МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

МЕТОДЫ ОБЕСПЕЧЕНИЯ БЕЗОПАСНОСТИ БАЗЫ ДАННЫХ POSTGRESQL

КУРСОВАЯ РАБОТА

студента 3 курса 351 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Быковой Марии Дмитриевны

Научный руководитель
зав.каф.техн.пр.,
к. ф.-м. н _____ И. А. Батраева

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Системы баз данных играют ключевую роль в хранении и управлении огромными объемами информации в различных сферах деятельности, от бизнеса и государственных учреждений до личных данных пользователей. Однако, с увеличением количества данных и повышением их важности становится все более актуальным обеспечение безопасности этих систем.

Целью данной работы является разработка базы данных в PostgreSQL, настройка ее защищенности и реализация оконного приложения для взаимодействия с базой с помощью Python.

В ходе написания работы должны быть решены следующие задачи:

1. Изучить основы безопасности баз данных.
2. Проанализировать угрозы информационной безопасности.
3. Изучить различные способы организации безопасности базы данных.
4. Рассмотреть различные виды SQL-инъекций и способы защиты от них.
5. Создать оконное приложение на языке Python для взаимодействия с базой.

1 Основы безопасности баз данных

1.1 Понятие защищенной базы данных

В основе многих настольных приложений и практически всех интернет-сервисов лежит некоторая база данных (БД), которая хранит данные о пользователях и сопутствующие данные предметной области (например, системы управления кадрами предприятия, системы бухгалтерского учета, интернет-магазины и т.д.). Очевидно, что разработчикам и администраторам этих систем необходимо обеспечить бесперебойное функционирование этих приложений и сервисов, а также сохранность данных и доступ к данным исключтельно согласно принятой политики безопасности. При несоблюдении этих очевидных требований возможна утечка персональных данных (например, данные клиентов финансовой организации, медицинская информация о пациентах клиники и т.д.), утечка данных, составляющих коммерческую тайну (например, клиентская база, спецификация оборудования и т.д.), значительное замедление либо полная неработоспособность сервиса или приложения (например, невозможность запустить приложение из-за поломки жесткого диска на сервере БД, невозможность выполнить перевод в интернет-банке из-за недоступности сервера т.д.). В любом случае подробного рода события наносят финансовый ущерб, ущерб деловой репутации организации, а также могут привести к неблагоприятным последствиям для клиентов компании.

Информационная система, частью которой является система управления базами данных (СУБД), должна обеспечивать конфиденциальность, целостность и доступность данных. Более того, должны выполняться все указанные аспекты информационной безопасности, иначе при невыполнении хотя бы одного из них безопасную систему построить не удастся.

Безопасность данных — это состояние защищенности, при котором обеспечиваются конфиденциальность, доступность и целостность данных. Конфиденциальность отвечает за обеспечение доступа к данным, только санкционированным пользователями; целостность исключает несанкционированное изменение структуры и содержания данных; доступность позволяет обеспечить доступ к данным, санкционированным пользователями, по их первому требованию.

Конфиденциальность, целостность и доступность составляют минимальный набор требований к безопасности данных. Эти требования могут быть дополнены критериями защищенности, например, аутентичностью (т.е. подлин-

ностью данных), апеллируемостью (т.е. подтверждением авторства), достоверностью и т.д [1].

Независимыми от данных можно назвать следующие требования к безопасной системе БД:

— функционирование в доверенной среде;

  Под доверенной понимается информационная среда, интегрирующая совокупность защитных механизмов, которые обеспечивают обработку информации без нарушения политики безопасности. В данном случае СУБД должна функционировать в доверенной информационной системе с соответствующими методами обмена данными

— организация физической безопасности файлов данных;

— организация безопасной и актуальной настройки СУБД (общие вопросы обеспечения безопасности, своевременная установка обновлений, отключение неиспользуемых модулей или применение эффективной политики паролей).

  Следующие требования можно назвать зависимыми от данных:

— безопасность пользовательского слоя ПО (построение безопасных интерфейсов и вызовов (в том числе с учетом интерфейса СУБД и механизма до- ступа к данным));

— безопасная организация данных и манипулирование ими.

1.2 Угрозы информационной безопасности

Угроза информационной безопасности информационной системы — это возможность воздействия на информацию, обрабатываемую в системе, приводящего к искажению, уничтожению, копированию, блокированию доступа к информации, а также возможность воздействия на компоненты информационной системы, приводящего к утрате, уничтожению или сбою функционирования носителя информации или средства управления программно-аппаратным комплексом системы.

Комплексная система обеспечения информационной безопасности должна строиться с учетом средств и методов, характерных для четырех уровней информационной системы:

1. уровня прикладного программного обеспечения, отвечающего за взаимодействие с пользователем;

2. уровня системы управления базами данных, обеспечивающего хранение и обработку данных информационной системы;
3. уровня операционной системы, отвечающего за функционирование СУБД и иного прикладного программного обеспечения;
4. уровня среды доставки, отвечающего за взаимодействие информационных серверов и потребителей информации.

Система защиты должна эффективно функционировать на всех этих уровнях.

Существуют различные варианты классификации угроз информационной системе: по природе возникновения, по источнику угроз, по способу доступа к защищаемому ресурсу, по степени воздействия на систему, по расположению источника и т.д. В контексте защиты базы данных рассматривается вариант классификации по способу осуществления угроз.

Выделяются явные угрозы:

— некорректная реализация механизма защиты;
— некорректная настройка механизма защиты;
— неполнота покрытия каналов доступа к информации сред- ствами защиты.

А также скрытые угрозы:

— нерегламентированные действия пользователя;
— ошибки и закладки в программном обеспечении.

Для защиты СУБД от рассмотренных угроз необходимо применять комплекс мер, не всегда имеющих прямое отношение к базе данных, например, обеспечить сетевую защиту, защиту от вирусов или обеспечить безопасность операционной системы в целом [2]. Однако многие вопросы обеспечения безопасности решаются именно путем администрирования СУБД, в частности:

— аутентификация и авторизация пользователя;
— криптографическая защита БД;
— резервное копирование данных;
— репликация и балансировка нагрузки;
— аудит событий безопасности БД;
— модернизация системного и прикладного ПО;
— доступ к данным только с использованием представлений и хранимых процедур.

1.3 Основные принципы обеспечения безопасности

Опыт создания систем защиты позволяет выделить следующие основные принципы построения систем компьютерной безопасности, которые необходимо учитывать при их проектировании и разработке:

1. Системность — необходимость учета всех взаимосвязанных, взаимодействующих и изменяющихся во времени элементов, условий и факторов, значимых для понимания и решения проблемы обеспечения безопасности автоматизированной системы.

   Система защиты должна строиться с учетом не только всех известных каналов проникновения и получения несанкционированного доступа к информации, но и с учетом возможности появления принципиально новых путей реализации угроз безопасности.

2. Комплексность — согласованное применение разнородных средств при построении целостной системы защиты, перекрывающей все существенные каналы реализации угроз и не содержащей слабых мест на стыках отдельных ее компонентов.

   Комплексная защита информационной системы должна обеспечиваться физическими средствами, организационными и правовыми мерами и использовать средства защиты, реализованные как на уровне операционных систем, так и на прикладном уровне с учетом особенностей предметной области.

3. Непрерывность защиты.

   Разработка системы защиты должна вестись параллельно с разработкой самой защищаемой системы. Это позволит учесть требования безопасности при проектировании архитектуры и, в конечном счете, позволит создать более эффективные (как по затратам ресурсов, так и по стойкости) защищенные системы.

4. Разумная достаточность — важно правильно выбрать тот достаточный уровень защиты, при котором затраты, риск и размер возможного ущерба были бы приемлемыми (задача анализа риска).

5. Гибкость управления и применения — для обеспечения возможности варьирования уровнем защищенности, средства защиты должны обладать определенной гибкостью.

6. Открытость алгоритмов и механизмов защиты — защита не должна обес-

печиваться только за счет секретности структурной организации и алгоритмов функционирования ее подсистем.

Знание алгоритмов работы системы защиты не должно давать возможности ее преодоления (даже автору). Однако это вовсе не означает, что информация о конкретной системе защиты должна быть общедоступна.

7. Простота применения защитных мер и средств — механизмы защиты должны быть интуитивно понятны и просты в использовании.

1.4 Понятие SQL-инъекции

SQL-инъекции — это один из очень распространённых способов взлома сайтов и веб-приложений, работающих с реляционными базами данных. Этот способ основан на внедрении в выполняемый приложением запрос к базе данных произвольного SQL-кода, переданного злоумышленником. SQL-инъекции являются одной из разновидностей атак типа «инъекция кода» [3].

Типы SQL-инъекций:

— на основе ошибки — используют сообщения об ошибке, выбрасываемой сервером базы данных (сообщения могут дать полезную информацию о структуре базы);

— с использованием UNION — задействует оператор SQL UNION для объединения результатов двух запросов SELECT в единую аблицу (можно получить информацию из других таблиц добавлением результатов к исходному запросу);

— «слепая» инъекция — приложение подвержено SQL-инъекции, но результаты запроса не возвращаются в HTT-ответе; тогда в базу данных выполняется запрос на любой из операторов true-false, и отслеживаются изменения для условий true и false;

— с использованием особенностей сервера — редкий тип инъекции, зависит от конкретных характеристик сервера базы данных; использует способность сервера базы данных выполнять веб-запросы типа HTTP, DNS и ftp для отправки данных злоумышленнику.

SQL-инъекция, в зависимости от типа уязвимости, может дать возможность атакующему выполнить произвольный запрос к базе данных. То есть атакующий сможет прочитать содержимое любых таблиц, удалить, изменить или добавить данные, а также есть вероятность получения возможности работы с локальными файлами и выполнения произвольных команд на атакуемом

сервере.

Последствия SQL-инъекций:

- кража данных;
- модификация данных;
- удаление данных;
- полный взлом системы.

Атаки типа внедрения SQL становятся возможными из-за некорректной обработки приложением входных данных. Разработчики сайтов и приложений, работающих с реляционными базами данных, должны знать о таких уязвимостях и принимать меры противодействия. С точки зрения безопасности сайта или веб-приложения, наиболее правильной является фильтрация всех полученных данных (приложение в принципе не должно принимать в обработку те параметры, которые непонятно зачем были переданы) и продуманная обработка тех параметров, которые используются для построения запроса к базе данных.

Для предотвращения SQL-инъекций основными направлениями обработки входных данных должны являться:

1. экранирование специальных символов в полученных строковых параметрах;
2. приведение полученных данных к типу, ожидаемому приложением;
3. нормализация и очистка полученных параметров регулярными выражениями;
4. валидация полученных параметров по справочникам;
5. усечение полученных параметров;
6. использование параметризованных запросов;
7. правильное использование клиентских библиотек и возможностей ORM;
8. фильтрация по ключевым словам.

2 Разработка базы данных и оконного приложения

2.1 Предметная область

Разработаем базу данных для библиотечной системы. Данный сервис предлагает следующие возможности:

— учет поступления книг;

— поиск книг по критериям (автор, год написания, издательство, год издания);

— учет выданных книг;

— работа с должниками.

Схема разрабатываемой базы данных представлена на рисунке 1 .

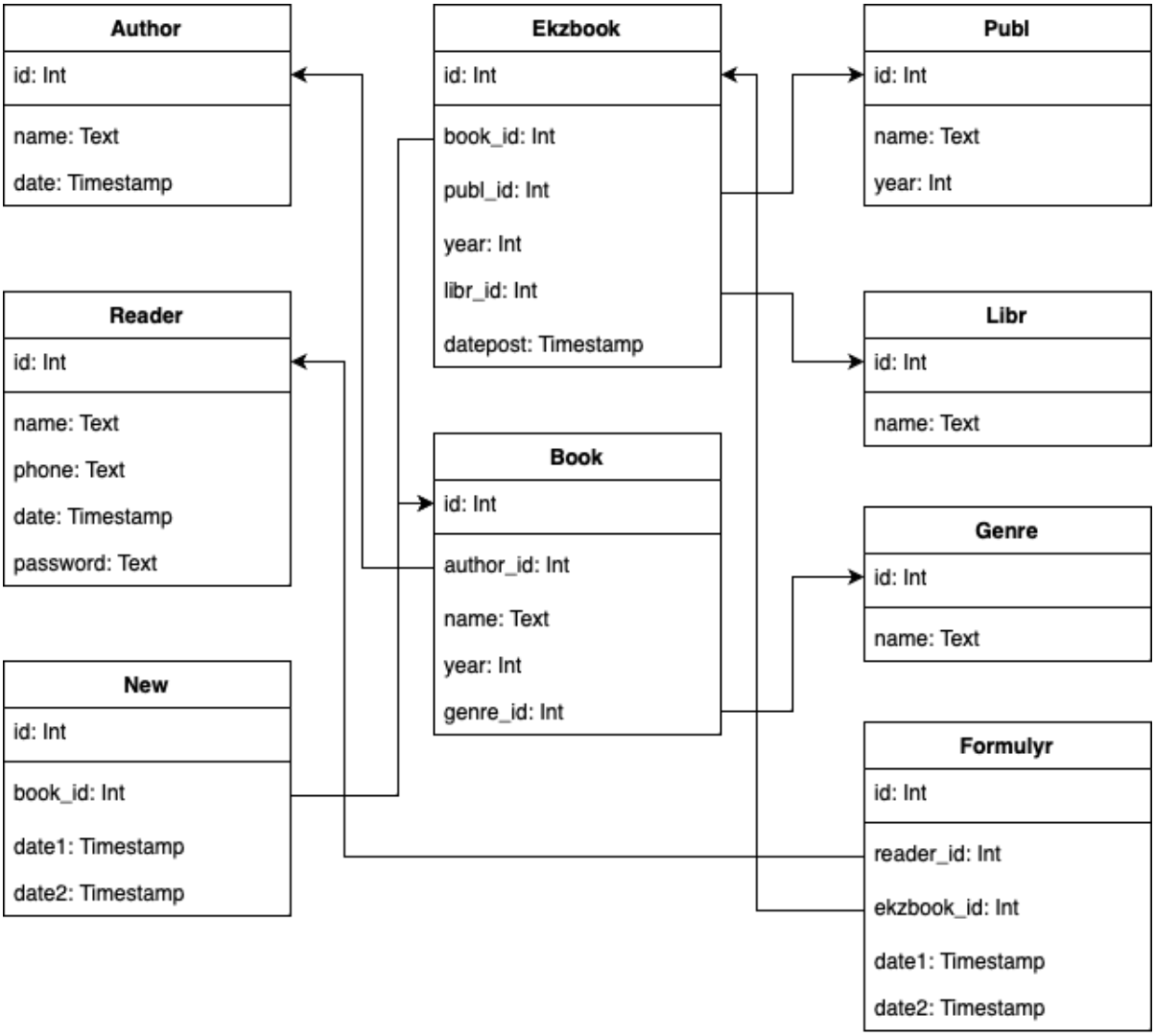


Рисунок 1 – Схема базы данных «Библиотечная система»

Система предполагает работу двух типов ролей: Читатель и Администратор. Для Читателя доступны следующие функции:

— поиск книги по критериям;

— просмотр «новинок»;
— чтение книги (добавление книги в формуляр Читателя);
— возврат книги (удаление книги из формуляра Читателя).

Для Администратора доступны следующие функции:

— добавление книг, авторов и издательств;
— добавление и редактирование «новинок»;
— выявление должников через карточку книги.

2.2 Создание базы данных

Для создания сущности «Автор» сделаем следующий SQL-запрос [4]:

```
CREATE TABLE author(
  id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  name text,
  date DATETIME)
```

Данная сущность имеет следующие атрибуты:

— идентификатор автора id: INT, NOT NULL;
— имя автора name: TEXT;
— дата рождения date: DATETIME.

Для создания сущности «Читатель» сделаем следующий SQL-запрос:

```
CREATE TABLE reader(
  id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
  name text,
  phone text,
  date DATETIME6
  password text)
```

Данная сущность имеет следующие атрибуты:

— идентификатор читателя id: INT, NOT NULL;
— имя читателя name: TEXT;
— номер телефона phone: TEXT;
— дату рождения date: DATETIME;
— пароль password: TEXT.

Аналогично создадим сущности «Книга», «Экземпляр книги», «Формуляр читателя», «Жанр», «Библиотека», «Новинки», «Издательство». Ознакомиться с полным кодом создания сущностей можно в приложении А.

2.3 Организация безопасности базы с помощью ролевой модели доступа

Для обеспечения безопасности базы данных есть возможность назначать роли для пользователей. PostgreSQL имеет комплексную систему прав пользователя, основанную на концепции ролей. В современных версиях PostgreSQL (8.1 и новее) понятие «роль» является синонимом понятия «пользователь», поэтому любое имя учетной записи базы данных, которое используется для подключения, на самом деле является ролью с атрибутом LOGIN, который позволяет ей подключаться к базе данных.

Помимо возможности входа в систему, роли могут иметь иные атрибуты, позволяющие им обходить все проверки прав доступа (SUPERUSER), создавать базы данных (CREATEDB), создавать другие роли (CREATEROLE) и т. д. Помимо атрибутов, ролям могут быть предоставлены права доступа, которые можно разделить на две категории: членство в других ролях и привилегии на объекты базы данных.

Привилегиями доступа называют разрешение на использование определенной услуги управления данными для доступа к объекту данных (например, чтение, запись или исполнение), предоставляемое идентифицированному пользователю. Ролью в свою очередь называется именованная совокупность привилегий, которые могут быть предоставлены пользователям или другим ролям.

Управление доступом на основе ролей — развитие политики избирательного управления доступом, при этом права доступа (привилегии) субъектов системы на объекты группируются с учётом специфики их применения, образуя роли.

Для создания роли «Администратор» сделаем следующий SQL-запрос [5]:

```
create role "admin" with login superuser
password 'adminserver2024'
```

В данном случае admin может обходить все проверки прав доступа, то есть фактически «обладает полной властью».

Для создания роли «Читатель» сделаем следующий SQL-запрос:

```
Create role "reader" password 'readerserver2024'
```

Сейчас reader не обладает никакими привилегиями и не может свободно взаимодействовать с базой.

Выдадим привилегии на некоторые функции:

```
grant select on book to reader
grant select on new to reader
grant select, insert, delete on formulyr to reader
```

Теперь reader может читать данные с таблицы «book», «new» и «formulyr», а также добавлять и удалять строки с таблицы «formulyr». При попытке любого другого взаимодействия с базой будет выведена следующая ошибка:

```
no privileges on tablespace
```

Другие наиболее часто используемые привилегии объектов базы данных — INSERT, UPDATE, DELETE и TRUNCATE аналогичны соответствующим SQL-выражениями. Также можно назначить права для подключения к определенным базам данных, создания новых схем или объектов в схеме, выполнения функции и т. д.

По сути, управление доступом состоит из следующих стадий [6]:

— для каждой роли создается набор полномочий, которые предоставляют пользователю определенные права;
— каждому пользователю предоставляется список ролей;
— после авторизации пользователя создается сессия.

Далее, в зависимости от доступных ролей в сессии, проверяются права доступа к объектам.

Использование ролевой модели во многом упрощает разграничение прав доступа и по своему определению наилучшим образом подходит для реализации принципа наименьших привилегий, а также четкого и прозрачного разграничения полномочий.

2.4 Реализация оконного приложения для Читателя и для Администратора

Создадим теперь fronted-оболочку сервиса — оконные приложения.

Для реализации оконного приложения подключим библиотеку Tkinter:

```
import tkinter
```

Пусть наше окно приложения Читателя содержит три элемента Label (надписи), три элемента Entry (текстовые поля ввода) и пять элементов Button (кнопка) [7]. Вид окна представлен на рисунке 2

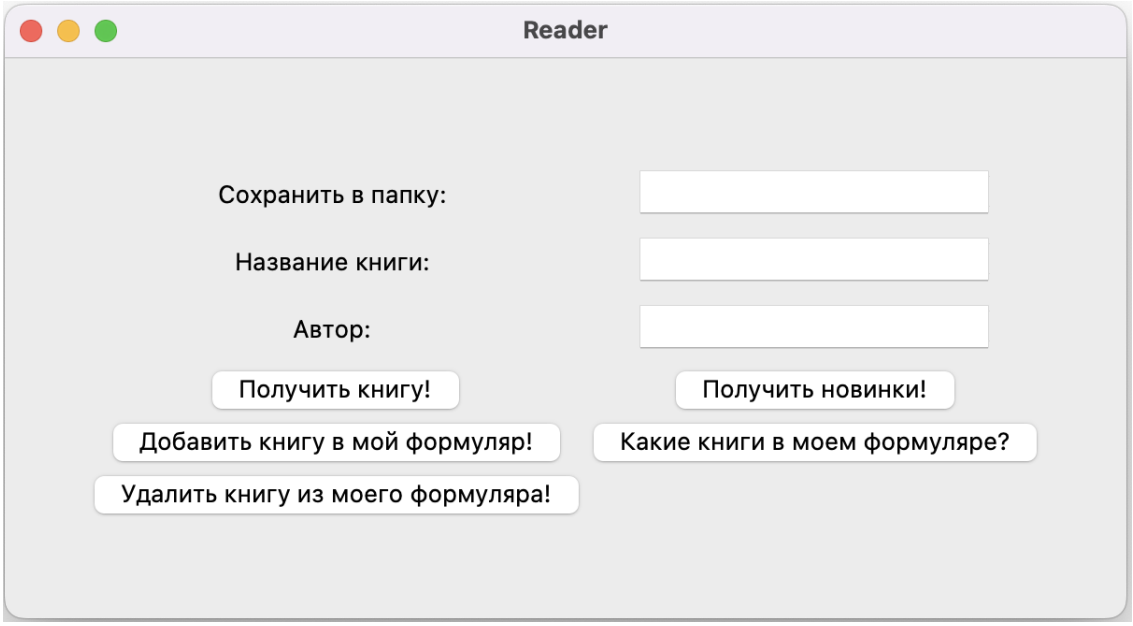


Рисунок 2 – Окно приложения Читателя

Механизм работы приложения следующий:

1. Пользователь вводит директорию папки, куда необходимо сохранить полученные данные, в поле «Сохранить в папку».
2. Если пользователь хочет получить книгу по ее названию, необходимо ввести название книги в поле «Название книги» и нажать на кнопку «Получить книгу». После этого в указанной папке будет сохранен csv-файл с данными о данной книге.
3. Если пользователь хочет получить все книги определенного автора, необходимо в поле «Автор» указать фамилию и инициалы автора и нажать на кнопку «Получить книгу». После этого в указанной папке будет сохранен csv-файл с данными о книгах данного автора.
4. Если пользователь хочет посмотреть, какие книги в данный момент записаны в его формуляре, необходимо нажать на кнопку «Какие книги в моем формуляре?». После этого в соответствующей директории появится файл с данными.
5. Если пользователь хочет добавить определенную книгу в свой формуляр, необходимо ввести название книги в поле «Название книги» и нажать на кнопку «Добавить книгу в мой формуляр!».
6. Если пользователь хочет удалить определенную книгу из своего формуляра, необходимо ввести название книги в поле «Название книги» и нажать на кнопку «Удалить книгу из моего формуляра!».

Ознакомиться с полным кодом оконного приложения Читателя можно в приложении Б.

Приложение Читателя реализует именно такой функционал как следствие ограниченности привилегий пользователя — позволяем реализовывать запросы только к тем таблицам, к которым открыт доступ. Но при этом пользователь все равно может просвести SQL-инъекцию (инъекция типа UNION).

Перейдем к оконному приложению Администратора. Пусть оно содержит два элемента Label (надписи), два элемента Entry (текстовые поля ввода) и один элемент Button (кнопка) [8]. Вид окна представлен на рисунке 3

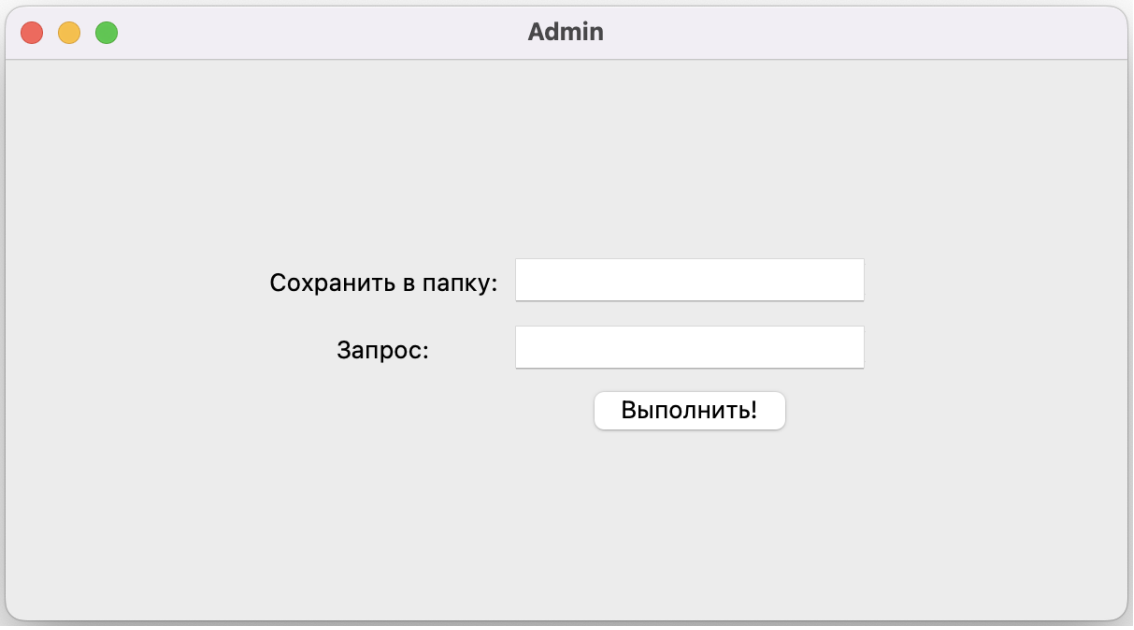


Рисунок 3 – Окно приложения Администратора

Механизм работы приложения следующий:

1. Администратор вводит директорию папки, куда необходимо сохранить полученные данные, в поле «Сохранить в папку».
2. Администратор может вводить любой SQL-запрос в поле «Запрос»; после этого необходимо нажать на кнопку «Выполнить!», и файл с данными будет сохранен в заданной директории.

Ознакомиться с полным кодом оконного приложения Администратора можно в приложении В.

Так как администратор обладает ролью SUPERUSER, он может сделать любой запрос к базе, что может быть потенциальной угрозой для базы. Рассмотрим некторые способы защиты.

2.5 Организация безопасности базы на уровне строк

Одной из наиболее продвинутых функций системы привилегий PostgreSQL является безопасность на уровне строк, которая позволяет предоставлять привилегии подмножеству строк в таблице. Сюда входят как строки, которые могут быть запрошены с помощью оператора SELECT, так и строки, которые могут быть результатами операторов INSERT, UPDATE и DELETE [9].

Чтобы начать использовать безопасность на уровне строк, необходимо включить ее для таблицы и определить политику, которая будет контролировать доступ на уровне строк. Предположим, что хотим разрешить пользователям обновлять новинки книг. Для реализации этого для начала включим RLS в таблице:

```
ALTER TABLE new ENABLE ROW LEVEL SECURITY;
```

Без какой-либо определенной политики PostgreSQL по умолчанию использует политику «запретить». Это означает, что никакая роль (кроме владельца таблицы, который обычно является ролью, создавшей таблицу) не имеет к ней доступа.

Политика безопасности строк — это логическое выражение, которое PostgreSQL будет применять для каждой строки, которая должна быть возвращена или обновлена. Строки, возвращаемые оператором SELECT, проверяются на соответствие выражению, указанному в разделе USING, в то время как строки, обновленные операторами INSERT, UPDATE или DELETE, проверяются на соответствие с WITH CHECK выражением.

Определим настройки, которые позволяют пользователям видеть все новинки и обновлять их:

```
CREATE POLICY select_all_new
ON "new" FOR SELECT
USING (true);

CREATE POLICY update_all_new
ON "new" FOR UPDATE
USING (true);
```

Необходимо сделать важное замечание: только владелец таблицы может создавать или обновлять для нее политику безопасности строк.

2.6 Защита базы от SQL-инъекций

SQL-инъекция (англ. SQL injection) — распространённая атака на веб-сайты и приложения, работающие с базой данных. Атака возможна в том случае, когда входные данные, получаемые от пользователя, некорректно или недостаточно фильтруются перед использованием в запросах к БД [10]. Это позволяет внедрять во входные данные произвольный SQL-код, меняющий логику работы запроса.

В зависимости от типа используемой СУБД и особенностей уязвимого приложения, SQL-инъекция может дать возможность атакующему выполнить произвольный запрос к БД, получить возможность чтения и записи локальных файлов или даже выполнения произвольных команд на атакуемом сервере.

Наиболее общими способами предотвращения SQL-инъекций являются фильтрация, экранирование и проверка всех входных данных.

Атака может быть реализована против любой СУБД, поддерживающей SQL: PostgreSQL, MySQL, Oracle и т.д.

Рассмотрим уязвимости, приводящих к возможности реализации SQ-инъекций.

2.6.1 SQL-инъекции типа UNION

Язык SQL позволяет объединять результаты нескольких запросов при помощи оператора UNION. Это позволяет злоумышленнику получить несанкционированный доступ к данным в таблицах, не используемых в исходном запросе.

Данный код предназначен для отображения книги с заданным id из таблицы book.

```
select book.author_id, book.year
 from book where book.id= $id
```

Если передать в качестве id строку «-1 UNION SELECT new.id, new.book_id FROM new», то запрос не вернёт ни одной строки из таблицы «book», поскольку строки с идентификатором «-1» заведомо не существует. При этом посредством оператора UNION будут выбраны все строки из таблицы «new». Для успешного внедрения оператора UNION количество столбцов в выборках должно совпадать. Если количество столбцов в первой (исходной) выборке больше, то вторую необходимо дополнить соответствующим количеством констант (например, NULL).

Результат выполнения данной SQL-инъекции Автором представлен на рисунке 4 .

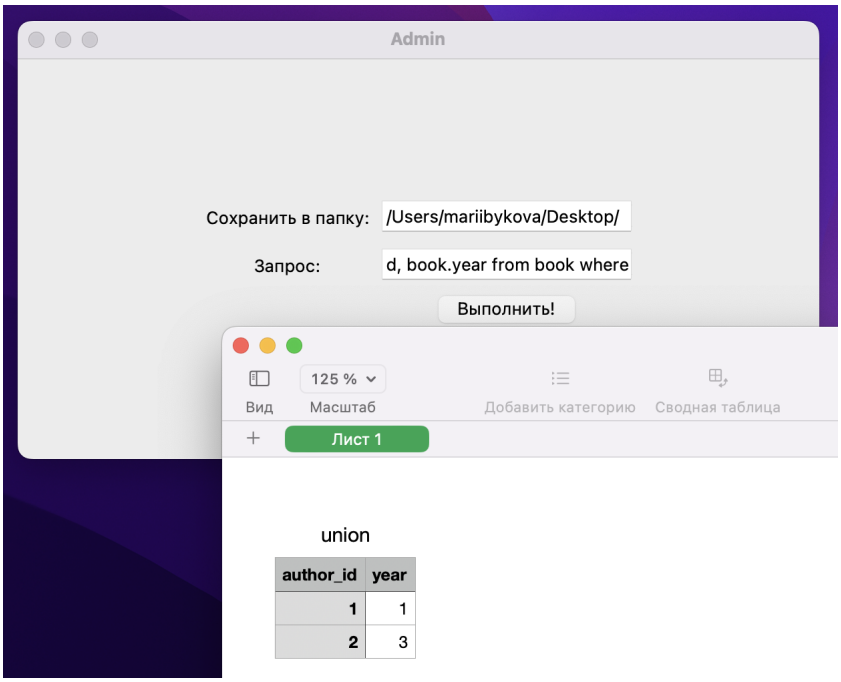


Рисунок 4 – SQL-инъекция типа UNION

2.6.2 Слепая SQL-инъекция

SQL-инъекция называется слепой в том случае, когда результат выполнения запроса недоступен злоумышленнику. При этом уязвимый объект по-разному реагирует на различные логические выражения, подставляемые в уязвимый параметр [11]. Таким образом, злоумышленник может подобрать значения некоторых параметров (версия СУБД, текущее имя и права пользователя и т. д.), подставляя в запрос соответствующие логические выражения.

Рассмотрим следующий запрос:

```
SELECT * FROM book WHERE book.id = $id
```

Задача данного запроса — вывести автора с указанным id.

Если в качестве id передать строку «1 AND 1 = 1», то условие запроса не изменится, поскольку выражение «1 = 1» всегда истинно. Если автор с id, равным 1, существует, то в ответ будет получено с его описанием. Если затем в качестве id передать строку «1 AND 1 = 2» с заведомо ложным условием, то будет получено сообщение о том, что запрошенный автор не существует.

Проведем такую SQL-инъекцию с через Читателя. В поле имени книги введем «’Poltava’ AND 1 = 2». Такая книга в таблице есть, но из-за дополнения

«AND 1 = 2» Читатель получит пустой файл. Результат выволнения представлен на рисунке 5 .

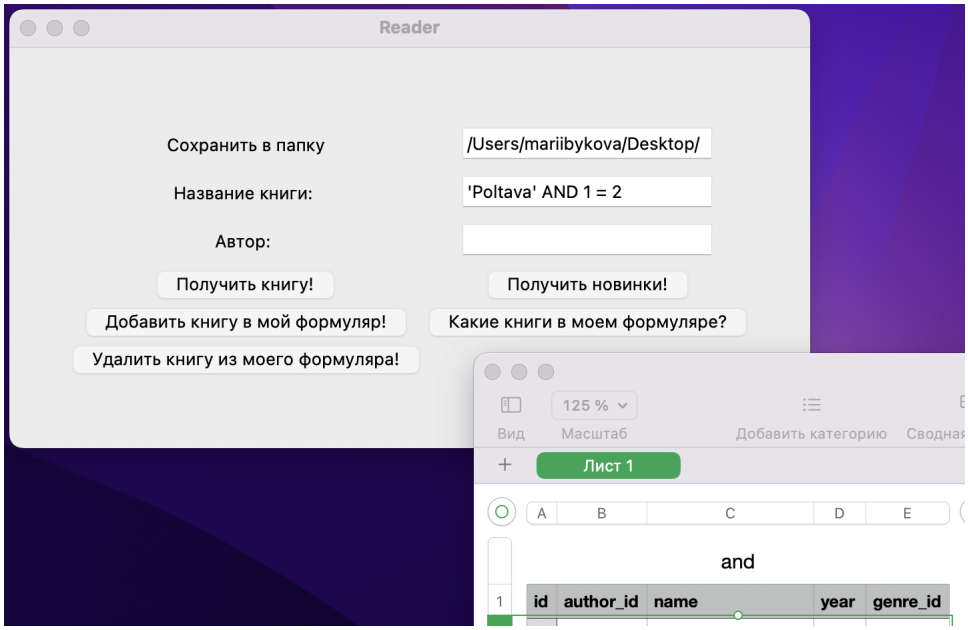


Рисунок 5 – Слепая SQL-инъекция

2.6.3 SQL-инъекции, использующие конкатенацию строк

Иногда злоумышленник может провести атаку, но не может видеть более одной колонки результатов. В таком случае можно объединить несколько строк в одну при помощи конкатенации:

```
SELECT CONCAT(name, ':', password) FROM reader
```

В некоторых случаях запрос, подверженный SQL-инъекции, имеет структуру, усложняющую или препятствующую внедрению операторов SQL. Данный код выбирает строки с указанным идентификатором «author_id», сортирует их по году и выводит 20 первых записей.

```
SELECT book.id, book.name FROM book
WHERE book.author_id = $author_id
ORDER BY year DESC LIMIT 20
```

Простая подстановка оператора UNION вместо «author_id» приведёт к ошибке из-за оставшейся части запроса «ORDER BY date DESC LIMIT 20». В этом случае часть запроса необходимо экранировать при помощи символов комментария. Часть строки, отделённая этими символами, будет проигнорирована и запрос успешно выполнится.

При этом важно помнить, что все скобки и кавычки должны быть закрыты, чтобы не вызывать ошибок при выполнении запроса. Если часть запроса экранируется, то соответствующие символы нужно передавать в явном виде.

2.6.4 SQL-инъекции, использующие дополнительные символы

Для разделения команд в языке SQL используется символ «;» (точка с запятой). Внедряя этот символ в запрос, злоумышленник получает возможность выполнить несколько команд в одном запросе. Это позволяет выполнять операции, отличные от применяемых в исходном запросе, например, вставлять, изменять или удалять строки.

```
SELECT * FROM new WHERE new.id = $id
```

Передав в качестве id строку «-1; INSERT INTO reader(id,name, password) VALUES (4,'foo', 'bar')», злоумышленник может несанкционированно вставить новую строку в таблицу «reader».

Если теперь вывести всех пользователей, то появится и эта дополнительная строка. Результат выволнения данной SQL-инъекции Автором представлен на рисунке 6 .

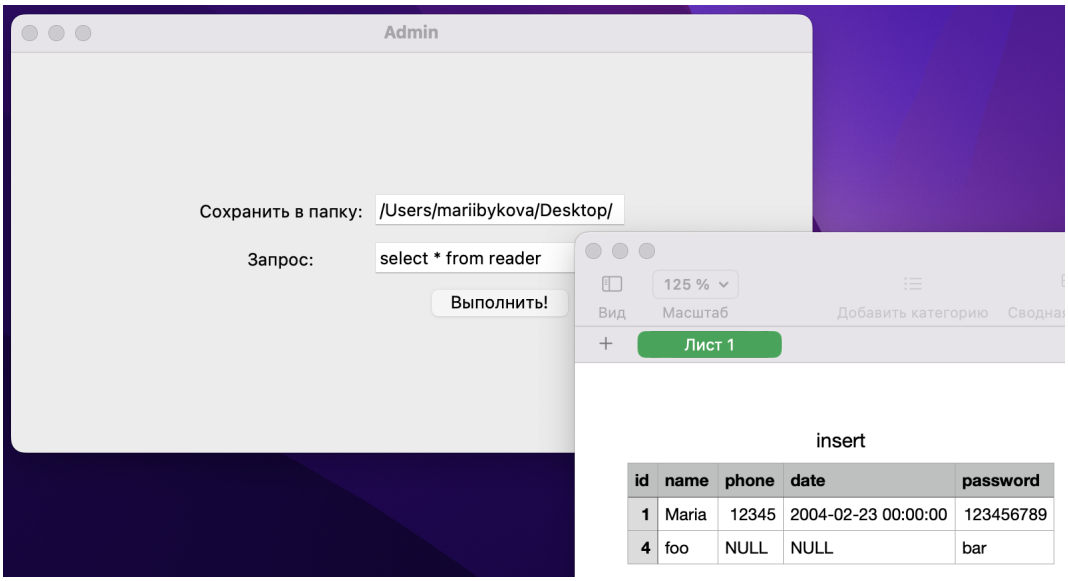


Рисунок 6 – SQL-инъекция с дополнительными символами

2.6.5 Методы защиты от SQL-инъекций

Большинство SQL-инъекций встречаются в числовом и строковом параметрах в запросах, использующих оператор SELECT. Поэтому для обеспечения

защиты от них в запросе нужно проверять все вводимые данные на соотвествие их типам.

Функция is_numeric(n) используется для проверки переменной на числовое значение, которая вернёт true, если параметр n — число, а в противном случае — false. Ее можно использовать при проверке значения id (защита от SQL-инъекции типа UNION и «слепой» инъекции). В Python данная функия реализуется как «.isdigit()».

Компрометации через SQL-конструкции происходят и по причине нахождения в строках небезопасных кавычек и других специальных символов. Для предотвращения такой угрозы необходимо использовать функцию addslashes(str), которая возвращает строку str с добавленным обратным слешем перед каждым специальным символом. Данный процесс называется экранированием.

Для реализации экранирования в Python используется знак «%», который ставится до и после параметров, добавляемых в SQL-запрос. Сделаем экранирование строки в полях ввода названия книги и имени автора для Пользователя. Ввод названия книги происходит в текстовое поле names, ввод имени автора — в поле authors. Поэтому во всех функциях, использующих данное поле, получение данных организуем следующим образом:

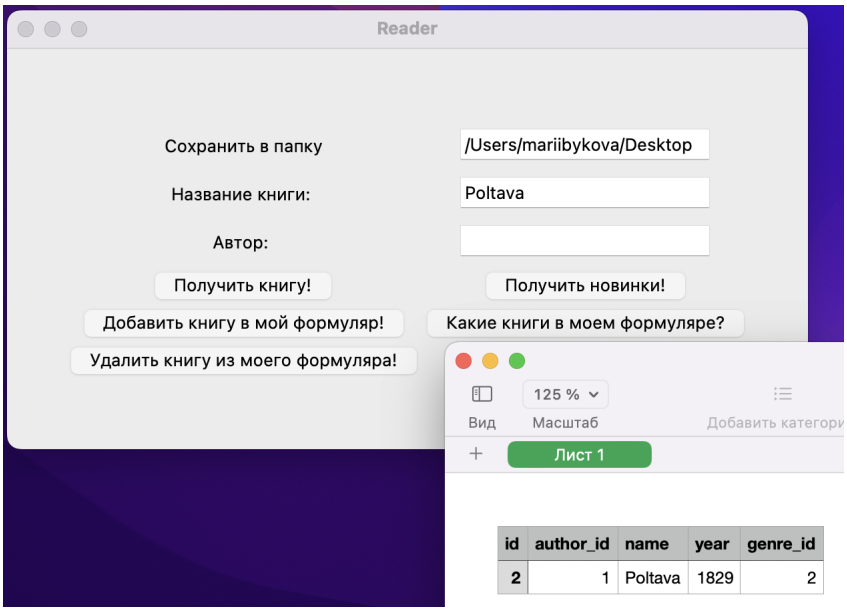
```
name = '%' + names.get() + '%'
author = '%' + authors.get() + '%'
```

Результат выполнения экранированного запроса Читателем представлен на рисунке 7 .

Для экранирования запроса у Администратора весь запрос поместим в «%»:

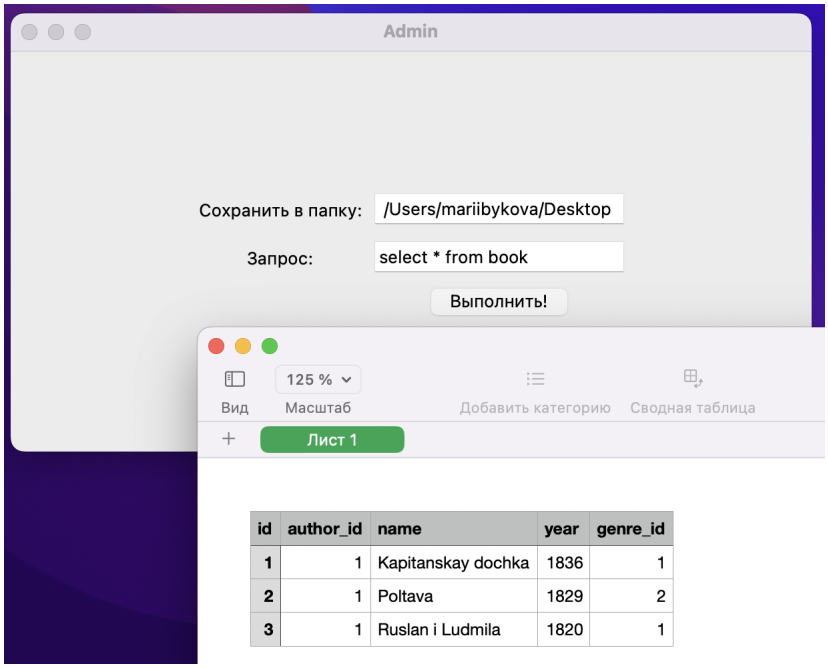
```
sqlite_select_query = "%" + take_tf.get() + "%"
```

Результат выполнения экранированного запроса Администратором представлен на рисунке 8 .



id	author_id	name	year	genre_id
2	1	Poltava	1829	2

Рисунок 7 – SQL-инъекция типа UNION



id	author_id	name	year	genre_id
1	1	Kapitanskay dochka	1836	1
2	1	Poltava	1829	2
3	1	Ruslan i Ludmila	1820	1

Рисунок 8 – SQL-инъекция типа UNION

ЗАКЛЮЧЕНИЕ

В ходе данной работы:

1. Были изучены основы безопасности баз данных.
2. Проанализированы угрозы информационной безопасности.
3. Были изучены празличные способы организация безопасности базы данных.
4. Были рассмотрены некоторые примеры SQL-инъекций и способы защиты от них.
5. Было создано оконное приложение Читателя и Администратора для взаимодействия с базой.

Таким образом, все поставленные в рамках работы задачи выполнены.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *Агафонов, А.* БЕЗОПАСНОСТЬ СИСТЕМ БАЗ ДАННЫХ / А. Агафонов. — Самара: Издательство Самарского университета, 2023. — С. 272.

2 *Clarke, J.* SQL Injection Attacks and Defense / J. Clarke. — Waltham: Syngress, 2012. — С. 576.

3 SQL injection [Электронный ресурс]. — URL: https://learn.microsoft.com/en-us/sql/relational-databases/security/sql-injection?view=sql-server-ver16 (Дата обращения 15.12.2023). Загл. с экр. Яз. англ.

4 PostgreSQL: Documentation: 16: CREATE DATABASE [Электронный ресурс]. — URL: https://www.postgresql.org/docs/current/sql-createdatabase.html (Дата обращения 22.04.2024). Загл. с экр. Яз. англ.

5 PostgreSQL: Documentation: 16: CREATE ROLE [Электронный ресурс]. — URL: https://www.postgresql.org/docs/current/sql-createrole.html (Дата обращения 25.05.2024). Загл. с экр. Яз. англ.

6 Информационная безопасность баз данных [Электронный ресурс]. — URL: https://searchinform.ru/informatsionnaya-bezopasnost/osnovy-ib/informatsionnaya-bezopasnost-v-otraslyakh/informatsionnaya-bezopasnost-baz-dannykh/ (Дата обращения 21.05.2024). Загл. с экр. Яз. рус.

7 Графический интерфейс на Tkinter [Электронный ресурс]. — URL: https://proglib.io/p/tkinter-2023-05-02?ysclid=lrthmh8j8k298075821 (Дата обращения 03.12.2023). Загл. с экр. Яз. англ.

8 Python Tkinter Image + Examples [Электронный ресурс]. — URL: https://pythonguides.com/python-tkinter-image/ (Дата обращения 14.01.2024). Загл. с экр. Яз. рус.

9 Безопасность на уровне строк [Электронный ресурс]. — URL: https://learn.microsoft.com/ru-ru/sql/relational-databases/security/row-level-security?view=sql-server-ver16 (Дата обращения 07.11.2023). Загл. с экр. Яз. рус.

10 Что такое SQL-инъекция? [Электронный ресурс]. — URL: https://www.kaspersky.ru/resource-center/definitions/sql-injection (Дата обращения 17.02.2024). Загл. с экр. Яз. рус.

11 Blind SQL Injection [Электронный ресурс]. — URL: https://www.stationx.net/blind-sql-injection/ (Дата обращения 10.05.2024). Загл. с экр. Яз. англ.

ПРИЛОЖЕНИЕ А

Полный SQL-код создания сущностей в базе данных

```
CREATE TABLE author(
  id INT PRIMARY KEY,
  name text,
  date timestamp)

CREATE TABLE reader(
  id INT PRIMARY KEY,
  name text,
  phone text,
  "date" timestamp,
  password text)

CREATE TABLE book(
  id INT PRIMARY KEY,
  author_id integer,
  name text,
  year integer,
  genre_id integer)

CREATE TABLE ekzbook(
  id INT PRIMARY KEY,
  book_id integer,
  publ_id integer,
  year integer,
  libr_id integer,
  datepost timestamp)

CREATE TABLE formulyr(
  id INT PRIMARY KEY,
  reader_id int,
  ekzbook_id int,
  date1 timestamp,
```

```
  date2 timestamp)

CREATE TABLE genre(
  id INT PRIMARY KEY,
  name text)

CREATE TABLE libr(
  id INT PRIMARY KEY,
  name text)

CREATE TABLE "new"(
  id INT PRIMARY KEY,
  book_id int,
  date1 timestamp,
  date2 timestamp)

CREATE TABLE publ(
  id INT PRIMARY KEY,
  name text,
  year int)
```

ПРИЛОЖЕНИЕ Б

Полный код оконного приложения для Читателя на языке Python

```
#  /Users/mariibykova/Desktop/biblioteka
import psycopg2 as pg
import pandas as pd
import tkinter as tki
from tkinter import filedialog
from tkinter import messagebox as mb
from pandas import DataFrame
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import requests
import pandas as pd
import pathlib
from bs4 import BeautifulSoup
from tkinter import *
from tkinter import messagebox
from datetime import datetime
try:
    connection = pg.connect(
        user="reader",
        password="readerserver2024",
        host="localhost",
        port="5432")

    cursor = connection.cursor()
    print("Connection established.")

    def take_new():
        s = url_tf.get() #в какую папку сохранить
        k = 4
        sqlite_select_query = """SELECT * from new"""
        cursor.execute(sqlite_select_query)
        records = cursor.fetchall()
```

```
    s += "/"
    headers = [""]*(k-1)
    # пустой список для колонок таблицы
    mydata = pd.DataFrame(columns = headers)
    a = [""]*(k-1)
    for i in range(len(records)):
        a = [""]*(k-1)
        for j in range(1, k):
            if isinstance(records[i][j], str):
                a[j-1] = records[i][j]
            else:
                a[j-1] = str(records[i][j])
        print(a)
        length = len(mydata)
        mydata.loc[length] = a
    filname = s+name +'.csv'
    mydata.to_csv(filname, index=False)
    messagebox.showinfo('INFO', f'Файл с данными сформирован!')
    nametable_tf.delete(0, END)

def poisk_book_author():
    s = url_tf.get() #в какую папку сохранить
    author = '%'+authors.get()+'%'
    sqlite_select_query = f"""
        SELECT book.name, author.name, book.year,
        gender.name from book, author, gender
        where author.name = {author}
        and book.gender_id = gender.id
        and book.author_id=author.id"""
    cursor.execute(sqlite_select_query)
    records = cursor.fetchall()
    s += "/"
    headers = [""]*(k-1)
    k = 4
```

```
        mydata = pd.DataFrame(columns = headers)
        a = [""]*(k-1)
        for i in range(len(records)):
            a = [""]*(k-1)
            for j in range(1, k):
                if isinstance(records[i][j], str):
                    a[j-1] = records[i][j]
                else:
                    a[j-1] = str(records[i][j])
            print(a)
            length = len(mydata)
            mydata.loc[length] = a
        filname = s+name +'.csv'
        mydata.to_csv(filname, index=False)
        messagebox.showinfo('INFO', f'Файл с данными сформирован!')

        nametable_tf.delete(0, END)

    def poisk_book_name():
        s = url_tf.get() #в какую папку сохранить
        name = '%'+names.get()+'%'
        sqlite_select_query = f"""
            SELECT book.name, author.name, book.year,
            gender.name from book, author, gender
            where book.name = {name}
            and book.gender_id = gender.id
            and book.author_id = author.id"""
        cursor.execute(sqlite_select_query)
        records = cursor.fetchall()
        s += "/"
        headers = [""]*(k-1)
        k = 4
        mydata = pd.DataFrame(columns = headers)
        a = [""]*(k-1)
```

```
        for i in range(len(records)):
            a = [""]*(k-1)
            for j in range(1, k):
                if isinstance(records[i][j], str):
                    a[j-1] = records[i][j]
                else:
                    a[j-1] = str(records[i][j])
            print(a)
            length = len(mydata)
            mydata.loc[length] = a
        filname = s+name +'.csv'
        mydata.to_csv(filname, index=False)
        messagebox.showinfo('INFO', f'Файл с данными сформирован!')

        nametable_tf.delete(0, END)

    def insert_ekzbook():
        #s = url_tf.get() #в какую папку сохранить
        name = '%'+names.get()+'%'
        sqlite_select_query = f"""
            SELECT ekzbook.id from ekzbook, book
            where ekzbook.book_id = book.id
            and book.name = {name} limit 1"""
        cursor.execute(sqlite_select_query)
        records = cursor.fetchall()
        now = datetime.now()
        now1 = now +datetime.timedelta(days=7)
        now = str(now.strftime("%H:%M:%S"))
        now1 = str(now1.strftime("%H:%M:%S"))

        iid = records[0][1]
        sqlite_select_query = f"""
            insert insert into formulyr
            (id, reader_id, ekzbook_id, date1, date2)
```

```
        values (1, 1, {iid}, {now}, {now1})"""
    cursor.execute(sqlite_select_query)
    messagebox.showinfo('INFO', f'Книга добавлена!')

def take_formulyr():
    k = 5
    sqlite_select_query = f"""SELECT * from formulyr"""
    cursor.execute(sqlite_select_query)
    records = cursor.fetchall()
    s += "/"
    headers = [""]*(k-1)
    mydata = pd.DataFrame(columns = headers)
    a = [""]*(k-1)
    for i in range(len(records)):
        a = [""]*(k-1)
        for j in range(1, k):
            if isinstance(records[i][j], str):
                a[j-1] = records[i][j]
            else:
                a[j-1] = str(records[i][j])
        print(a)
        length = len(mydata)
        mydata.loc[length] = a
    filname = s+'my_formulyr.csv'
    mydata.to_csv(filname, index=False)
    messagebox.showinfo('INFO', f'Файл с данными сформирован!')
    nametable_tf.delete(0, END)

def delete_ekzbook():
    name = '%'+names.get()+'%'
    sqlite_select_query = f"""delete from ekzbook
    where ekzbook.name = {name}"""
    cursor.execute(sqlite_select_query)
    messagebox.showinfo('INFO', f'Книга сдана!')
```

```
window = Tk() # создаём окно приложения
window.title('Reader') # вызов функции парсеринга
window.geometry('600x300') # размеры окна
frame = Frame(
   window,
   padx=10,
   pady=10  )
frame.pack(expand=True)
urle_lb = Label( # надпись 1
   frame,
   text="Сохранить в папку: "
)urle_lb.grid(row=1, column=1)
url_tf = Entry( # текстовое поле 1
   frame,
)url_tf.grid(row=1, column=2, pady=4)
new_btn = Button( # кнопка
   frame,
   text='Получить новинки!',
   command=take_new
)new_btn.grid(row=4, column=2)
authors_lb = Label( # надпись 1
   frame,
   text="Автор: "
)authors_lb.grid(row=3, column=1)
authors = Entry( # текстовое поле 1
   frame,
)authors.grid(row=3, column=2, pady=4)
names_lb = Label( # надпись 1
   frame,
   text="Название книги: "
)names_lb.grid(row=2, column=1)
names = Entry( # текстовое поле 1
   frame,
```

```
    )names.grid(row=2, column=2, pady=4)
    book_btn = Button( # кнопка
       frame,
       text='Получить книгу!',
       command=poisk_book_name
    )book_btn.grid(row=4, column=1)
    insert_btn = Button( # кнопка
       frame,
       text='Добавить книгу в мой формуляр!',
       command=insert_ekzbook
    )insert_btn.grid(row=5, column=1)
    delete_btn = Button( # кнопка
       frame,
       text='Удалить книгу из моего формуляра!',
       command=delete_ekzbook
    )delete_btn.grid(row=6, column=1)
    take_btn = Button( # кнопка
       frame,
       text='Какие книги в моем формуляре?',
       command=take_formulyr
    )take_btn.grid(row=5, column=2)

    def on_closing():
       if messagebox.askokcancel("Выход", "Вы хотите выйти?"):
            window.destroy()

    window.protocol("WM_DELETE_WINDOW", on_closing)
    window.mainloop()

finally:
    if connection:
        cursor.close()
        connection.close()
        print("Соединение с PostgreSQL закрыто")
```

ПРИЛОЖЕНИЕ В

Полный код оконного приложения для Администратора на языке Python

```
#  /Users/mariibykova/Desktop/biblioteka
import psycopg2 as pg
import pandas as pd
import tkinter as tki
from tkinter import filedialog
from tkinter import messagebox as mb
from pandas import DataFrame
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import requests
import pandas as pd
import pathlib
from bs4 import BeautifulSoup
from tkinter import *
from tkinter import messagebox
from datetime import datetime
try:
    connection = pg.connect(
        user="admin",
        password="readerserver2024",
        host="localhost",
        port="5432")

    cursor = connection.cursor()
    print("Connection established.")

    def run():
        s = url_tf.get()
        sqlite_select_query = "%" + take_tf.get() + "%"
        cursor.execute(sqlite_select_query)
        records = cursor.fetchall()
        s += "/"
```

```
    k = k = len(records[0])
    headers = [""]*(k-1)
    mydata = pd.DataFrame(columns = headers)
    a = [""]*(k-1)
    for i in range(len(records)):
        a = [""]*(k-1)
        for j in range(1, k):
            if isinstance(records[i][j], str):
                a[j-1] = records[i][j]
            else:
                a[j-1] = str(records[i][j])
        print(a)
        length = len(mydata)
        mydata.loc[length] = a
    filname = s+name +'.csv'
    mydata.to_csv(filname, index=False)
    messagebox.showinfo('INFO', f'Файл с данными сформирован!')
    nametable_tf.delete(0, END)

window = Tk() # создаём окно приложения
window.title('Admin') # вызов функции парсеринга
window.geometry('600x300') # размеры окна
frame = Frame(
   window,
   padx=10,
   pady=10)
frame.pack(expand=True)
urle_lb = Label( # надпись 1
   frame,
   text="Сохранить в папку: "
)urle_lb.grid(row=1, column=1)
url_tf = Entry( # текстовое поле 1
   frame,
)url_tf.grid(row=1, column=2, pady=4)
```

```
    take_lb = Label( # надпись 1
       frame,
       text="Запрос: "
    )take_lb.grid(row=2, column=1)
    take_tf = Entry( # текстовое поле 1
       frame,
    )take_tf.grid(row=2, column=2, pady=4)
    new_btn = Button( # кнопка
       frame,
       text='Выполнить!',
       command=run
    )new_btn.grid(row=3, column=2)
    def on_closing():
       if messagebox.askokcancel("Выход", "Вы хотите выйти?"):
            window.destroy()

    window.protocol("WM_DELETE_WINDOW", on_closing)
    window.mainloop()

finally:
    if connection:
        cursor.close()
        connection.close()
        print("Соединение с PostgreSQL закрыто")
```

ПРИЛОЖЕНИЕ Г

Flash-носитель с отчетом о выполненной работе

Папка `Coursework` — LaTeX- вариант курсовой работы;

Папка `Program` — код программы и рабочий файл file.txt;

`Coursework.pdf` — курсовая работа.