Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

Библиотечная система

Студент ИУ7-61Б _____ **Е. А. Варламова**
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ **К. Л. Тассов**
(Подпись, дата) (И.О.Фамилия)

2022 г.

Содержание

Введение

Библиотека – это учреждение, собирающее и осуществляющее хранение книг и иных печатных изданий для общественного пользования.

Электронная библиотечная система (ЭБС) является средством управления хранения книг и процессом получения доступа к ним. Такие системы значительно упрощают процесс учёта книг из библиотечного фонда. В настоящее время ЭБС используются многими организациями для обеспечения своих сотрудников необходимой литературой, университетами и школами – для организации учебного процесса, а также городскими библиотеками.

Цель работы – разработать базу данных электронной библиотечной системы, которая позволит читателям получать информацию о доступных книгах в разных библиотеках, библиотекарям – выдавать и принимать книги, а администраторам библиотечной системы – редактировать информацию о книгах и библиотеках.

Для достижения поставленной цели требуется решить следующие задачи:

- проанализировать существующие решения задачи, выявить их недостатки;
- сформулировать требования к разрабатываемому приложению с учётом выявленных недостатоков у аналогов;
- проанализировать варианты представления данных и выбрать подходящий вариант для решения задачи;
- спроектировать базу данных, описать ее сущности и связи;
- разработать архитектуру приложения;
- реализовать программное обеспечение на основе разработанной структуры приложения;
- провести исследование производительности в зависимости от наличия индексов.

1 Аналитический раздел

В данном разделе проводится анализ существующих ЭБС и способов хранения данных, а также формализуются задача, структура хранимых данных и требования к системе.

1.1 Обзор существующих решений

1.1.1 ЭБС «Лань»

ЭБС «Лань» – электронно-библиотечная система, предоставляющая студентам, аспирантам и преподавателям подключенных библиотек доступ к чтению электронных версий книг, журналов. Для поиска информационных ресурсов организован гибкий поиск по различным фильтрам. Система не предоставляет возможности получения бумажных книг, а значит и интеграции в обычные библиотеки.

1.1.2 Московская ЭБС

Московская ЭБС предоставляет возможность просмотра доступных книг во всех библиотеках Москвы, при этом функционал фильтрации довольно ограниченный. Система не предоставляет возможности онлайн доступа к информационным ресурсам и не поддерживает добавление библиотек, находящихся в других городах.

1.1.3 eLibrary

eLibrary – научная электронная библиотека, крупнейший российский информационно-аналитический портал в области науки, технологии, медицины и образования, предоставляющий возможность разнообразной фильтрации и поиска. Система работает только с электронными ресурсами. Кроме того, система ориентирована только на научные труды, статьи и рефераты.

1.1.4 Сравнение существующих решений

Таблица 1.1: Сравнительная таблица существующих решений

ЭБС	Возможность использования в обычных библиотеках	Возможность разнообразной фильтрации	Возможность использования большой аудиторией
Лань	-	+	+
ЭБС Москвы	+	-	-
eLibrary	-	+	+

На основе анализа существующих решений было выяснено, что в сети Интернет есть много ресурсов, позволяющих получить доступ к книгам в электронном виде. Такие сервисы предлагают возможность разнообразной фильтрации, однако они могут быть использованы исключительно в онлайн-режиме. Возможность интеграции в реальные библиотеки они не предусматривают. С другой стороны, во многих городах России есть системы, позволяющие записаться в библиотеки города или посмотреть наличие книг в них. Недостатком таких систем является то, что они не предоставляют возможности гибко фильтровать книги и то, что в разных городах (а иногда и в разных сетях библиотек города) системы разные.

1.2 Формализация задачи

Требуется разработать электронную библиотечную систему, которая позволит читателям получать информацию о доступных книгах в разных библиотеках, а также информацию о ранее взятых книгах, библиотекарям – выдавать и принимать книги, а администраторам библиотечной системы – редактировать информацию о книгах и библиотеках.

Таким образом, выделяются 3 роли: читатель, библиотекарь, администратор.

Взаимодействие с продуктом происходит по схеме, представленной на рисунке 1.1.

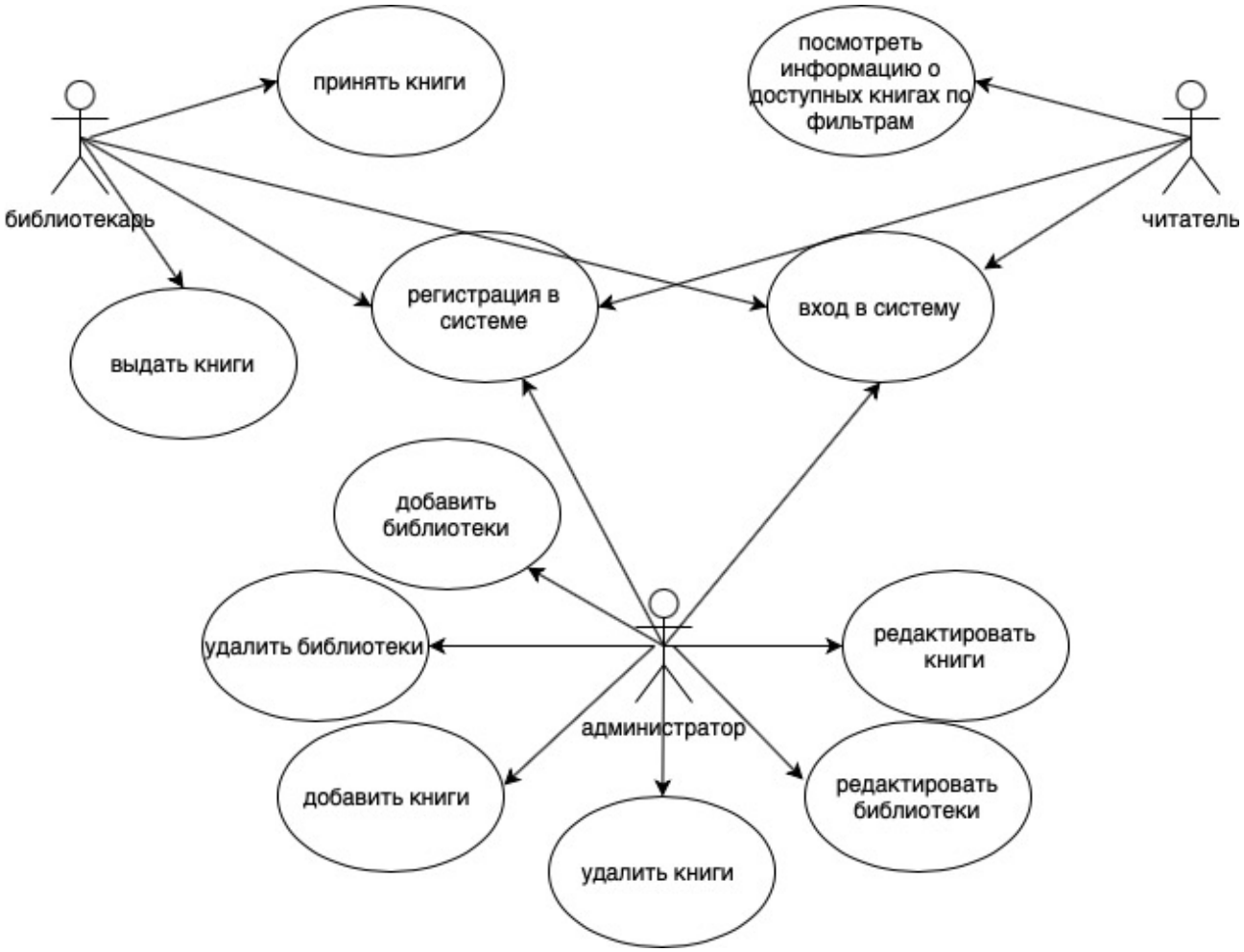


Рис. 1.1: Use-case диаграмма

1.3 Базы данных и структура хранимых данных

В задаче разработки электронной библиотечной системы важную роль играет выбор модели хранения данных. Наиболее общим подходом к хранению данных является использование баз данных [1]. Для управления базами используется системы управления базами данных – СУБД [2]. Система управления базами данных – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных.

Базы данных по способу хранения можно классифицировать следующим образом:

1. строковые базы данных;
2. колоночные базы данных.

Строковые базы данных

Строковыми базами данных называются базы данных, данные в которых хранятся построчно. Базы данных этого типа применяются в системах, для которых характерно наличие большого количества коротких транзакций вставки, удаления, изменения. Важным показателем таких систем является их отзывчивость, то есть запросы должны обрабатываться быстро.

Колоночные базы данных

Колоночными базами данных называются базы данных, данные в которых хранятся по столбцам. Базы данных этого типа применяются в системах, для которых характерно небольшое количество транзакций, однако при этом запросы могут быть достаточно сложными.

1.3.1 Выбор модели хранения данных

Для реализации библиотечной системы было решено использовать построчное хранение данных, поскольку:

- предполагается большое количество коротких транзакций;
- предполагается высокий уровень отзывчивости системы;
- не предполагается выполнение сложных аналитических запросов.

1.3.2 Формализация структуры хранимых данных

Для создания базы данных, хранящей библиотечную систему, выделены слудующие сущности предметной области: книга, экземпляр книги, библиотека, аккаунт, администратор, библиотекарь, читатель.

На рисунке 1.2 представлена диаграмма сущность-связь для данной задачи.

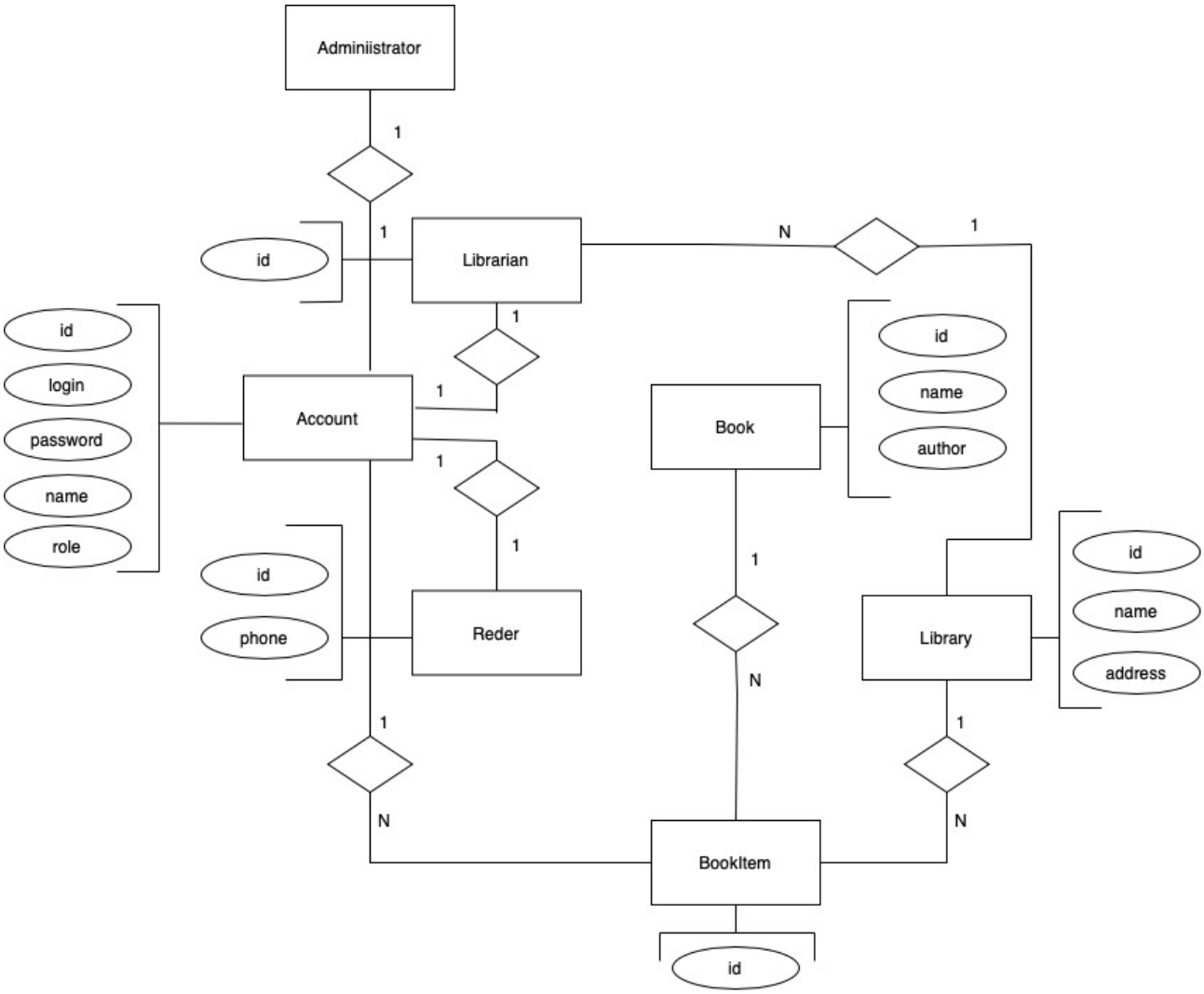


Рис. 1.2: Диаграмма сущность-связь

Вывод

В данном разделе был проведён анализ существующих ЭБС и способов хранения данных. Было решено использовать строковую базу данных. Кроме того, были формализованы задача, структура хранимых данных и требования к системе.

2 Конструкторский раздел

В данном разделе будет спроектированы база данных и приложение для доступа к ней.

2.1 Проектирование базы данных

2.1.1 Описание таблиц базы данных

В базе данных библиотечной системы 7 таблиц.

Таблица `Library` описывает основную информацию о библиотеках. Её поля:

Поля таблицы `Library`:

- `id` – идентификатор библиотеки (целое);
- `name` – название библиотеки (строка);
- `address` – автор библиотеки (строка).

Таблица `Account` описывает основную информацию о библиотеках. Её поля:

Поля таблицы `Account`:

- `id` – идентификатор аккаунта (целое);
- `login` – логин пользователя (строка, уникальный);
- `password` – пароль пользователя (строка);
- `name` – имя пользователя (строка);
- `role` – роль пользователя в системе (администратор, библиотекарь, читатель) (строка).

Таблица `Book` описывает основную информацию о книгах. Её поля:

- `id` – идентификатор книги (целое);

- `name` – название книги (строка);
- `author` – автор книги (строка).

Таблица `BookItem` описывает экземпляр книги. Данная таблица реализует 3 связи 1 ко многим. Её поля:

- `id` – идентификатор экземпляра книги (целое);
- `bokk_id` – идентификатор дескриптора книги (целое);
- `lib_id` – идентификатор библиотеки, в которой хранится экземпляр книги (целое);
- `acc_id` – идентификатор аккаунта читателя (целое или пустое, если книга в библиотеке).

Таблица `Librarian` описывает сущность библиотекаря. Её поля:

- `id` – идентификатор аккаунта библиотекаря (целое);
- `lib_id` – идентификатор библиотеки, в которой работает библиотекарь (целое).

Таблица `Administrator` описывает сущность администратора. Её поля:

- `id` – идентификатор аккаунта администратора (целое).

Таблица `Librarian` описывает сущность читателя. Её поля:

- `id` – идентификатор аккаунта читателя (целое);
- `lib_id` – телефон читателя (строка).

2.1.2 Описание функций базы данных

Кроме того, в базе данных определены 2 функции:

1. Функция взятия книги в библиотеке, состоящая из следующих действий: по логину библиотекаря вычисляется идентификатор библиотеки, в которой библиотекарь имеет возможность выдавать книги; вычисляется идентификатор читателя по логину читателя; идентификатор читателя ставится в поле `acc_id` экземпляру книги, находящемуся в данной библиотеке.

2. Функция возврата книги в библиотеку, состоящая из следующих действий: по логину библиотекаря вычисляется идентификатор библиотеки, в которой библиотекарь имеет возможность выдавать книги; вычисляется идентификатор читателя по логину читателя; поле `acc_id` становится пустым у того экземпляра книги, который находится в данной библиотеке и взят читателем с найденным идентификатором.

На рисунке 2.1 представлены схемы алгоритмов описанных функций.

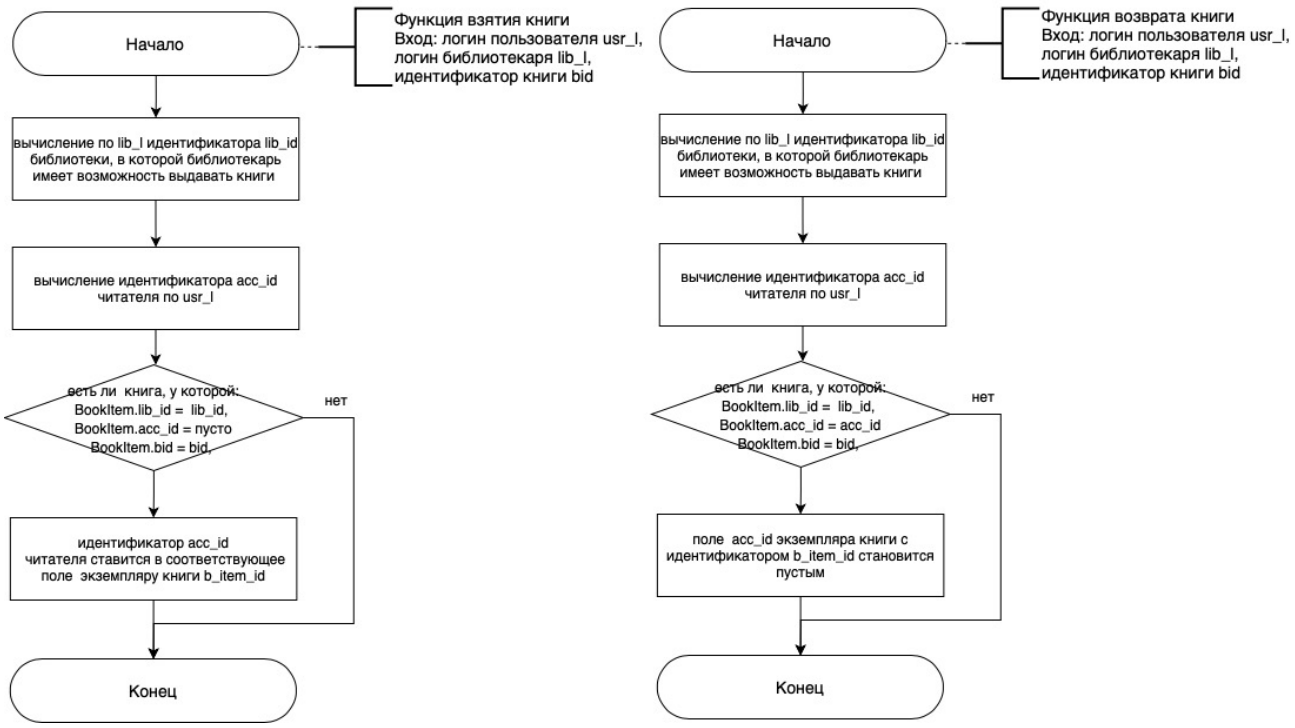


Рис. 2.1: Схемы функций базы данных

2.1.3 Описание триггеров базы данных

Также для поддержания логической непротиворечивости базы данных определено 2 триггера:

1. `DML-триггер before delete`, определённый на таблице `Account`. До удаления аккаунта все книги, взятые пользователем, принудительно возвращаются в бибиотеку.

2. `DML-триггер before delete`, определённый на таблице `Library`. До удаления все аккаунты библиотекарей, работающих в данной библиотеке, удаляются.

На рисунке 2.2 представлены схемы алгоритмов описанных триггеров.

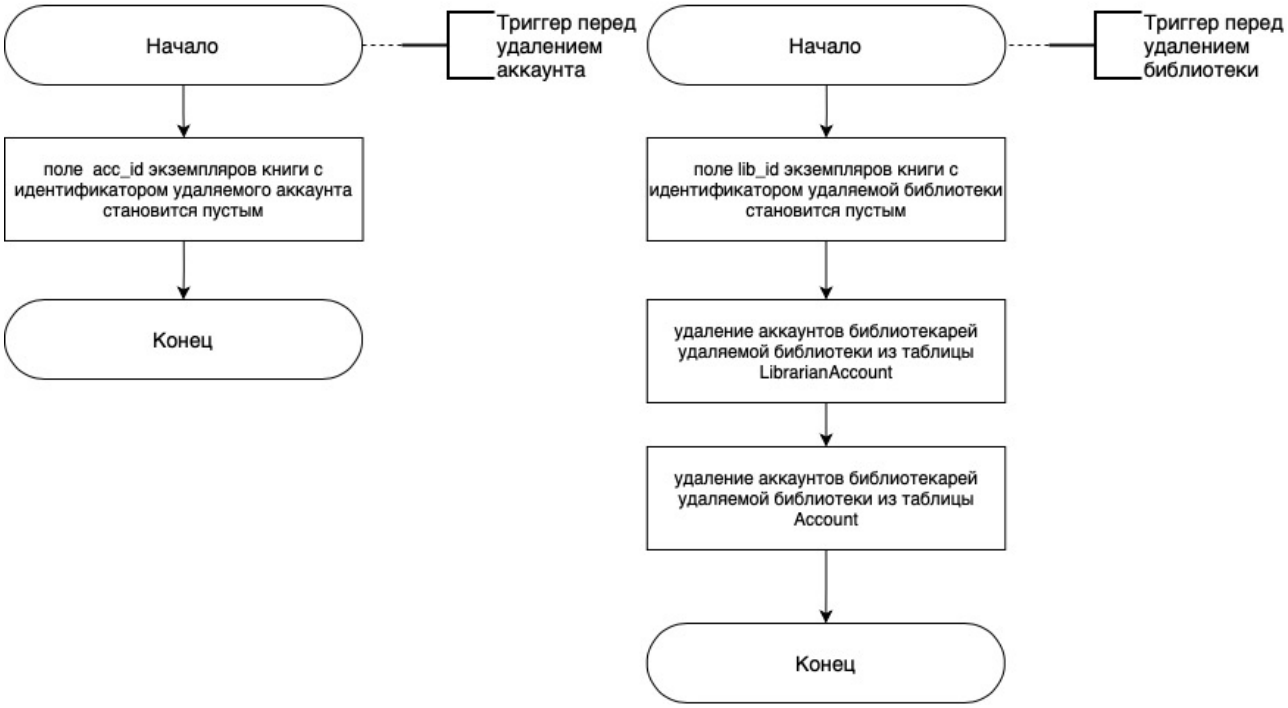


Рис. 2.2: Схемы триггеров базы данных

2.1.4 Описание транзакций базы данных

Для поддержания целостности базы данных используются следующие транзакции:

1. при добавлении книги проверяется, есть ли уже книга с таким названием и автором; если есть, то добавляется экземпляр книги и возвращается его идентификатор; если нет, то обновляются две таблицы: дескрипторов книг и экземпляров книг, возвращается идентификатор экземпляра;

2. при изменении книги проверяется, существует ли библиотека с указанным идентификатором; если существует, то обновляются дескриптор книги (обновляются все поля, кроме идентификатора) и экземпляр книги (меняется идентификатор библиотеки); если нет, то происходит откат;

3. при добавлении, удалении и изменении аккаунта обновляются 2 таблицы: таблица аккаунтов и таблица, соответсвующая роли; кроме того, при добавлении и изменении аккаунта библиотекаря проверяется, существует ли библиотека с указанным идентификатором;

2.1.5 Ролевая модель

Для безопасной работы с данными вводится 3 типа пользователей:

1. администраторы, которые имеют полный доступ ко всем объектам базы данных;

2. библиотекари, которые имеют права на обновление таблицы экземпляров книг и на чтение всех остальных таблиц;

3. читатели, которые имеют права только на чтение всех таблиц.

2.2 Проектирование приложения

Для реализации поставленной задачи наиболее подходящей является клиент-серверная архитектура, так как она позволяет нескольким клиентам работать одновременно с приложением. Таким образом, приложение должно состоять из 3 основных компонентов:

- компонент доступа к данным, который реализует обращение к базе данных и предоставляет интерфейс в виде доступных команд;
- компонент бизнес-логики, который занимается диспетчеризацией поступающих запросов соответствующим командам компонента доступа к данным;
- компонент интерфейса, который обращается к компоненту бизнес-логики за данными и занимается их отображением.

Связь компонентов показана на рисунке 2.3.

Рис. 2.3: Схема компонентов приложения

Вывод

В данном разделе в качестве архитектуры приложения была выбрана клиент-серверная архитектура и приведена схема компонентов для её реализации. Кроме того, была спроектирована база данных: описаны её триггеры, функции, сущности и связи между ними, а также транзакции.

3 Технологический раздел

В данном разделе выбраны средства разработки программного обеспечения, показаны детали реализации и способы взаимодействия с программным продуктом.

3.1 Средства реализации ПО

В качестве технологии программирования был выбран объектно-ориентированный подход, так как он обеспечивает простую модификацию кода и добавление нового функционала за счёт использования различных паттернов проектирования.

Для разработки приложения был выбран язык программирования C++. Этот язык подходит для реализации разрабатываемого программного продукта, так как поддерживает парадигму объектно-ориентированного программирования, обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, механизм виртуальности.

Для связи клиента и сервера по REST-API был использован фреймворк OAT++ [3], так как он может быть легко встроен в код приложения, кроме того, он предоставляет возможность асинхронной обработки поступающих запросов.

В качестве СУБД была выбрана PostgreSQL [4], так как она удовлетворяет требованиям, выдвинутым в разделе 1.3. Кроме того, она предоставляет возможность использования индексов.

В качестве библиотеки доступа к данным была выбрана библиотека SOCI [5], так как она предоставляет возможность обращения к разным базам данных с минимальным изменением кода (требуется изменить только строку подключения). Кроме того, библиотека предоставляет интерфейс для ORM (Object-Relational-Mapping).

Для реализации графической части приложения был использован фреймворк QT [6], так как он включает в себя все основные классы, которые могут потребоваться при разработке прикладного программного обеспечения, включая элементы графического интерфейса.

3.2 Архитектура компонентов приложения

3.2.1 Архитектура компонента интерфейса

Для реализации компонента интерфейса был выбран паттерн MVP, так как он позволяет создать несколько вариантов отображения данных без изменения логики. Так, интерфейс приложения был разработан в 2 вариантах: в графическом и консольном. На рисунке 3.1 представлена UML-диаграмма классов компонента интерфейса.

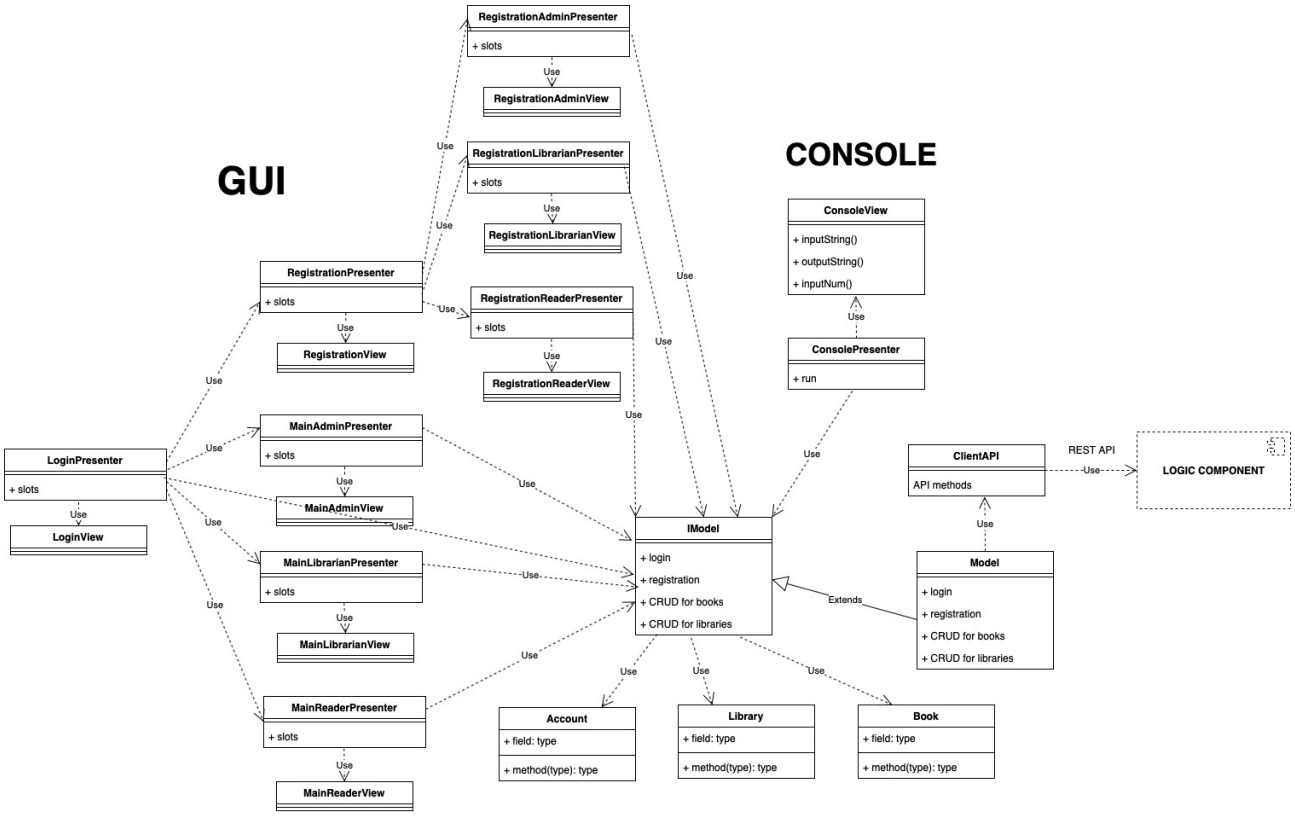


Рис. 3.1: UML-диаграмма классов компонента интерфейса

3.2.2 Архитектура компонента бизнес-логики

Компонент бизнес-логики представляет собой REST-API контроллер сервера и класс-менеджер для доступа к данным. На рисунке 3.2 представлена UML-диаграмма классов компонента бизнес-логики.

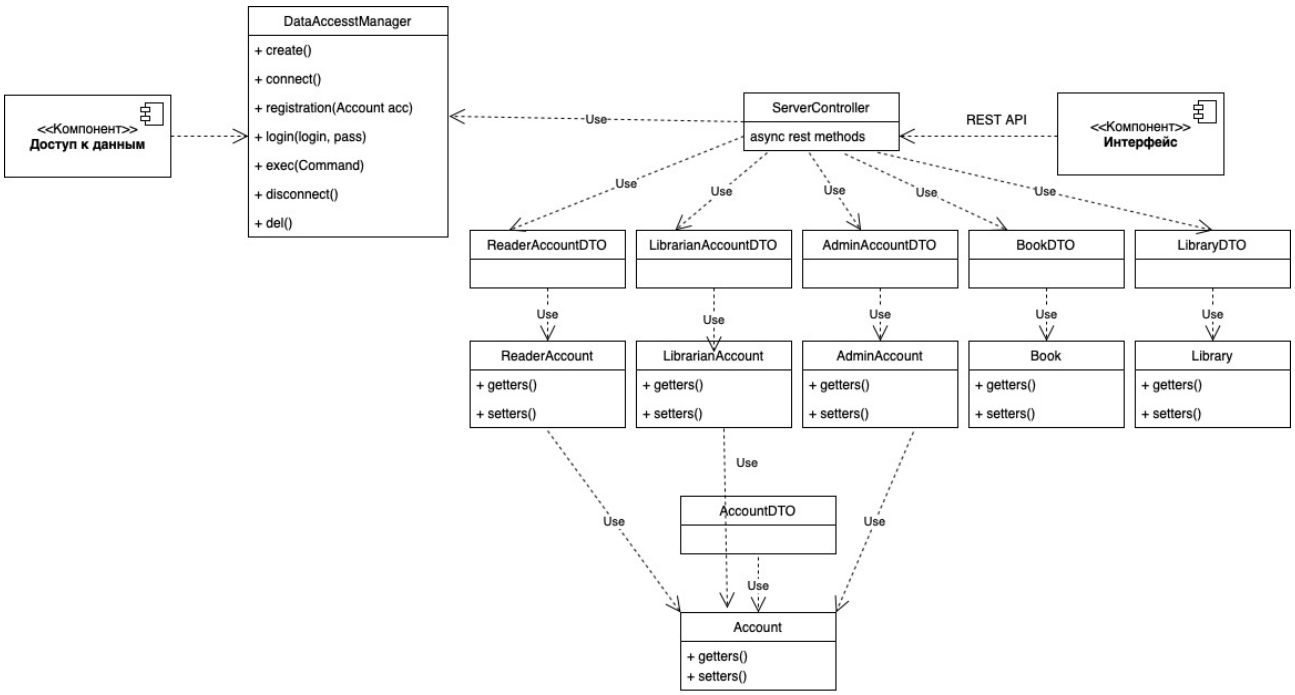


Рис. 3.2: UML-диаграмма классов компонента бизнес-логики

3.2.3 Архитектура компонента доступа к данным

Компонент доступа к данным состоит из следующих наиболее важных классов.

- Класс `Session`, который абстрагирует понятие сессии. Класс является надстройкой над соответсвующим классом библиотеки SOCI.
- Класс `ConnectionPoool`, который представляет собой набор сессий базы данных. Класс является надстройкой над соответсвующим классом библиотеки SOCI.
- Классы репозиториев: `AccountRepository`, `AdminAccountRepository`, `LibrarianAccountRepository`, `ReaderAccountRepository`, `BookRepository`, `LibraryRepository`, которые являются интерфейсом для получения данных о соответствующей сущности из базы. Все репозитории реализуют CRUD-методы (CRUD – Create & Remove & Update & Delete).
- Классы команд (`AddBookCommand`, `UpdateLibraryCommand`, и. т. д.).

На рисунке 3.3 представлена UML-диаграмма классов компонента доступа к данным.

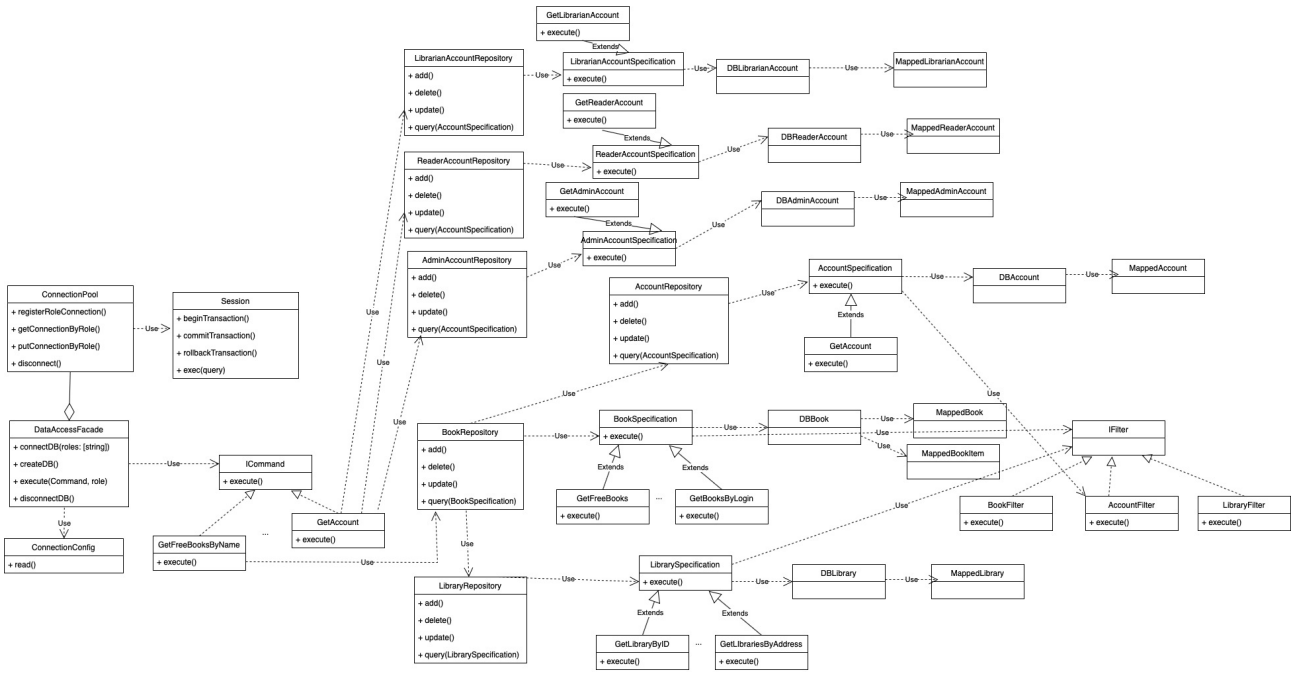


Рис. 3.3: UML-диаграмма классов компонента доступа к данным

3.3 Создание объектов БД

3.3.1 Создание таблиц

В листингах 3.1 - 3.7 приведён код создания таблиц БД.

Таблица аккаунта каждого пользователя:

```
1 create table if not exists Account (id serial primary key,
2                                     login varchar(30) unique,
3                                     password varchar(30),
4                                     name varchar(30),
5                                     role varchar(30) );
```

Листинг 3.1: Создание таблицы Account

Таблица, представляющая сущность библиотеки:

```
1 create table if not exists Library (id serial primary key,
2                                     name varchar(30),
3                                     address varchar(30) );
```

Листинг 3.2: Создание таблицы Library

Таблица аккаунта библиотекаря:

```
1 create table if not exists LibrarianAccount (id serial primary key,
2                                               acc_id int references Account(
    id),
3                                               lib_id int references Library(
    id) );
```

Листинг 3.3: Создание таблицы LibrarianAccount

Таблица аккаунта администратора:

```
1 create table if not exists AdminAccount (id serial primary key,
2                                          acc_id int references Account(id) )
    ;
```

Листинг 3.4: Создание таблицы AdminAccount

Таблица аккаунта читателя:

```
1 create table if not exists ReaderAccount (id serial primary key,
2                                           acc_id int references Account(id),
3                                           phone varchar(40) );
```

Листинг 3.5: Создание таблицы ReaderAccount

Таблица, представляющая дескриптор книги:

```
1 create table if not exists Book (id serial primary key,
2                                  name varchar(40),
3                                  author varchar(40) );
```

Листинг 3.6: Создание таблицы Book

Таблица, представляющая экземпляр книги:

```
1 create table if not exists BookItem (id serial primary key,
2                                      book_id int references Book(id),
3                                      lib_id int references Library(id),
4                                      acc_id int references Account(id) );
```

Листинг 3.7: Создание таблицы BookItem

3.3.2 Создание триггеров

В листингах 3.8 - 3.9 приведён код создания триггеров БД.

DML-триггер, который срабатывает до удаления записи из таблицы Account:

```
create or replace function f_delete_acc_trigger()
    returns trigger
as
$$
begin
    update BookItem
    set acc_id = null
    where acc_id = old.id;

    return old;
end
$$  language plpgsql;

create or replace trigger bef before delete on Account
    for each row
    execute procedure f_delete_acc_trigger();
```

Листинг 3.8: Создание триггера (до удаления аккаунта)

DML-триггер, который срабатывает до удаления записи из таблицы Library:

```
create or replace function f_delete_lib_trigger()
returns trigger
as
$$
declare
    ids integer[];
begin
    delete from BookItem
    where lib_id = old.id;

    select into ids array_agg(LibrarianAccount.acc_id)
    from LibrarianAccount join Library on Library.id = lib_id
    where lib_id = old.id;

    delete from LibrarianAccount
    where lib_id = old.id;

    delete from Account
    where id = any(ids);

    return old;
end
```

```
23 $$  language plpgsql;
24
25 create or replace trigger bef_lib before delete on Library
26     for each row
27     execute procedure f_delete_lib_trigger();
```

Листинг 3.9: Создание триггера (до удаления библиотеки)

3.3.3 Создание функций

В листингах 3.10 - 3.11 приведён код создания функций БД.

Функция взятия книги из библиотеки:

```
1  create or replace function take_book(login_user varchar, login_lib varchar,
       bid_p int)
2      returns int
3  as $$
4  declare
5      lid int;
6    aid int;
7    biid int;
8  begin
9      select into lid max(lib_id)
10     from LibrarianAccount join Library on Library.id = lib_id join Account
       on acc_id = Account.id
11     where login_lib = login;
12
13     select into aid max(id)
14     from Account
15     where login = login_user;
16
17     select into biid min(bookitem.id)
18     from bookitem join book on book_id = Book.id
19     where bookitem.id = bid_p and lib_id = lid and acc_id is null;
20
21     update bookitem
22     set acc_id = aid
23     where id = biid;
24     return biid;
25 end;
26 $$
27 LANGUAGE PLPGSQL;
```

Листинг 3.10: Создание функции взятия книги

Функция возвращения книги в библиотеку:

```
create or replace function return_book(login_user varchar, login_lib varchar
    , bid_p int)
returns int
as $$
declare
lid int;
  aid int;
  biid int;
begin
    select into lid max(lib_id)
    from LibrarianAccount join Library on Library.id = lib_id join Account
   on acc_id = Account.id
    where login_lib = login;

    select into aid max(id)
    from Account
    where login = login_user;

    select into biid min(bookitem.id)
    from bookitem join book on book_id = Book.id
    where bookitem.id = bid_p and lib_id = lid and acc_id = aid;

    update bookitem
    set acc_id = null
    where id = biid;
    return biid;
end;
$$
LANGUAGE PLPGSQL;
```

Листинг 3.11: Создание функции возвращения книги

3.3.4 Создание ролей

В листингах 3.12 - 3.13 приведён код создания ролей БД.

Роль библиотекаря с правами на изменение таблицы экземпляров книг и на чтение всех остальных:

```
create user librarian WITH PASSWORD 'librarian';
grant select on book, bookitem, account, librarianaccount, library to
    librarian;
grant update on bookitem to librarian;
```

Листинг 3.12: Создание ролей

Роль читателя с правами только на чтение всех таблиц:

```
1 create user reader WITH PASSWORD 'reader';
2 grant select on book, bookitem, account, readeraccount, library to reader;
```

Листинг 3.13: Создание ролей

3.4 Интерфейс программы

На рисунках 3.4 – 3.6 представлены примеры работы программы.

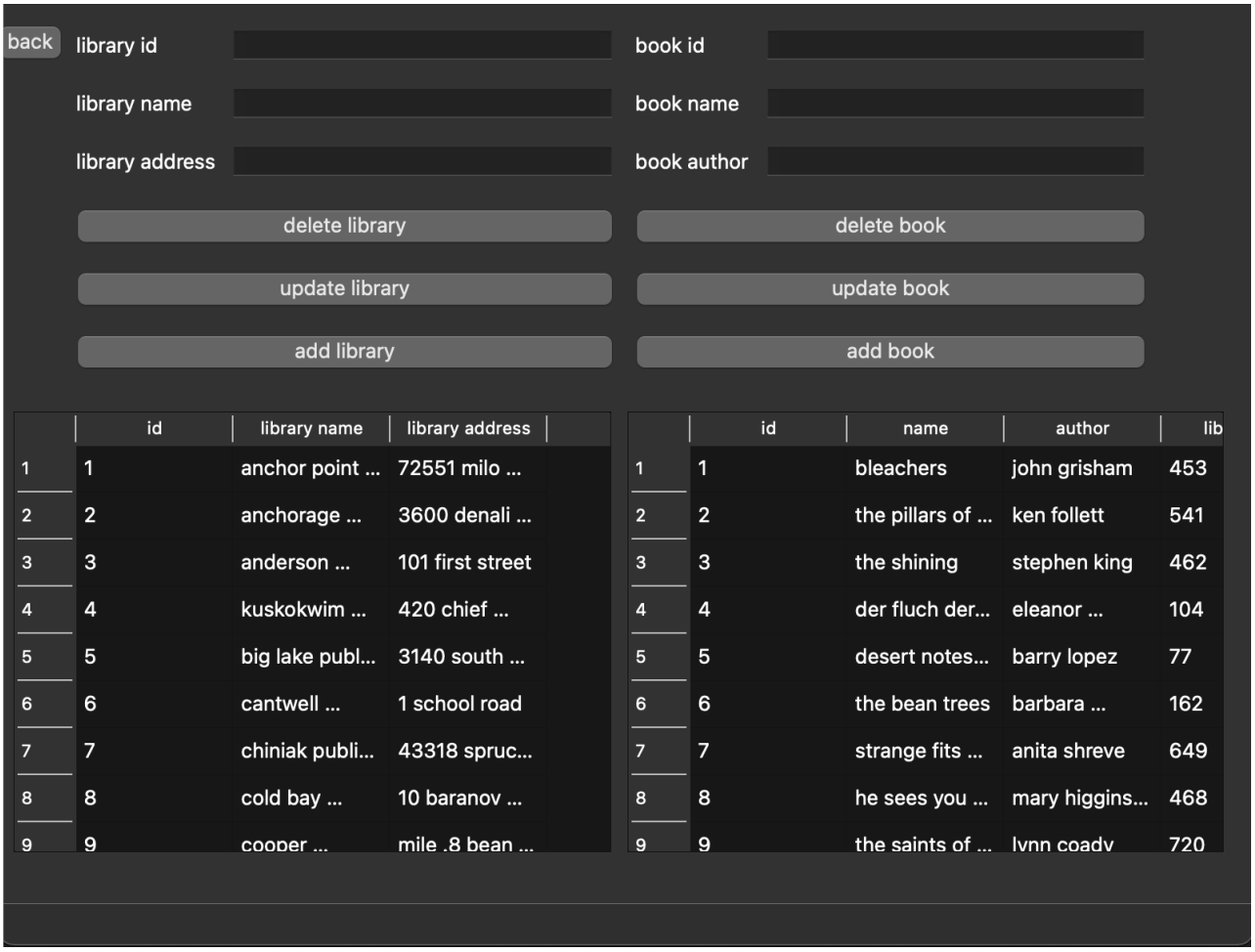


Рис. 3.4: Главная страница администратора

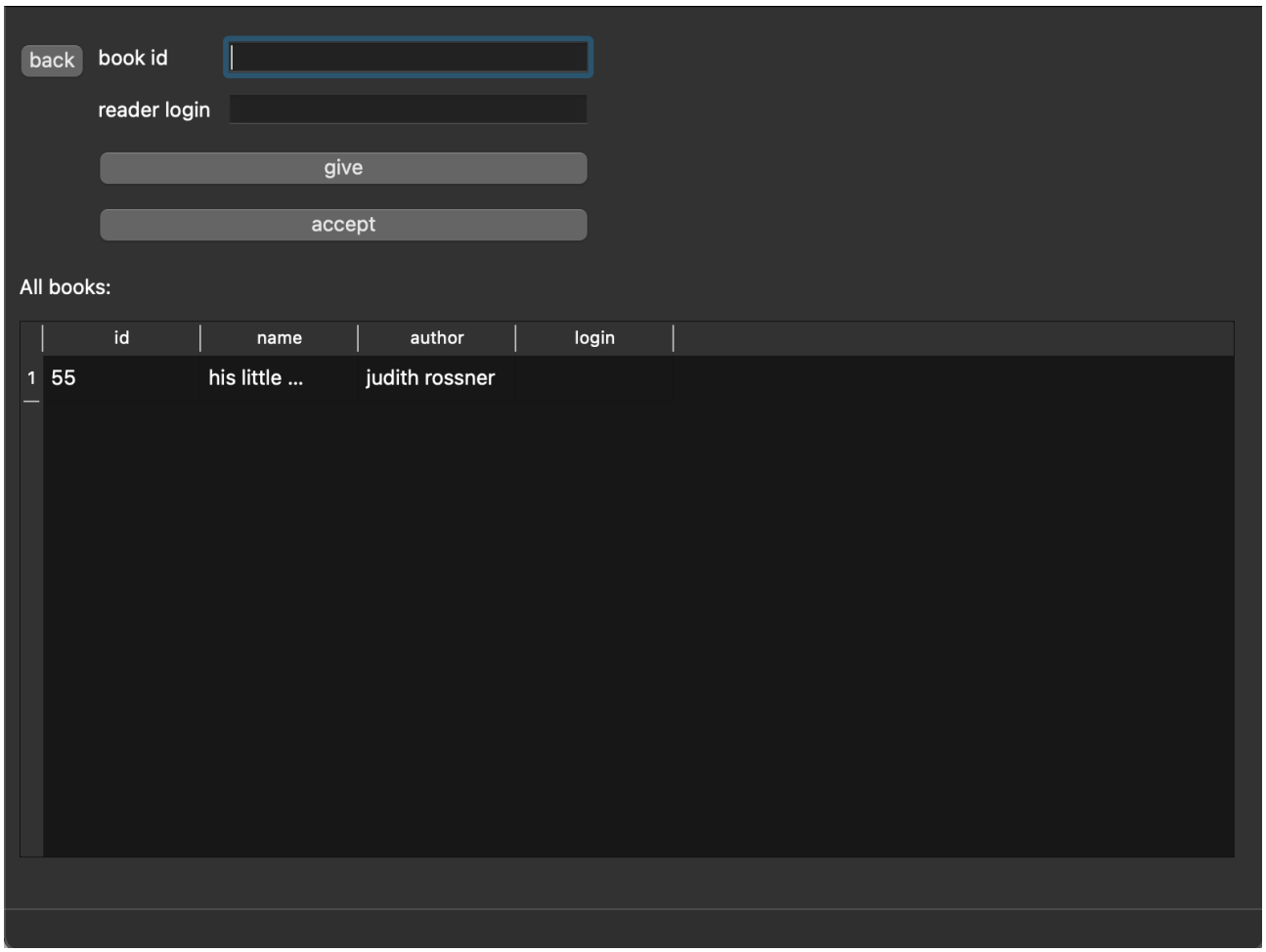


Рис. 3.5: Главная страница библиотекаря

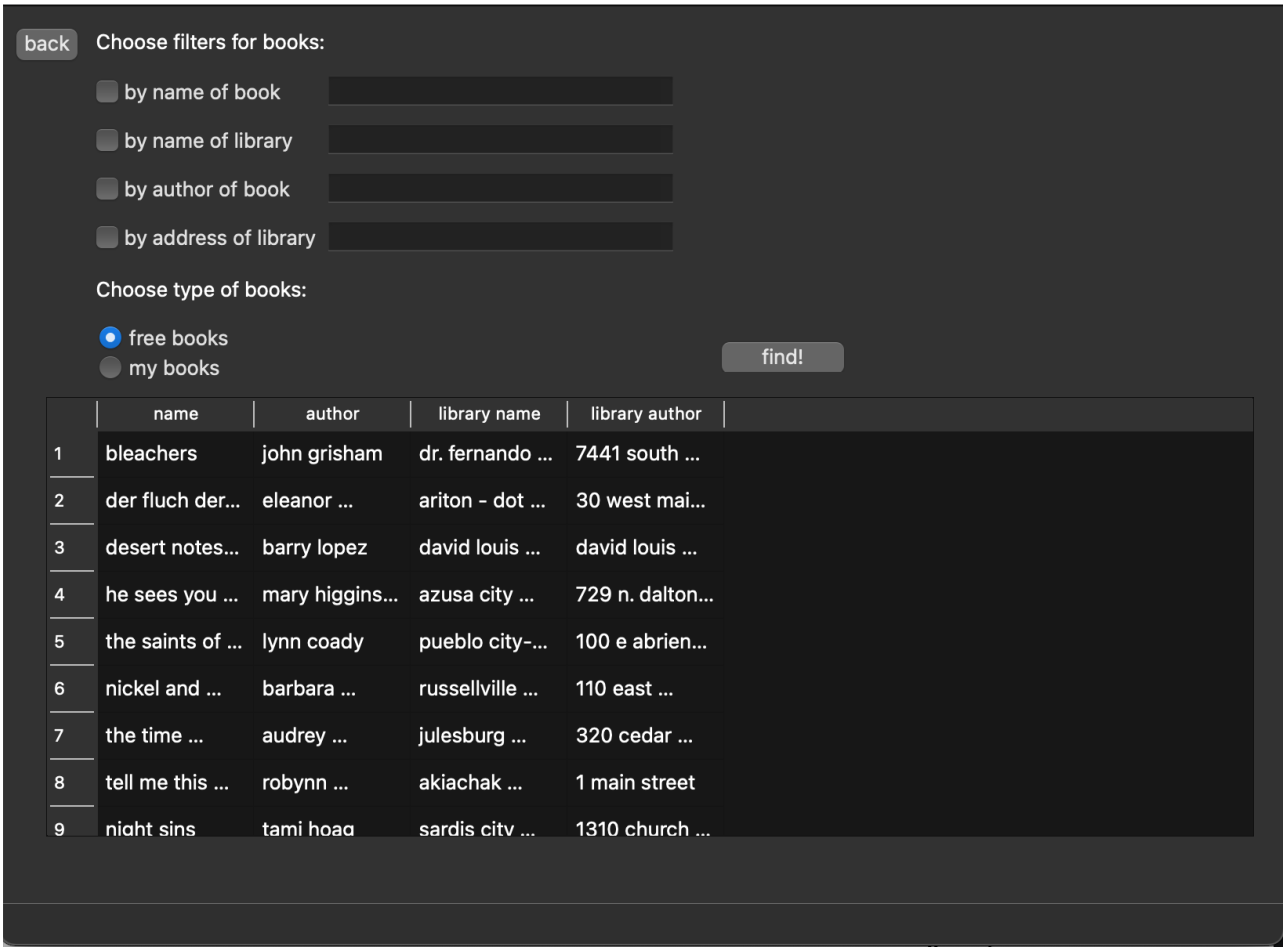


Рис. 3.6: Главная страница читателя

Вывод

В данном разделе были выбраны средства разработки программного обеспечения, показаны детали реализации и способы взаимодействия с программным продуктом.

4 Экспериментальный раздел

В данном разделе проводится анализ врмени выполнения запросов в зависимости от наличия индексов. Все исследования проводились на таблицах, состоящих из 1000 записей. Для повышения производительности используются B-tree индексы, так как с помощью B-дерева можно проиндексировать любые данные, которые могут быть отсортированы, т. е. для которых применимы операции сравнения больше/меньше/равно.

4.1 Поиск по первичному ключу

Выполним следующий запрос:

```
1 select * from Account where id = 1;
```

Листинг 4.1: Запрос поиска по первичному ключу

```
postgres=# explain (analyze) select * from Account where id = 1;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Index Scan using account_pkey on account  (cost=0.28..8.29 rows=1 width=50) (actual time=0.024..0.026 rows=1 loops=1)
   Index Cond: (id = 1)
 Planning Time: 0.113 ms
 Execution Time: 0.050 ms
(4 rows)
```

Рис. 4.1: Поиск по первчиному ключу без индекса

Создадим индекс:

```
1 create index id_idx on account using btree(id);
```

Листинг 4.2: Создание индекса для поиска по первичному ключу

```
postgres=# create index id_idx on account using btree(id);
CREATE INDEX
postgres=# explain (analyze) select * from Account where id = 1;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Index Scan using id_idx on account  (cost=0.28..8.29 rows=1 width=50) (actual time=0.043..0.044 rows=1 loops=1)
   Index Cond: (id = 1)
 Planning Time: 0.459 ms
 Execution Time: 0.068 ms
(4 rows)
```

Рис. 4.2: Поиск по первчиному ключу с индексом

Видим, что время выполнения запроса почти не изменилось, что говорит о том, что для первичного ключа индекс создаётся автоматически, поэтому создание ещё одного индекса не приведёт к повышению производительности.

4.2 Поиск по таблице с фильтрацией

Выполним следующий запрос:

```
1 select id, login, password, name, role from account where name = 'Eric';
```

Листинг 4.3: Запрос поиска с фильтрацией

```
postgres=# explain (analyze) select id, login, password, name, role from account where name = 'Eric';
                                                 QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on account  (cost=0.00..23.51 rows=4 width=50) (actual time=0.060..0.255 rows=4 loops=1)
   Filter: ((name)::text = 'Eric'::text)
   Rows Removed by Filter: 997
 Planning Time: 0.124 ms
 Execution Time: 0.275 ms
(5 rows)
```

Рис. 4.3: Поиск с фильтрацией без индекса

Создадим индекс:

```
1 create index name_idx on account using btree(name);
```

Листинг 4.4: Создание индекса для поиска с фильтрацией

```
postgres=# create index name_idx on account using btree(name);
CREATE INDEX
postgres=# explain (analyze) select id, login, password, name, role from account where name = 'Eric';
                                                       QUERY PLAN
-------------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on account  (cost=4.31..13.12 rows=4 width=50) (actual time=0.082..0.088 rows=4 loops=1)
   Recheck Cond: ((name)::text = 'Eric'::text)
   Heap Blocks: exact=3
   ->  Bitmap Index Scan on name_idx  (cost=0.00..4.31 rows=4 width=0) (actual time=0.074..0.074 rows=4 loops=1)
         Index Cond: ((name)::text = 'Eric'::text)
 Planning Time: 0.470 ms
 Execution Time: 0.125 ms
(7 rows)
```

Рис. 4.4: Поиск с фильтрацией с индексом

Видим, что время выполнения запроса уменьшилось в 2.2 раза, что говорит о том, что создание индекса приводит к повышению производительности.

4.3 Поиск по внешнему ключу при объединении таблиц

Выполним следующий запрос:

```
1 select bookitem.id, name, author from bookitem join book on book.id =
    bookitem.book_id where bookitem.book_id = 5;
```

Листинг 4.5: Запрос поиска по внешнему ключу при объединении таблиц

```
postgres=# explain (analyze) select bookitem.id, name, author from bookitem join book on book.id = bookitem.book_id where bookitem.book_id = 5;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.28..26.80 rows=1 width=53) (actual time=0.101..0.241 rows=2 loops=1)
   ->  Seq Scan on bookitem  (cost=0.00..18.50 rows=1 width=8) (actual time=0.079..0.211 rows=2 loops=1)
         Filter: (book_id = 5)
         Rows Removed by Filter: 998
   ->  Index Scan using book_pkey on book  (cost=0.28..8.29 rows=1 width=53) (actual time=0.011..0.012 rows=1 loops=2)
         Index Cond: (id = 5)
 Planning Time: 0.170 ms
 Execution Time: 0.277 ms
(8 rows)
```

Рис. 4.5: Поиск по внешнему ключу при объединении таблиц без индекса

Создадим индекс:

```
1 create index ind_book_id on bookitem using btree(book_id);
```

Листинг 4.6: Создание индекса для поиска по внешнему ключу при объединении таблиц

```
postgres=# create index ind_book_id on bookitem using btree(book_id);
CREATE INDEX
postgres=# explain (analyze) select bookitem.id, name, author from bookitem join book on book.id = bookitem.book_id where bookitem.book_id = 5;
                                                      QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------
 Nested Loop  (cost=0.55..16.60 rows=1 width=53) (actual time=0.063..0.070 rows=2 loops=1)
   ->  Index Scan using ind_book_id on bookitem  (cost=0.28..8.29 rows=1 width=8) (actual time=0.037..0.039 rows=2 loops=1)
         Index Cond: (book_id = 5)
   ->  Index Scan using book_pkey on book  (cost=0.28..8.29 rows=1 width=53) (actual time=0.012..0.012 rows=1 loops=2)
         Index Cond: (id = 5)
 Planning Time: 0.453 ms
 Execution Time: 0.094 ms
(7 rows)
```

Рис. 4.6: Поиск по внешнему ключу при объединении таблиц с индексом

Видим, что время выполнения запроса уменьшилось в 2.95 раза, что говорит о том, что создание индекса приводит к повышению производительности.

Вывод

В результате исследования было выяснено, что использование индексов в большинстве случаев повышает производительность выполнения запросов.

Заключение

В результате работы было разработано приложение электронной библиотечной системы, а также решены следующие задачи:

- проанализированы существующие решения задачи и выявлены их недостатки;
- сформулированы требования к разрабатываемому приложению с учётом выявленных недостатоков у аналогов;
- проанализированы варианты представления данных и выбран подходящий вариант для решения задачи;
- спроектированв база данных, описаны ее сущности и связи;
- разработана архитектура приложения;
- реализовано программное обеспечение на основе разработанной структуры приложения;
- проведено исследование производительности в зависимости от наличия индексов.

В ходе выполнения экспериментально-исследовательской части было установлено, что при использовании индексов запросы к базе данных выполняются быстрее.

Список использованных источников

[1] Кузин А. В., Левонисова С. В. Базы данных. – 2005.

[2] Лазицкас Е. А., Загумённикова И. Н., Гилевский П. Г. Базы данных и системы управления базами данных. – 2016.

[3] OAT++ [Электронный ресурс] URL: https://oatpp.io (дата обращения: 30.05.2022).

[4] PostgreSQL [Электронный ресурс] URL: https://www.postgresql.org/ (дата обращения: 30.05.2022).

[5] SOCI [Электронный ресурс] URL: http://soci.sourceforge.net (дата обращения: 30.05.2022).

[6] QT [Электронный ресурс] URL: https://www.qt.io (дата обращения: 30.05.2022).

Приложения

Приложение 1. Презентация

Библиотечная система

Студент: Е. А. Варламова, ИУ7-61Б
Руководитель: К. Л. Тассов

Постановка задачи

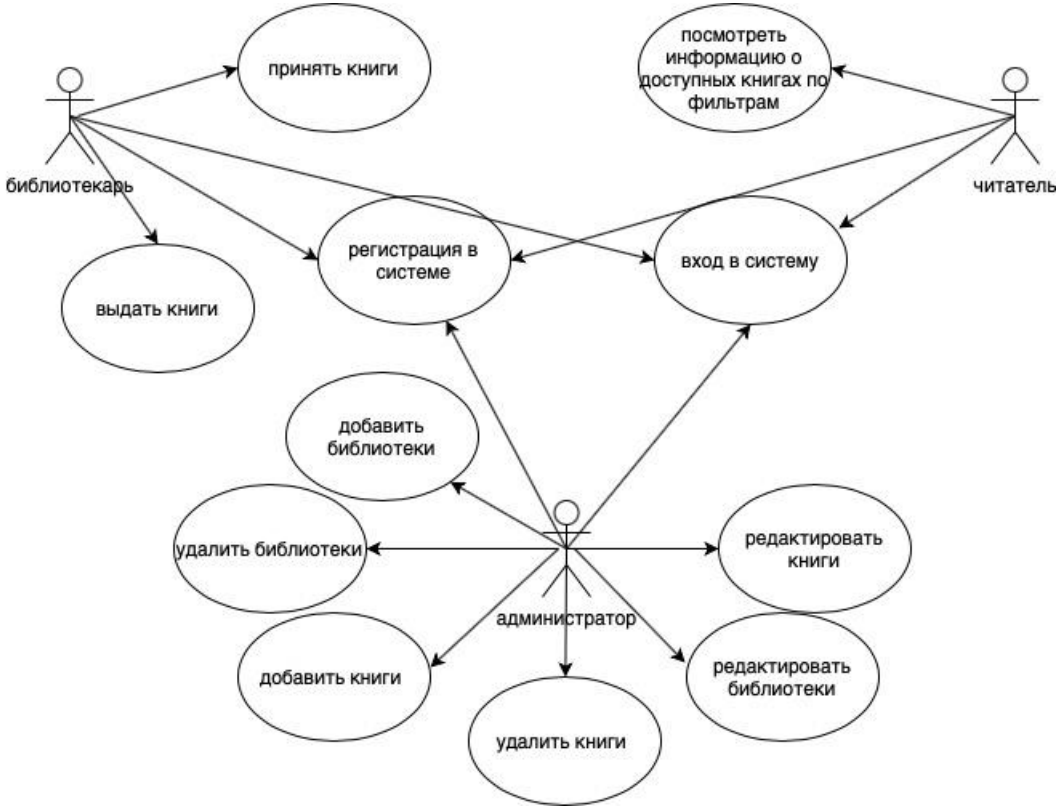
Задача: спроектировать и реализовать базу данных, содержащую данные о библиотечной системе. Разработать программный интерфейс приложения, который позволит:

- администратору библиотеки – добавлять, удалять и редактировать информацию о книгах;
- библиотекарю - выдавать книги читателям;
- читателю - получать информацию о наличии книг в разных библиотеках и о книгах, выданных ему.

Обзор существующих решений

ЭБС	Возможность использования в обычных библиотеках	Возможность разнообразной фильтрации	Возможность использования большой аудиторией
Лань	-	+	+
ЭБС Москвы	+	-	-
eLibrary	-	+	+

USE-CASE диаграмма

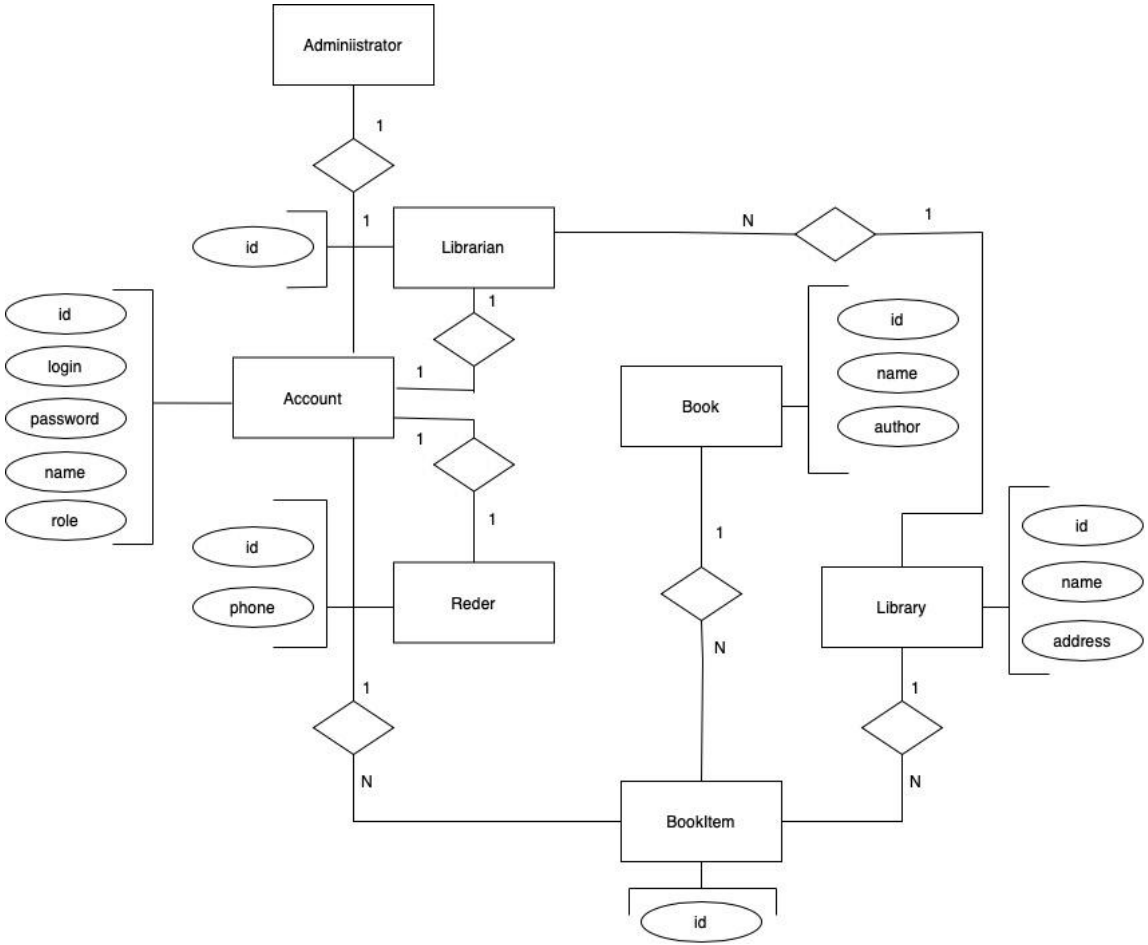


Модель хранения данных

Для реализации библиотечной системы было решено использовать построчное хранение данных, так как:

- предполагается большое количество коротких транзакций;
- предполагается высокий уровень отзывчивости системы;
- не предполагается выполнение сложных аналитических запросов.

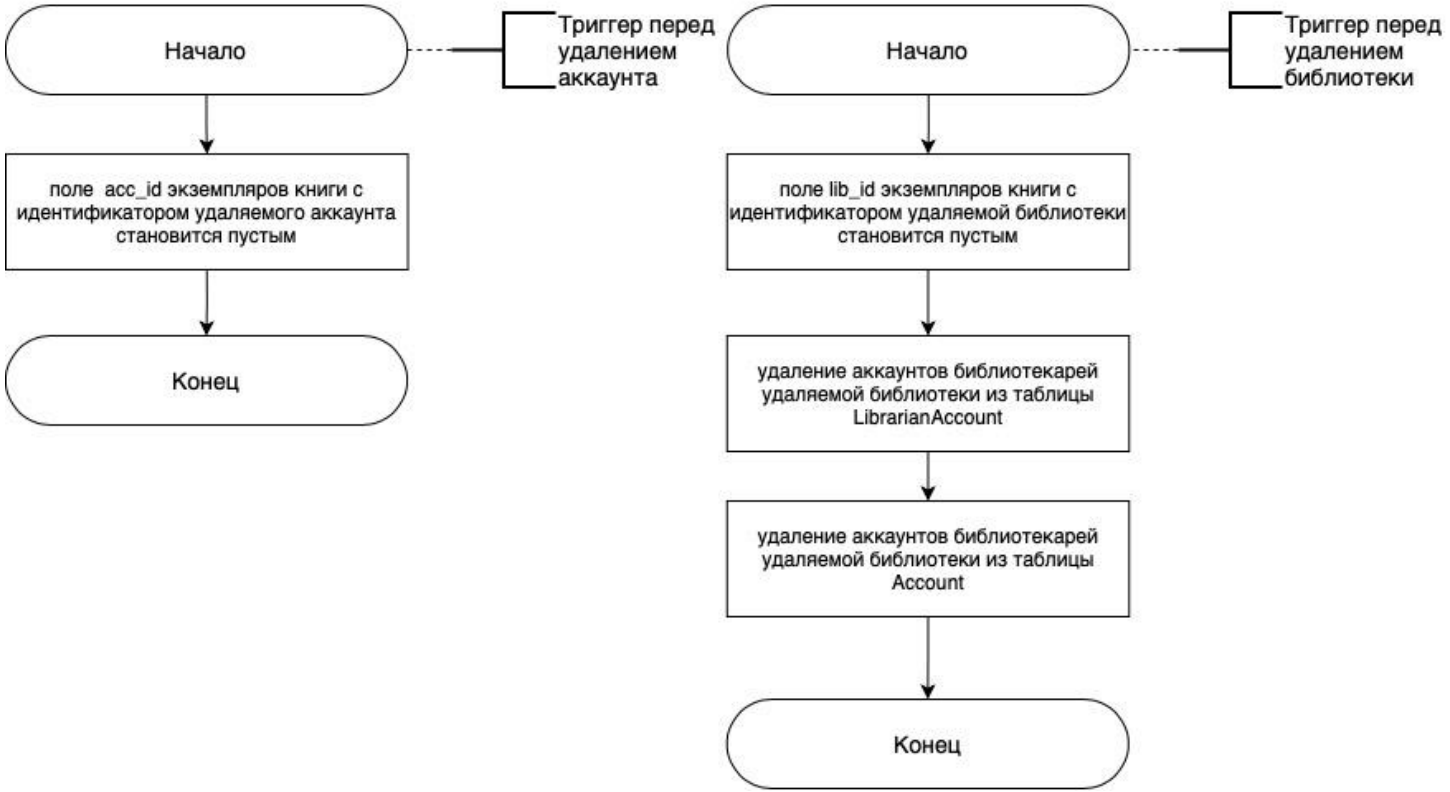
ER-модель


Adminiistrator
1
1
id
Librarian
N
1
id
login
password
name
role
1
Account
1
1
1
Book
id
name
author
id
phone
Reder
1
N
Library
id
name
address
1
N
1
N
BookItem
id

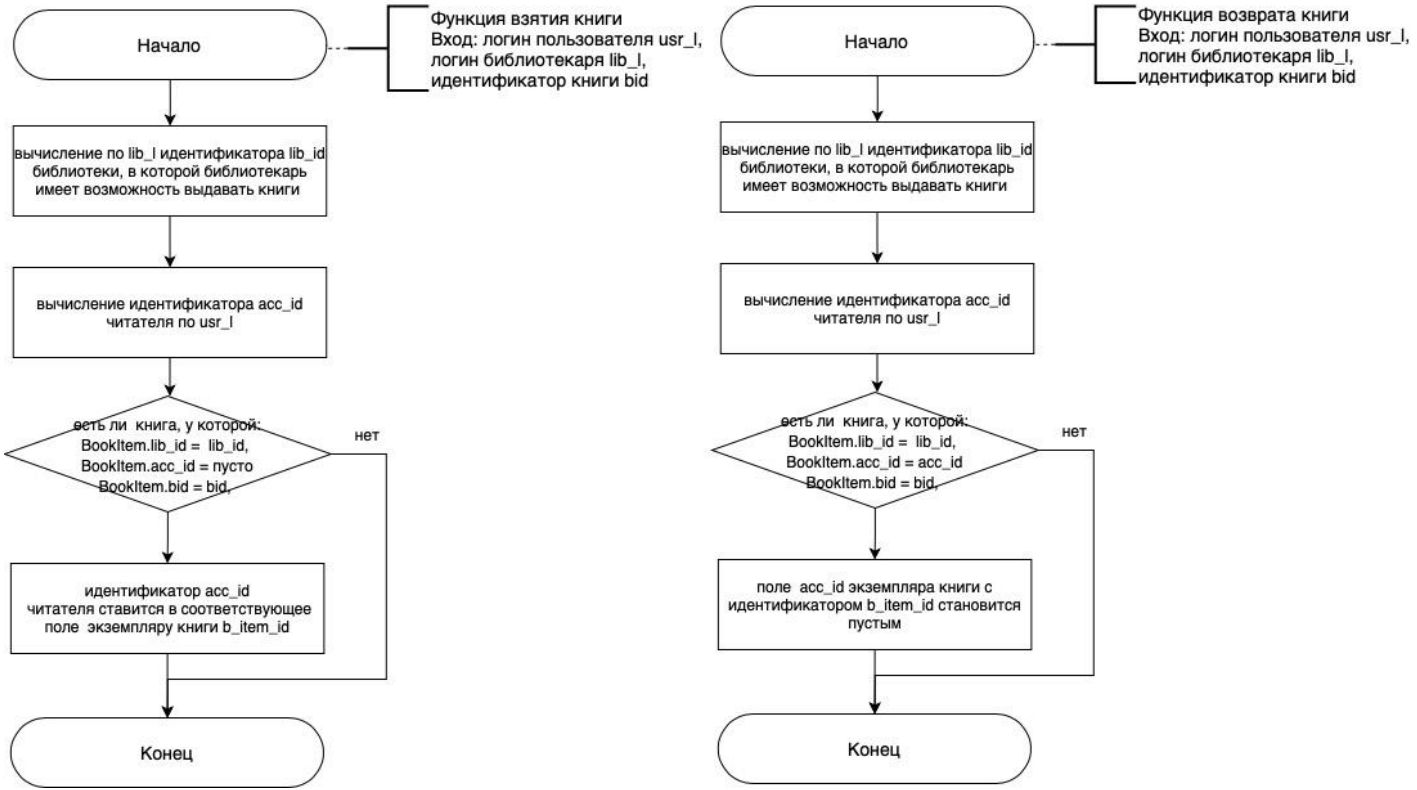
Ролевая модель базы данных

- администраторы, которые имеют полный доступ ко всем объектам базы данных;
- библиотекари, которые имеют права на обновление таблицы экземпляров книг и на чтение всех остальных таблиц;
- читатели, которые имеют права только на чтение всех таблиц.

Реализованные триггеры



Реализованные функции



Экспериментальная часть

Эксперимент 1: создаем индекс для первичного ключа. Результат: время выполнения **не изменилось**.

Эксперимент 2: создаем индекс для поля, по которому проводим фильтрацию. Результат: время выполнения **уменьшилось в 2.2 раза**.

Эксперимент 3: создаем индекс для внешнего ключа. Результат: время выполнения **уменьшилось в 2.95 раза**.