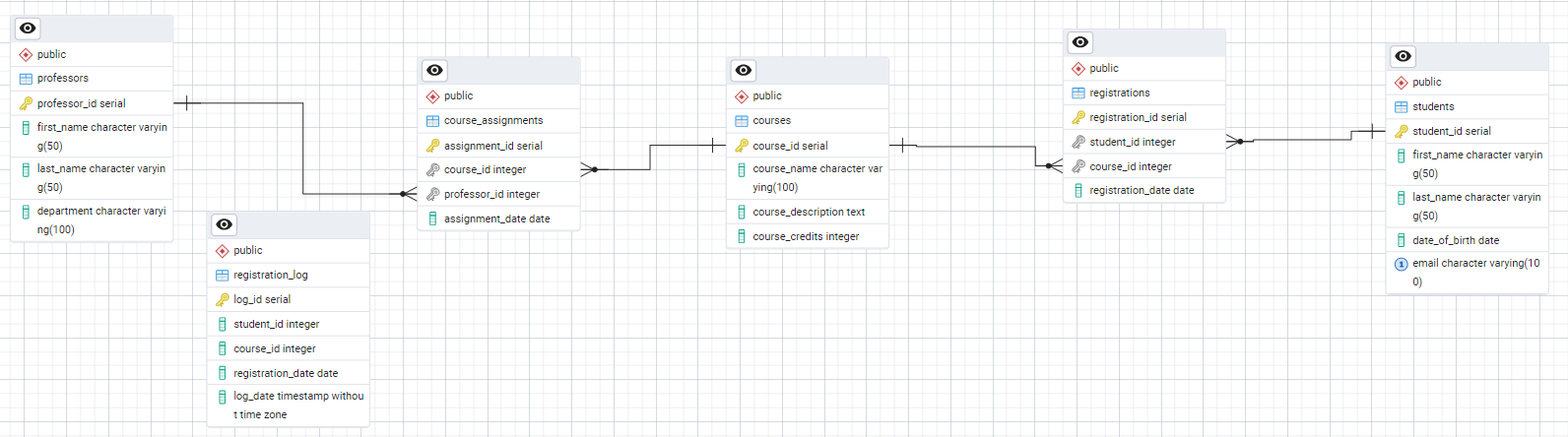
Таким образом, разработанная система может быть полезна для компаний и проектов, работающих в условиях географически распределенных команд или требующих доступа к информации в реальном времени. В будущем возможно расширение функциональных возможностей бота для решения более сложных задач и интеграция с другими системами и сервисами.

Работа над проектом позволила углубить знания в области разработки информационных систем, использования API Telegram и управления базами данных PostgreSQL. Полученные результаты могут быть использованы для дальнейших исследований и разработки новых решений в области удаленного управления данными.

Список литературы

1. Горячев, В. С. Базы данных: учебное пособие / В. С. Горячев. — М.: Альфа-М, 2014. — 352 с.
2. Сильвестров, С. Базы данных: проектирование, внедрение, эксплуатация / С. Сильвестров. — СПб.: Питер, 2016. — 256 с.
3. Стивенс, М. Основы Python: Руководство по программированию / М. Стивенс. — М.: Вильямс, 2017. — 412 с.
4. PostgreSQL Documentation [Электронный ресурс] – URL: <https://www.postgresql.org/docs/>.
5. pyTelegramBotAPI Documentation [Электронный ресурс] – URL: <https://github.com/eternnoir/pyTelegramBotAPI>.

Приложение 1



Приложение 2

**Файл main.py:**

import MyKeyboards

import sql\_delete

import sql\_insert

import sql\_request

import sql\_update

import sql\_select

from config import \*

request\_file = '../request.csv'

@bot.message\_handler(commands=["start"])

def com\_start(message):

bot.send\_message(message.chat.id, "Hello", reply\_markup=MyKeyboards.main())

@bot.message\_handler(commands=["help"])

def com\_help(message):

bot.send\_message(message.chat.id, "<b>Добро пожаловать!</b>\n"

"<i>Этот бот позволяет выполнять различные операции с базой данных PostgreSQL. Вот доступные команды и их возможности:</i>\n"

"\n"

"<b>/start</b>: Начните работу с ботом. Эта команда выводит приветственное сообщение и главное меню.\n"

"<b>/help</b>: Получите список всех доступных команд и их описание.\n"

"\n"

"<b>Основные команды:</b>\n"

"\n"

"1. <b>SELECT</b>:\n"

" - Выбор и отображение данных из таблиц.\n"

" - Включает возможность выбора отдельных колонок и использования DISTINCT для удаления дубликатов.\n"

"\n"

"2. <b>CREATE</b>:\n"

" - Создание новых таблиц в базе данных.\n"

" - Добавление новых записей в существующие таблицы.\n"

"\n"

"3. <b>INSERT</b>:\n"

" - Вставка новых данных в таблицы.\n"

" - Поддержка ввода значений по умолчанию, проверка типов данных и длины значений.\n"

"\n"

"4. <b>UPDATE</b>:\n"

" - Обновление существующих данных в таблицах.\n"

" - Возможность выбора таблицы, колонки и строки для обновления.\n"

" - Проверка на допустимость значений и типов данных.\n"

"\n"

"5. <b>DELETE</b>:\n"

" - Удаление данных из таблиц.\n"

" - Возможность удаления целых таблиц, отдельных колонок или строк по первичному ключу.\n"

" - Подтверждение удаления для предотвращения случайных действий.\n"

"\n"

"<b>Дополнительные функции:</b>\n"

"\n"

"- <b>Валидация данных</b>:\n"

" - Проверка типов данных и максимальной длины значений для вставки и обновления.\n"

" - Поддержка различных типов данных, таких как integer, numeric, boolean, date, timestamp и текстовые поля.\n"

"\n"

"- <b>Работа с ключами</b>:\n"

" - Выбор и использование первичных ключей для операций обновления и удаления.\n"

" - Автоматическое определение и отображение первичных ключей таблиц.\n"

"\n"

"<b>Пример использования</b>:\n"

"- Введите команду <b>INSERT</b> для начала процесса вставки данных в таблицу.\n"

"- Выберите таблицу и введите значения для каждой колонки по очереди.\n"

"- Бот проверит введенные значения и вставит их в таблицу.\n"

"\n"

"<i>Если у вас возникли вопросы или вам нужна дополнительная помощь, обратитесь к документации или свяжитесь с поддержкой.</i>\n",

parse\_mode='HTML', reply\_markup=MyKeyboards.main())

@bot.message\_handler(content\_types=["text"])

def com\_text(message):

if message.text == "SELECT":

bot.send\_message(message.chat.id, "Выберите таблицу для выполнения команды SELECT.", reply\_markup=MyKeyboards.tables("select"))

elif message.text == "INSERT":

bot.send\_message(message.chat.id, "Выберите таблицу для вставки данных.", reply\_markup=MyKeyboards.tables("insert"))

elif message.text == "UPDATE":

bot.send\_message(message.chat.id, "Выберите таблицу для обновления данных.", reply\_markup=MyKeyboards.tables("update"))

elif message.text == "DELETE":

sql\_delete.start\_delete(message)

elif message.text == "Выполнить":

msg = bot.send\_message(message.chat.id, "Напиши свой запрос.")

bot.register\_next\_step\_handler(msg, sql\_request.one)

elif message.text == "Запросы":

msg = bot.send\_message(message.chat.id, "Запросы", reply\_markup=MyKeyboards.requests(request\_file))

bot.register\_next\_step\_handler(msg, sql\_request.one, True)

elif message.text == "Настройки":

com\_settings(message)

elif message.text == "Помощь":

com\_help(message)

def com\_settings(message):

bot.send\_message(message.chat.id, "Настройки", reply\_markup=MyKeyboards.settings())

@bot.callback\_query\_handler(func = lambda call: call.data.startswith("settings#save") or call.data.startswith("settings#dist"))

def call\_settings(call):

if call.data == "settings#save#0":

bot\_settings.set\_save(0)

elif call.data == "settings#save#1":

bot\_settings.set\_save(1)

elif call.data == "settings#save#2":

bot\_settings.set\_save(2)

elif call.data == "settings#dist#0":

bot\_settings.set\_dist(0)

elif call.data == "settings#dist#1":

bot\_settings.set\_dist(1)

elif call.data == "settings#dist#2":

bot\_settings.set\_dist(2)

bot.edit\_message\_text("Настройки", call.message.chat.id, call.message.message\_id, reply\_markup=MyKeyboards.settings())

bot.polling()

**Файл config.py**

import csv

import os

import psycopg2

import telebot

from prettytable import PrettyTable

from Settings import Settings

from dotenv import load\_dotenv

load\_dotenv(dotenv\_path='../config.env')

conn = psycopg2.connect(dbname=os.getenv("DB\_NAME"), user=os.getenv("DB\_USER"), password=os.getenv("DB\_PASSWORD"), host=os.getenv("DB\_HOST"), port=os.getenv("DB\_PORT"))

token = os.getenv("TELEGRAM\_TOKEN")

bot = telebot.TeleBot(token)

bot\_settings = Settings()

def list\_to\_str(list, colnames) -> str:

table = PrettyTable()

table.field\_names = colnames

for row in list:

table.add\_row(row)

# Отправляем таблицу в Telegram

table\_str = table.get\_string()

return table\_str

def list\_to\_csv(list, colnames, file\_path) -> None:

# Открытие файла и запись данных в CSV

with open(f"./{file\_path}", mode='w', newline='', encoding='utf-8') as file:

writer = csv.writer(file)

# Запись заголовков колонок

writer.writerow(colnames)

# Запись данных

writer.writerows(list)

**Файл MyKeyboards.py**

from telebot import types

import Postgres

from config import \*

def main():

main\_key = types.ReplyKeyboardMarkup(True, True)

main\_key.row('SELECT', 'INSERT')

main\_key.row('UPDATE', 'DELETE')

main\_key.row('Выполнить', 'Запросы')

main\_key.row('Настройки', 'Помощь')

return main\_key

def tables(prefix):

tables\_key = types.InlineKeyboardMarkup()

for table in Postgres.get\_tables():

tables\_key.add(types.InlineKeyboardButton(table, callback\_data=prefix + "#" + table))

return tables\_key

def select\_columns(table):

columns\_key = types.InlineKeyboardMarkup()

for column in Postgres.get\_columns(table):

columns\_key.add(types.InlineKeyboardButton(column, callback\_data="select#" + table + "@" + column))

columns\_key.add(types.InlineKeyboardButton("ALL", callback\_data="select#" + table + "@\*"))

return columns\_key

def update\_columns(table, primary\_key):

columns\_key = types.InlineKeyboardMarkup()

for column in Postgres.get\_columns(table):

columns\_key.add(types.InlineKeyboardButton(column, callback\_data="update#" + table + "@" + primary\_key + "@" + column))

return columns\_key

def requests():

request\_key = types.ReplyKeyboardMarkup(True, True)

request\_key.row('Выполнить')

request\_key.row('Создать')

request\_key.row('Удалить')

def settings():

settings\_key = types.InlineKeyboardMarkup()

if bot\_settings.get\_save() == 0:

settings\_key.add(types.InlineKeyboardButton("Вывод: текстом в сообщение", callback\_data="settings#save#1"))

elif bot\_settings.get\_save() == 1:

settings\_key.add(types.InlineKeyboardButton("Вывод: в csv файл", callback\_data="settings#save#2"))

else:

settings\_key.add(types.InlineKeyboardButton("Вывод: спрашивать", callback\_data="settings#save#0"))

if bot\_settings.get\_dist() == 0:

settings\_key.add(types.InlineKeyboardButton("Значения: все", callback\_data="settings#dist#1"))

elif bot\_settings.get\_dist() == 1:

settings\_key.add(types.InlineKeyboardButton("Значения: только уникальные", callback\_data="settings#dist#2"))

else:

settings\_key.add(types.InlineKeyboardButton("Значения: спрашивать", callback\_data="settings#dist#0"))

settings\_key.add(types.InlineKeyboardButton("Создать запрос", callback\_data="settings#request#new"))

settings\_key.add(types.InlineKeyboardButton("Удалить запрос", callback\_data="settings#request#del"))

settings\_key.add(types.InlineKeyboardButton("Удалить все запросы", callback\_data="settings#request#delall"))

return settings\_key

def requests(request\_file):

requests\_key = types.ReplyKeyboardMarkup(True, True)

with open(request\_file, mode='r', newline='', encoding='utf-8') as file:

reader = csv.reader(file)

next(reader)

for row in reader:

requests\_key.row(row[1])

requests\_key.row('Назад')

return requests\_key

**Файл Settings.py**

class Settings:

def \_\_init\_\_(self, save: int = 2, dist: int = 2):

self.save = save

self.dist = dist

def get\_save(self) -> int:

return self.save

def get\_dist(self) -> int:

return self.dist

def set\_save(self, save) -> None:

self.save = save

def set\_dist(self, dist) -> None:

self.dist = dist

**Файл sql\_select.py**

import os

import re

import MyKeyboards

from config import \*

@bot.callback\_query\_handler(func=lambda call: call.data.startswith("select#"))

def call\_select(call):

if len(call.data.split("@")) == 1:

table = call.data.split("#")[1]

bot.send\_message(call.message.chat.id, "Пожалуйста, выберите столбцы для вашего запроса.", reply\_markup=MyKeyboards.select\_columns(table))

else:

table = re.sub("select#", "", call.data).split("@")[0]

column = re.sub("select#", "", call.data).split("@")[1]

if bot\_settings.get\_dist() == 0:

sql\_select(call.message, table, column)

elif bot\_settings.get\_dist() == 1:

sql\_select(call.message, table, column, True)

else:

msg = bot.send\_message(call.message.chat.id, f"Хотите выбрать уникальные значения? (Y/N)")

bot.register\_next\_step\_handler(msg, dist\_select, table, column)

def dist\_select(message, table, column):

if message.text == "Y" or message.text == "y":

sql\_select(message, table, column, True)

else:

sql\_select(message, table, column)

def sql\_select(message, table, column, distinct=False):

try:

cur = conn.cursor()

if distinct:

cur.execute("SELECT DISTINCT " + column + " FROM " + table)

else:

cur.execute("SELECT " + column + " FROM " + table)

rows = cur.fetchall()

except Exception as e:

bot.send\_message(message.chat.id, f"Произошла ошибка: {e}", parse\_mode='HTML')

return

cols = [desc[0] for desc in cur.description]

if bot\_settings.get\_save() == 0:

table\_str = list\_to\_str(rows, cols)

bot.send\_message(message.chat.id, f"Вот результаты вашего запроса:\n\n<pre>{table\_str}</pre>", parse\_mode='HTML')

elif bot\_settings.get\_save() == 1:

list\_to\_csv(rows, cols, f"{table}\_{column if column != '\*' else 'all'}.csv")

bot.send\_document(message.chat.id, open(f"{table}\_{column if column != '\*' else 'all'}.csv", "rb"))

os.remove(f"{table}\_{column if column != '\*' else 'all'}.csv")

else:

msg = bot.send\_message(message.chat.id, "Вывести в сообщении или сохранить в файл? (M/F)")

bot.register\_next\_step\_handler(msg, save\_select, table, column, rows, cols)

def save\_select(message, table, column, rows, cols):

if message.text == "M":

table\_str = list\_to\_str(rows, cols)

bot.send\_message(message.chat.id, f"Вот результаты вашего запроса:\n\n<pre>{table\_str}</pre>", parse\_mode='HTML')

else:

list\_to\_csv(rows, cols, f"{table}\_{column if column != '\*' else 'all'}.csv")

bot.send\_document(message.chat.id, open(f"{table}\_{column if column != '\*' else 'all'}.csv", "rb"))

os.remove(f"{table}\_{column if column != '\*' else 'all'}.csv")

**Файл sql\_insert.py**

import Postgres

from config import \*

import MyKeyboards

@bot.callback\_query\_handler(func=lambda call: call.data.startswith("insert#"))

def call\_insert(call):

sql\_insert(call.message, call.data.split("#")[1], [])

def sql\_insert(message, table, data, current\_column=None):

columns = Postgres.get\_columns(table)

if current\_column is None:

current\_column = columns[0]

else:

info = Postgres.get\_column\_info(table, current\_column)

if not info['is\_nullable'] and message.text == 'NULL':

bot.send\_message(message.chat.id, f"Значение для столбца '{current\_column}' не может быть NULL. Пожалуйста, введите значение.")

msg = bot.send\_message(message.chat.id, f"INSERT INTO {table} ({current\_column})\n\n<pre>{info}</pre>",

parse\_mode='HTML')

bot.register\_next\_step\_handler(msg, sql\_insert, table, data, current\_column)

return

elif not Postgres.validate\_input(message.text, info['data\_type'], info['character\_maximum\_length']) and message.text != 'DEFAULT':

bot.send\_message(message.chat.id,

f"Недопустимый тип данных или длина для столбца '{current\_column}'. Ожидается {info['data\_type']} с максимальной длиной {info['character\_maximum\_length']}.")

msg = bot.send\_message(message.chat.id, f"INSERT INTO {table} ({current\_column})\n\n<pre>{info}</pre>",

parse\_mode='HTML')

bot.register\_next\_step\_handler(msg, sql\_insert, table, data, current\_column)

return

else:

if message.text == 'DEFAULT' and info['column\_default'] is not None:

data.append("DEFAULT")

else:

data.append(message.text)

cc\_index = columns.index(current\_column)

if cc\_index < len(columns) - 1:

current\_column = columns[cc\_index + 1]

else:

return end\_insert(message, table, data)

info = Postgres.get\_column\_info(table, current\_column)

msg = bot.send\_message(message.chat.id, f"Введите значение для столбца '{current\_column}':\n\n<pre>{info}</pre>", parse\_mode='HTML')

bot.register\_next\_step\_handler(msg, sql\_insert, table, data, current\_column)

def end\_insert(message, table, data):

data = tuple(data)

info = ', '.join([str(i) for i in tuple(Postgres.get\_columns(table))])

values = []

for item in data:

if item == 'DEFAULT':

values.append('DEFAULT')

else:

values.append(f"'{item}'")

values\_str = ', '.join(values)

try:

cur = conn.cursor()

script = f"INSERT INTO {table} ({info}) VALUES ({values\_str})"

cur.execute(script)

conn.commit()

bot.send\_message(message.chat.id, "Данные успешно вставлены в таблицу.", reply\_markup=MyKeyboards.main())

except Exception as e:

bot.send\_message(message.chat.id, f"Произошла ошибка: {e}", parse\_mode='HTML')

return

**Файл sql\_update.py**

import re

import MyKeyboards

import Postgres

from config import \*

from telebot import types

@bot.callback\_query\_handler(func=lambda call: call.data.startswith("update#"))

def call\_update(call):

if len(call.data.split("@")) == 1:

table = call.data.split("#")[1]

primary\_keys\_update = types.InlineKeyboardMarkup()

for key in Postgres.get\_data\_from\_column(table, Postgres.get\_primary\_keys(table)):

primary\_keys\_update.add(types.InlineKeyboardButton(key, callback\_data="update#" + table + "@" + str(key)))

msg = bot.send\_message(call.message.chat.id, "Пожалуйста, выберите первичный ключ.", reply\_markup=primary\_keys\_update)

elif len(call.data.split("@")) == 2:

table = re.sub("update#", "", call.data).split("@")[0]

primary\_key = re.sub("update#", "", call.data).split("@")[1]

bot.send\_message(call.message.chat.id, "Пожалуйста, выберите столбец для обновления.", reply\_markup=MyKeyboards.update\_columns(table, primary\_key))

else:

table = re.sub("update#", "", call.data).split("@")[0]

primary\_key = re.sub("update#", "", call.data).split("@")[1]

column = re.sub("update#", "", call.data).split("@")[2]

msg = bot.send\_message(call.message.chat.id,

f"Введите новое значение для столбца {column}. Текущее значение: {Postgres.get\_data\_by\_key\_from\_column(table, column, primary\_key)}.\n\n"

f"<pre>{Postgres.get\_column\_info(table, column)}</pre>", parse\_mode='HTML')

bot.register\_next\_step\_handler(msg, sql\_update, table, column, primary\_key)

def sql\_update(message, table, column, primary\_key):

info = Postgres.get\_column\_info(table, column)

new\_value = message.text

if not info['is\_nullable'] and new\_value == 'NULL':

msg = bot.send\_message(message.chat.id, "Значение для столбца не может быть NULL. Пожалуйста, введите новое значение.")

bot.register\_next\_step\_handler(msg, sql\_update, table, column, primary\_key)

return

if not Postgres.validate\_input(new\_value, info['data\_type'], info['character\_maximum\_length']):

msg = bot.send\_message(message.chat.id, "Недопустимый ввод. Пожалуйста, введите новое значение.")

bot.register\_next\_step\_handler(msg, sql\_update, table, column, primary\_key)

return

if message.text == 'DEFAULT' and info['column\_default'] is not None:

new\_value = info['column\_default']

elif info['data\_type'] in ['character varying', 'text', 'date', 'timestamp', 'timestamp with time zone', 'timestamp without time zone']:

new\_value = f"'{new\_value}'"

# Construct the SQL update statement

try:

cur = conn.cursor()

primary\_key\_column = Postgres.get\_primary\_keys(table)

update\_query = f"""

UPDATE {table}

SET {column} = {new\_value}

WHERE {primary\_key\_column} = '{primary\_key}';

"""

cur.execute(update\_query)

conn.commit()

bot.send\_message(message.chat.id, "Данные успешно обновлены.")

except Exception as e:

bot.send\_message(message.chat.id, f"Произошла ошибка: {e}", parse\_mode='HTML')

return

**Файл sql\_delete.py**

from telebot import types

import Postgres

from config import \*

def start\_delete(message):

delete\_menu = types.InlineKeyboardMarkup()

delete\_menu.add(types.InlineKeyboardButton("TABLE", callback\_data="delete#table"))

delete\_menu.add(types.InlineKeyboardButton("COLUMN", callback\_data="delete#column"))

delete\_menu.add(types.InlineKeyboardButton("ROW", callback\_data="delete#row"))

bot.send\_message(message.chat.id, "Выберите тип удаления: таблица, столбец или строка.", reply\_markup=delete\_menu)

@bot.callback\_query\_handler(func=lambda call: call.data.startswith("delete#"))

def call\_delete(call):

if len(call.data.split("@")) == 1 and call.data.startswith("delete#table"):

tables\_key = types.InlineKeyboardMarkup()

for table in Postgres.get\_tables():

tables\_key.add(types.InlineKeyboardButton(table, callback\_data=call.data + "@" + table))

bot.send\_message(call.message.chat.id, "Выберите таблицу для удаления.", reply\_markup=tables\_key)

elif len(call.data.split("@")) == 2 and call.data.startswith("delete#table"):

yn\_menu = types.InlineKeyboardMarkup()

yn\_menu.add(types.InlineKeyboardButton("Да", callback\_data=call.data + "@Y"))

yn\_menu.add(types.InlineKeyboardButton("Нет", callback\_data=call.data + "@N"))

msg = bot.send\_message(call.message.chat.id, "Вы уверены, что хотите удалить таблицу?", reply\_markup=yn\_menu)

elif len(call.data.split("@")) == 3 and call.data.startswith("delete#table") and call.data.split("@")[2] == "Y":

try:

cur = conn.cursor()

cur.execute(f"DROP TABLE {call.data.split("@")[1]} CASCADE")

conn.commit()

bot.send\_message(call.message.chat.id, "Таблица успешно удалена.")

except Exception as e:

bot.send\_message(call.message.chat.id, f"Произошла ошибка: {e}", parse\_mode='HTML')

start\_delete(call.message)

elif len(call.data.split("@")) == 1 and call.data.startswith("delete#column"):

tables\_key = types.InlineKeyboardMarkup()

for table in Postgres.get\_tables():

tables\_key.add(types.InlineKeyboardButton(table, callback\_data=call.data + "@" + table))

bot.send\_message(call.message.chat.id, "Выберите таблицу для удаления.", reply\_markup=tables\_key)

elif len(call.data.split("@")) == 2 and call.data.startswith("delete#column"):

columns\_key = types.InlineKeyboardMarkup()

for column in Postgres.get\_columns(call.data.split("@")[1]):

columns\_key.add(types.InlineKeyboardButton(column, callback\_data=call.data + "@" + column))

bot.send\_message(call.message.chat.id, "Выберите столбец для удаления.", reply\_markup=columns\_key)

elif len(call.data.split("@")) == 3 and call.data.startswith("delete#column"):

yn\_menu = types.InlineKeyboardMarkup()

yn\_menu.add(types.InlineKeyboardButton("Да", callback\_data=call.data + "@Y"))

yn\_menu.add(types.InlineKeyboardButton("Нет", callback\_data=call.data + "@N"))

msg = bot.send\_message(call.message.chat.id, "Вы уверены, что хотите удалить столбец?", reply\_markup=yn\_menu)

elif len(call.data.split("@")) == 4 and call.data.startswith("delete#column") and call.data.split("@")[3] == "Y":

try:

cur = conn.cursor()

cur.execute(f"ALTER TABLE {call.data.split("@")[1]} DROP COLUMN {call.data.split("@")[2]}")

conn.commit()

bot.send\_message(call.message.chat.id, "Столбец успешно удален.")

except Exception as e:

bot.send\_message(call.message.chat.id, f"Произошла ошибка: {e}", parse\_mode='HTML')

return

elif len(call.data.split("@")) == 1 and call.data.startswith("delete#row"):

tables\_key = types.InlineKeyboardMarkup()

for table in Postgres.get\_tables():

tables\_key.add(types.InlineKeyboardButton(table, callback\_data=call.data + "@" + table))

bot.send\_message(call.message.chat.id, "Выберите таблицу для удаления.", reply\_markup=tables\_key)

elif len(call.data.split("@")) == 2 and call.data.startswith("delete#row"):

primary\_keys\_update = types.InlineKeyboardMarkup()

table = call.data.split("@")[1]

for key in Postgres.get\_data\_from\_column(table, Postgres.get\_primary\_keys(table)):

primary\_keys\_update.add(types.InlineKeyboardButton(key, callback\_data=call.data + "@" + str(key)))

bot.send\_message(call.message.chat.id, "Выберите значение первичного ключа для удаления строки.", reply\_markup=primary\_keys\_update)

elif len(call.data.split("@")) == 3 and call.data.startswith("delete#row"):

yn\_menu = types.InlineKeyboardMarkup()

yn\_menu.add(types.InlineKeyboardButton("Да", callback\_data=call.data + "@Y"))

yn\_menu.add(types.InlineKeyboardButton("Нет", callback\_data=call.data + "@N"))

msg = bot.send\_message(call.message.chat.id, "Вы уверены, что хотите удалить строку?", reply\_markup=yn\_menu)

elif len(call.data.split("@")) == 4 and call.data.startswith("delete#row") and call.data.split("@")[3] == "Y":

try:

cur = conn.cursor()

cur.execute(f"DELETE FROM {call.data.split("@")[1]} WHERE {Postgres.get\_primary\_keys(call.data.split("@")[1])} = '{call.data.split("@")[2]}'")

conn.commit()

bot.send\_message(call.message.chat.id, "Строка успешно удалена.")

except Exception as e:

bot.send\_message(call.message.chat.id, f"Произошла ошибка: {e}", parse\_mode='HTML')

return

**Файл sql\_request.py**

from config import \*

import csv

from telebot import types

import MyKeyboards

request\_file = '../request.csv'

def one(message, name = False):

if message.text == "Назад":

bot.send\_message(message.chat.id, "Отменено", reply\_markup=MyKeyboards.main())

return

request = get(message.text) if name else message.text

bot.send\_message(message.chat.id, execute(request), parse\_mode='HTML', reply\_markup=MyKeyboards.main())

@bot.callback\_query\_handler(func = lambda call: call.data.startswith("settings#request"))

def call\_settings(call):

if call.data == "settings#request#new":

msg = bot.send\_message(call.message.chat.id, "Напиши запрос для сохранения.")

bot.register\_next\_step\_handler(msg, save\_name)

elif call.data == "settings#request#del":

msg = bot.send\_message(call.message.chat.id, "Выбери запрос для удаления.", reply\_markup=MyKeyboards.requests(request\_file))

bot.register\_next\_step\_handler(msg, delete\_request)

elif call.data == "settings#request#delall":

msg = bot.send\_message(call.message.chat.id, "Вы уверены? (Y/N)")

bot.register\_next\_step\_handler(msg, delete\_all\_request)

def save\_name(message):

msg = bot.send\_message(message.chat.id, "Напиши название для удобства. Если не хочешь сохранять запрос, напишите -")

bot.register\_next\_step\_handler(msg, save\_request, message.text)

def save\_request(message, request):

save(request, message.text if message.text != '-' else None)

bot.send\_message(message.chat.id, "Сохранено", reply\_markup=MyKeyboards.main())

def delete\_request(message):

if message.text == "Назад":

bot.send\_message(message.chat.id, "Отменено", reply\_markup=MyKeyboards.main())

return

delete(message.text)

bot.send\_message(message.chat.id, "Удалено", reply\_markup=MyKeyboards.main())

def delete\_all\_request(message):

if message.text == "Y":

delete\_all()

bot.send\_message(message.chat.id, "Удалено", reply\_markup=MyKeyboards.main())

def execute(request):

try:

cur = conn.cursor()

cur.execute(request)

rows = cur.fetchall()

cols = [desc[0] for desc in cur.description]

table\_str = list\_to\_str(rows, cols)

return f"Вот результаты вашего запроса:\n\n<pre>{table\_str}</pre>"

except Exception as e:

return f"Произошла ошибка: {e}"

def save(request, name = None):

with open(request\_file, mode='a', newline='', encoding='utf-8') as file:

writer = csv.writer(file)

writer.writerow([request, name if name is not None else request])

def get(name):

with open(request\_file, mode='r', newline='', encoding='utf-8') as file:

reader = csv.reader(file)

for row in reader:

if row[1] == name:

return row[0]

def get\_all():

rows = {}

with open(request\_file, mode='r', newline='', encoding='utf-8') as file:

reader = csv.reader(file)

for row in reader:

rows[row[1]] = row[0]

def delete(name):

with open(request\_file, mode='r', newline='', encoding='utf-8') as file:

reader = csv.reader(file)

rows = []

for row in reader:

if row[1] != name:

rows.append(row)

with open(request\_file, mode='w', newline='', encoding='utf-8') as file:

writer = csv.writer(file)

writer.writerows(rows)

def delete\_all():

with open(request\_file, mode='w', newline='', encoding='utf-8') as file:

writer = csv.writer(file)

**Файл создания тестовой базы данных**

-- Удаление таблиц, если они уже существуют

DROP TABLE IF EXISTS registration\_log;

DROP TABLE IF EXISTS registrations;

DROP TABLE IF EXISTS course\_assignments;

DROP TABLE IF EXISTS professors;

DROP TABLE IF EXISTS students;

DROP TABLE IF EXISTS courses;

-- Создаем таблицу студентов

CREATE TABLE students (

student\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

date\_of\_birth DATE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL

);

-- Создаем таблицу курсов

CREATE TABLE courses (

course\_id SERIAL PRIMARY KEY,

course\_name VARCHAR(100) NOT NULL,

course\_description TEXT,

course\_credits INT NOT NULL

);

-- Создаем таблицу преподавателей

CREATE TABLE professors (

professor\_id SERIAL PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

department VARCHAR(100) NOT NULL

);

-- Создаем таблицу назначений курсов

CREATE TABLE course\_assignments (

assignment\_id SERIAL PRIMARY KEY,

course\_id INT NOT NULL,

professor\_id INT NOT NULL,

assignment\_date DATE NOT NULL,

FOREIGN KEY (course\_id) REFERENCES courses(course\_id),

FOREIGN KEY (professor\_id) REFERENCES professors(professor\_id)

);

-- Создаем таблицу регистраций

CREATE TABLE registrations (

registration\_id SERIAL PRIMARY KEY,

student\_id INT NOT NULL,

course\_id INT NOT NULL,

registration\_date DATE NOT NULL,

FOREIGN KEY (student\_id) REFERENCES students(student\_id),

FOREIGN KEY (course\_id) REFERENCES courses(course\_id)

);

-- Заполнение таблицы студентов

INSERT INTO students (first\_name, last\_name, date\_of\_birth, email) VALUES

('John', 'Doe', '1995-05-20', 'john.doe@example.com'),

('Jane', 'Smith', '1996-08-15', 'jane.smith@example.com'),

('Alice', 'Johnson', '1997-03-22', 'alice.johnson@example.com'),

('Bob', 'Brown', '1994-11-12', 'bob.brown@example.com'),

('Carol', 'Davis', '1998-07-30', 'carol.davis@example.com'),

('David', 'Clark', '1997-05-14', 'david.clark@example.com'),

('Emma', 'Walker', '1996-10-22', 'emma.walker@example.com'),

('Frank', 'Wright', '1995-06-18', 'frank.wright@example.com'),

('Grace', 'Hall', '1997-08-05', 'grace.hall@example.com'),

('Hank', 'Lee', '1996-01-12', 'hank.lee@example.com'),

('Ivy', 'Harris', '1998-09-23', 'ivy.harris@example.com'),

('Jack', 'Young', '1994-12-31', 'jack.young@example.com'),

('Kathy', 'King', '1995-11-11', 'kathy.king@example.com'),

('Liam', 'Hill', '1997-04-03', 'liam.hill@example.com'),

('Mia', 'Green', '1996-05-25', 'mia.green@example.com'),

('Noah', 'Adams', '1998-10-15', 'noah.adams@example.com'),

('Olivia', 'Nelson', '1997-02-10', 'olivia.nelson@example.com'),

('Paul', 'Baker', '1995-03-20', 'paul.baker@example.com'),

('Quinn', 'Carter', '1996-06-17', 'quinn.carter@example.com'),

('Ryan', 'Mitchell', '1998-12-07', 'ryan.mitchell@example.com');

-- Заполнение таблицы курсов

INSERT INTO courses (course\_name, course\_credits, course\_description) VALUES

('Mathematics', 3, 'An introduction to mathematical concepts'),

('History', 4, 'Overview of world history'),

('Physics', 4, 'Fundamentals of Physics'),

('Chemistry', 3, 'Basics of Chemistry'),

('Biology', 3, 'Introduction to Biology'),

('Computer Science', 3, 'Basics of computer science'),

('English Literature', 4, 'Study of English literature'),

('Art History', 3, 'Introduction to art history'),

('Economics', 4, 'Principles of economics'),

('Philosophy', 3, 'Introduction to philosophy'),

('Political Science', 4, 'Basics of political science'),

('Psychology', 3, 'Fundamentals of psychology'),

('Sociology', 3, 'Introduction to sociology'),

('Statistics', 3, 'Basics of statistics'),

('Anthropology', 4, 'Introduction to anthropology'),

('Music Theory', 3, 'Fundamentals of music theory'),

('Geography', 3, 'Basics of geography'),

('Environmental Science', 4, 'Introduction to environmental science'),

('Linguistics', 3, 'Study of language structures'),

('Theatre Arts', 3, 'Basics of theatre arts');

-- Заполнение таблицы преподавателей

INSERT INTO professors (first\_name, last\_name, department) VALUES

('Dr. Emily', 'Clark', 'Mathematics'),

('Dr. Michael', 'Brown', 'History'),

('Dr. Sarah', 'Wilson', 'Physics'),

('Dr. James', 'Taylor', 'Chemistry'),

('Dr. Linda', 'Miller', 'Biology'),

('Dr. William', 'Moore', 'Computer Science'),

('Dr. Elizabeth', 'Jackson', 'English Literature'),

('Dr. Daniel', 'White', 'Art History'),

('Dr. Patricia', 'Harris', 'Economics'),

('Dr. Robert', 'Martin', 'Philosophy'),

('Dr. Barbara', 'Thompson', 'Political Science'),

('Dr. Richard', 'Garcia', 'Psychology'),

('Dr. Mary', 'Martinez', 'Sociology'),

('Dr. Charles', 'Robinson', 'Statistics'),

('Dr. Joseph', 'Walker', 'Anthropology'),

('Dr. Susan', 'Young', 'Music Theory'),

('Dr. Thomas', 'Allen', 'Geography'),

('Dr. Karen', 'King', 'Environmental Science'),

('Dr. Christopher', 'Wright', 'Linguistics'),

('Dr. Jennifer', 'Scott', 'Theatre Arts');

-- Заполнение таблицы назначений курсов

INSERT INTO course\_assignments (course\_id, professor\_id, assignment\_date) VALUES

(1, 1, '2023-01-05'),

(2, 2, '2023-01-10'),

(3, 3, '2023-01-15'),

(4, 4, '2023-01-20'),

(5, 5, '2023-01-25'),

(6, 6, '2023-02-01'),

(7, 7, '2023-02-05'),

(8, 8, '2023-02-10'),

(9, 9, '2023-02-15'),

(10, 10, '2023-02-20'),

(11, 11, '2023-02-25'),

(12, 12, '2023-03-01'),

(13, 13, '2023-03-05'),

(14, 14, '2023-03-10'),

(15, 15, '2023-03-15'),

(16, 16, '2023-03-20'),

(17, 17, '2023-03-25'),

(18, 18, '2023-03-30'),

(19, 19, '2023-04-01'),

(20, 20, '2023-04-05');

-- Заполнение таблицы регистраций

INSERT INTO registrations (student\_id, course\_id, registration\_date) VALUES

(1, 1, '2023-01-10'),

(2, 2, '2023-02-12'),

(3, 3, '2023-03-10'),

(4, 4, '2023-04-15'),

(5, 5, '2023-05-20'),

(6, 6, '2023-06-25'),

(7, 7, '2023-07-30'),

(8, 8, '2023-08-05'),

(9, 9, '2023-09-10'),

(10, 10, '2023-10-15'),

(11, 11, '2023-11-20'),

(12, 12, '2023-12-25'),

(13, 13, '2023-01-30'),

(14, 14, '2023-02-04'),

(15, 15, '2023-03-10'),

(16, 16, '2023-04-15'),

(17, 17, '2023-05-20'),

(18, 18, '2023-06-25'),

(19, 19, '2023-07-30'),

(20, 20, '2023-08-05'),

(1, 2, '2023-09-10'),

(2, 3, '2023-10-15'),

(3, 4, '2023-11-20'),

(4, 5, '2023-12-25'),

(5, 6, '2023-01-30'),

(6, 7, '2023-02-04'),

(7, 8, '2023-03-10'),

(8, 9, '2023-04-15'),

(9, 10, '2023-05-20'),

(10, 11, '2023-06-25'),

(11, 12, '2023-07-30'),

(12, 13, '2023-08-05'),

(13, 14, '2023-09-10'),

(14, 15, '2023-10-15'),

(15, 16, '2023-11-20'),

(16, 17, '2023-12-25'),

(17, 18, '2023-01-30'),

(18, 19, '2023-02-04'),

(19, 20, '2023-03-10'),

(20, 1, '2023-04-15');

-- Создание таблицы для логов регистраций

CREATE TABLE registration\_log (

log\_id SERIAL PRIMARY KEY,

student\_id INT NOT NULL,

course\_id INT NOT NULL,

registration\_date DATE NOT NULL,

log\_date TIMESTAMP NOT NULL

);

-- Создание триггера для логирования регистраций

CREATE OR REPLACE FUNCTION log\_registration() RETURNS TRIGGER AS $$

BEGIN

INSERT INTO registration\_log (student\_id, course\_id, registration\_date, log\_date)

VALUES (NEW.student\_id, NEW.course\_id, NEW.registration\_date, CURRENT\_TIMESTAMP);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

-- Привязка триггера к таблице регистраций

CREATE TRIGGER after\_registration

AFTER INSERT ON registrations

FOR EACH ROW

EXECUTE FUNCTION log\_registration();