МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 05 01 Информационные системы и технологии

Специализация Базы данных

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

«Реализация базы данных библиотеки университета с применением технологии резервного копирования и восстановления»

Выполнил студент С.С. Жук

(Ф.И.О.)

Руководитель проекта асс. Н.И. Уласевич

(учен. степень, звание, должность, Ф.И.О., подпись)

Консультант: асс. Н.И. Уласевич.

(учен. степень, звание, должность, Ф.И.О., подпись)

Нормоконтролер: асс. Н.И. Уласевич.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовой проект защищен с оценкой

Содержание

[Введение 5](#_Toc167096900)

[1 Аналитический обзор литературы по теме проекта 6](#_Toc167096901)

[1.1. Постановка задачи 6](#_Toc167096902)

[1.1.1 Аналог Yale Library 7](#_Toc167096903)

[1.1.2 Аналог Universiteit Leiden 8](#_Toc167096904)

[1.2. Определение вариантов использования 9](#_Toc167096905)

[1.3 Вывод 10](#_Toc167096906)

[2 Проектирование и разработка базы данных 11](#_Toc167096907)

[2.1 UML схема базы данных 11](#_Toc167096908)

[2.2 Описание информационных объектов и ограничений целостности 12](#_Toc167096909)

[2.3 Вывод 13](#_Toc167096910)

[3 Разработка объектов базы данных 14](#_Toc167096911)

[3.1 Разработка таблиц базы данных 14](#_Toc167096912)

[3.2 Разработка представлений базы данных 17](#_Toc167096913)

[3.3 Разработка индексов базы данных 18](#_Toc167096914)

[3.4 Разработка функций базы данных 18](#_Toc167096915)

[3.5 Разработка процедур базы данных 19](#_Toc167096916)

[3.6 Разработка триггеров базы данных 20](#_Toc167096917)

[3.7 Разработка пользователей 21](#_Toc167096918)

[3.8 Описание технологии и применение в базе данных 21](#_Toc167096919)

[3.9 Вывод 22](#_Toc167096920)

[4. Описание процедур импорта и экспорта 23](#_Toc167096921)

[4.1 Экспорт данных в JSON 23](#_Toc167096922)

[4.2 Импорт данных из JSON 24](#_Toc167096923)

[5. Тестирование производительности 25](#_Toc167096924)

[6. Руководство пользователя 28](#_Toc167096925)

[Заключение 33](#_Toc167096926)

[Список используемых источников 34](#_Toc167096927)

[Приложение А 35](#_Toc167096928)

[Приложение Б 39](#_Toc167096929)

[Приложение В 42](#_Toc167096930)

[Приложение Г 47](#_Toc167096931)

[Приложение Д 48](#_Toc167096932)

[Приложение Е 50](#_Toc167096933)

# Введение

Информативные технологические процессы делаются обязательной составляющей существования трех тысячелетий. Современные читатели приобретают высококлассные знания и осознают их личностно, благодаря доступу к нестандартным и информативным ресурсам. По этой причине углубленное использование информационных технологий для изучения библиотечных ресурсов значительно расширяет возможности их применения в саморазвитии, самообучении, творческой и научно-исследовательской деятельности. Читатели всех уровней и категорий в библиотеках отмечают присутствие автоматизированных библиотечных концепций. Библиотеки, которые внедрили автоматизацию, оснащаются зарубежной компьютерной техникой, исходя из ее универсальности. Однако принципы работы иностранных библиотек не всегда полностью соответствуют этой концепции, поэтому для их адаптации необходимо разработать соответствующие решения на муниципальном уровне, а затем реализовать их на государственном уровне.

Библиографическая деятельность осуществляется в библиотеках всех типов, независимо от состава читателей и профиля работы. Она заключается в подборе нужной литературы для читателя. Так как мы живем в XXI веке, то объем литературы непрерывно растет. Библиотекарям все сложнее и сложнее найти быстро нужную книгу или журнал. Для этого необходимо создать базу данных, с помощью которой будет легко осуществляться поиск нужной книги, читателя и не только. Так же в ней можно отражать всех пользователей, сроки возврата книг, количество экземпляров книг. Создание базы данных возникает из-за необходимости повышения качества и эффективности библиографической работы.

Курсовому проекту поставлена задача создания информационной системы библиотеки университета для работы в ней как обычному пользователю, читателю, так библиотекарю и администратору самой библиотеки.

Для создания информационной базы данных будет использоваться СУБД «MSSQL», а для создания приложения – HTML, CSS и серверный язык PHP.

Для обеспечения функциональности приложения используются технология резервного копирования и восстановления базы данных.

# 1 Аналитический обзор литературы по теме проекта

## 1.1. Постановка задачи

Одним из важных этапов в создании базы данных для университетской библиотеки с использованием технологии резервного копирования и восстановления является анализ уже существующих решений на рынке. Этот анализ позволяет определить, какие системы уже применяются в других учебных заведениях, выявить их преимущества и недостатки, а также определить наиболее эффективные направления для дальнейшего совершенствования.

Применительно к университетской библиотеке, база данных играет ключевую роль в управлении книжным фондом, доступе к информационным ресурсам и обслуживании пользователей. Технология резервного копирования и восстановления обеспечивает защиту данных от потери, повреждения или несанкционированного доступа, что является критически важным для сохранности информации о книгах и читателях.

На сегодняшний день существует множество решений для создания баз данных с использованием технологии резервного копирования и восстановления. Однако, не все из них обладают необходимым уровнем функционала и безопасности. Это создает потребность в новых разработках в этой области, которые могли бы предложить полноценный комплекс инструментов для управления данными библиотеки университета, сочетающий в себе удобство, гибкость и высокие стандарты безопасности.

Задача проекта: разработать архитектуру приложения, взаимодействие с которой будет понятно любому пользователю. Построить базу данных и выполнить тестирование готового продукта.

Функционально должны быть выполнены следующие задачи:

* Определение ролей (администратор, библиотекарь, читатель);
* Просмотр информации о пользователях;
* Просмотр списка книг и наличие их в библиотеке;
* Взятие книг в аренду;
* Добавление экземпляров книг;
* Удаление книг из библиотеки;
* Выдача и прием книг обратно.

Должны быть выполнены следующие требования:

* База данных должна быть спроектирована в СУБД «MSSQL»;
* Доступ к данным должен осуществляться только через соответствующие процедуры;
* Должен быть проведен импорт данных из JSON файлов, экспорт данных в формат JSON;
* Необходимо протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости. Необходимо проанализировать планы запросов к таблице;
* Применить технологию резервного копирования и восстановления в базе данных.

### 1.1.1 Аналог Yale Library

Yale Library [1] – это цифровой архив и интеллектуальный центр, находящийся в сердце университетского сообщества Йельского университета. Данный веб-сайт служит важным ресурсом для студентов, исследователей и всех, кто интересуется образованием и знанием. Пример интерфейса данного сайта представлен на рисунке 1.1.



Рисунок 1.1 – Интерфейс сайта Yale Library

Одной из ключевых функций сайта является возможность поиска информации по различным критериям, включая автора, заголовок, ключевые слова и тематическое описание, что делает процесс поиска информации более удобным и эффективным. Пользователи могут также фильтровать результаты поиска по различным параметрам, таким как тип ресурса, год издания и доступность онлайн или в библиотеке.

Кроме того, на сайте предоставляются разнообразные услуги поддержки, такие как консультации библиотекарей, помощь в поиске информации и обучающие материалы по использованию библиотечных ресурсов. Это способствует повышению информационной грамотности пользователей и помогает им успешно осуществлять свои исследования и обучение.

Сайт также функционирует как платформа для социального взаимодействия среди пользователей. Они могут подписываться на обновления других пользователей, обмениваться рекомендациями, а также обсуждать и совместно использовать ресурсы.

Важно отметить, что для обеспечения эффективной работы сайта используются различные базы данных, включая системы управления базами данных для хранения информации о ресурсах, пользователях, истории поисковых запросов и других параметров. Эти базы данных обеспечивают быстрый доступ к информации и обработку данных для предоставления персонализированных услуг и улучшения пользовательского опыта.

### 1.1.2 Аналог Universiteit Leiden

Universiteit Leiden [2] – важный онлайн-ресурс для доступа к обширной коллекции библиотеки университета Лейдена. Этот сайт обеспечивает удобный и эффективный поиск по всей доступной коллекции, позволяя пользователям быстро находить нужные материалы. Пример интерфейса данного сайта представлен на рисунке 1.2.

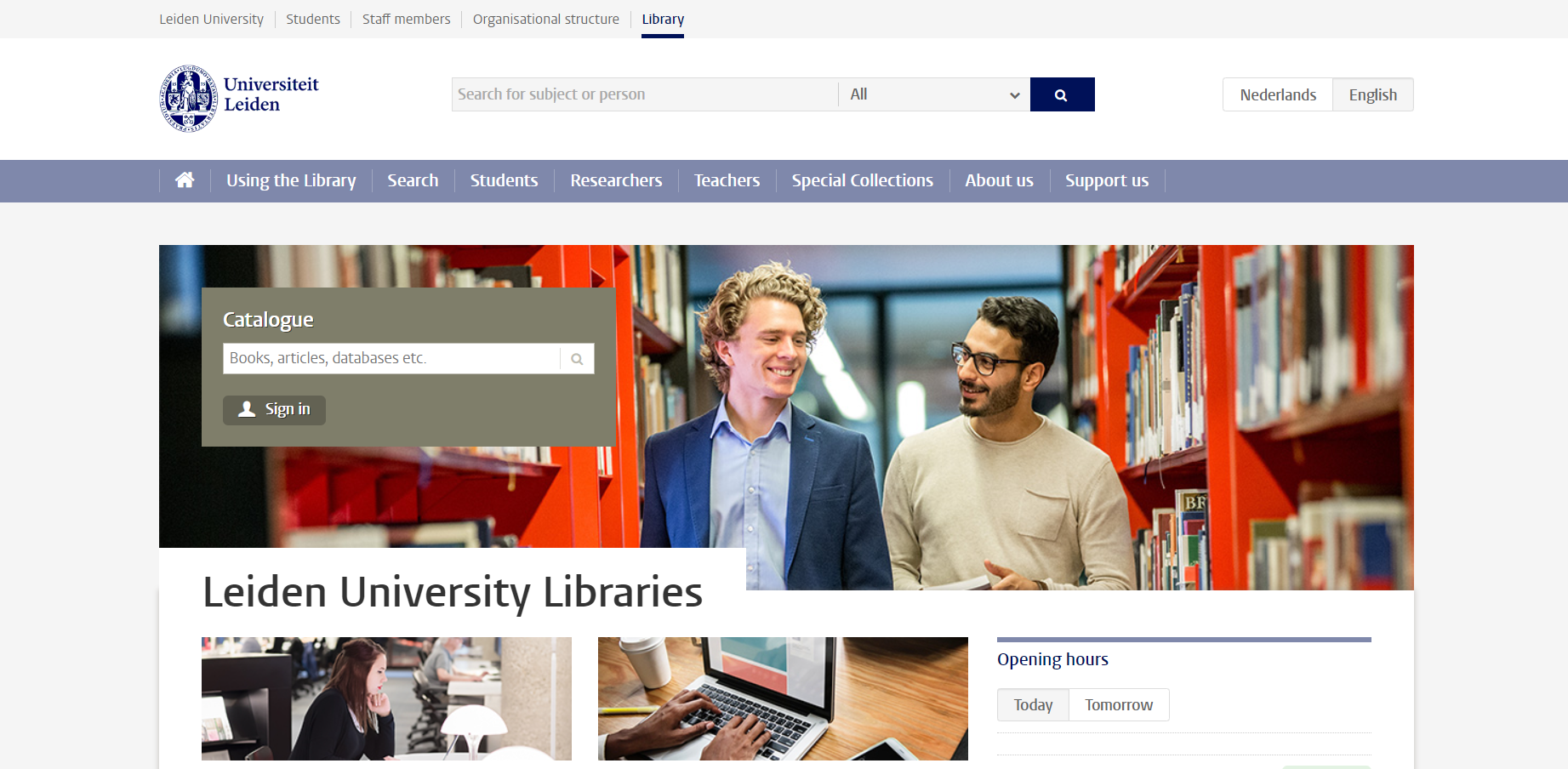


Рисунок 1.2 – Интерфейс сайта Universiteit Leiden

Сайт библиотеки Universiteit Leiden предоставляет обширный функционал, связанный с базой данных. В центре этого функционала находится организация и доступ к богатой коллекции ресурсов, включающей книги, статьи, журналы, электронные ресурсы и многие другие источники. База данных библиотеки обеспечивает удобный поиск по всем имеющимся материалам, позволяя пользователям быстро находить нужные ресурсы по ключевым словам, авторам, заголовкам и другим критериям.

Кроме того, сайт предоставляет возможность управления библиотечным аккаунтом, включая возможность регистрации, заказа и продления книг, а также просмотра истории выдачи и возврата материалов. Пользователи могут также оставлять отзывы и оценки книгам, делиться своими впечатлениями с другими читателями.

Функционал сайта также включает в себя доступ к электронным ресурсам, таким как электронные версии книг и журналов, которые могут быть скачаны или прочитаны онлайн. Пользователи могут оформлять заказы на доставку книг из коллекции библиотеки, а также получать доступ к электронным материалам удаленно через удаленный доступ.

Важным компонентом функционала является также поддержка и консультации библиотечных специалистов, которые могут помочь пользователям с поиском информации, подбором ресурсов и ответами на вопросы. Это создает уникальную возможность для студентов, преподавателей и исследователей получить поддержку и помощь в своей учебе и исследовательской деятельности.

## 1.2. Определение вариантов использования

Основной точкой старта при разработке функциональных требований для базы данных является определение способа обработки данных и обеспечение необходимой функциональности пользовательскому интерфейсу. Помимо этого, важно также учитывать роли пользователей и их сценарии использования системы. Сценарии использования описывают, как пользователи будут взаимодействовать с системой в зависимости от своих ролей. Это помогает определить, какие функции должны быть доступны для каждой роли, какие данные доступны, а также как должна быть организована навигация. Обычно сценарии использования представляются в виде диаграмм Use Case, которые наглядно отображают взаимодействие между пользователями и системой. На рисунке 1.3 представлена UML диаграмма вариантов использования.

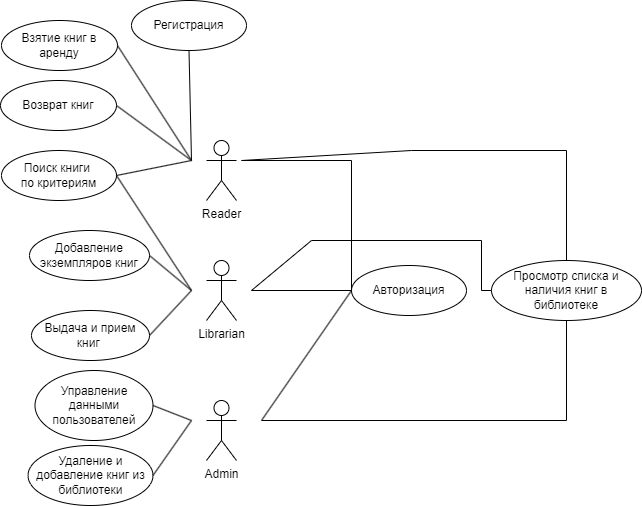


Рисунок 1.3 – UML диаграмма вариантов использования

Для того, чтобы начать пользоваться приложением, читателю нужно пойти регистрацию и авторизацию, а администратору и библиотекарю просто авторизироваться. После авторизации на сайте каждая роль получает свои привилегии.

Роль читателя (Reader) получает такие привилегии, как взятие книг в аренду, возврат книг, просмотр списка и наличия книг в библиотеке, поиск книги по критериям.

Роль библиотекаря (Librarian) предусматривает поиск книги по критериям, добавление экземпляров книг, выдача и прием книг, просмотр списка и наличия книг в библиотеке.

Роль администратора (Admin) заключается в управлении данными пользователей, просмотре списка и наличия книг в библиотеке, удалении и добавлении книг из библиотеки.

## 1.3 Вывод

Таким образом, был проведен тщательный аналитический обзор существующих библиотек в университетах для детального определения основных характеристик и функциональных возможностей, которые необходимо учитывать при разработке базы данных. В рамках этого исследования были выявлены ключевые аспекты, влияющие на эффективность и удобство использования библиотечных систем. Также были определены функциональные требования к этой базе данных.

Дополнительно, особое внимание было уделено определению ролей пользователей и их вариантов использования в зависимости от этих ролей. Для каждой категории пользователей были проанализированы типичные сценарии взаимодействия с системой, что позволило разработать детализированные требования к функциональности базы данных. Эти требования охватывают широкий спектр операций, начиная от поиска и выдачи книг, до управления библиотечными ресурсами и анализа пользовательской активности.

На основе полученных данных была разработана UML диаграмма, на которой наглядно показаны основные функции, доступные для каждой из ролей пользователей в контексте университетской библиотеки. Эта диаграмма служит важным инструментом для визуализации взаимодействий между пользователями и системой, а также для определения областей, требующих дальнейшего улучшения и оптимизации. В результате, созданная база данных будет не только соответствовать текущим потребностям библиотеки, но и обладать гибкостью для адаптации к будущим изменениям и расширениям.

# 2 Проектирование и разработка базы данных

## 2.1 UML схема базы данных

Схема базы данных представляет собой логическую конфигурацию либо целой реляционной базы данных, либо ее части. Схема может существовать как в виде наглядного представления базы данных, так и в виде набора формул (также именуемых «условиями целостности»), которые регулируют ее устройство. Эти формулы выражаются с помощью языка описания данных, например, SQL. Будучи частью словаря данных, схема показывает, как связаны между собой сущности, из которых состоит база данных (таблицы, представления, хранимые процедуры и так далее). Схема базы данных будет представлена на рисунке 2.1.

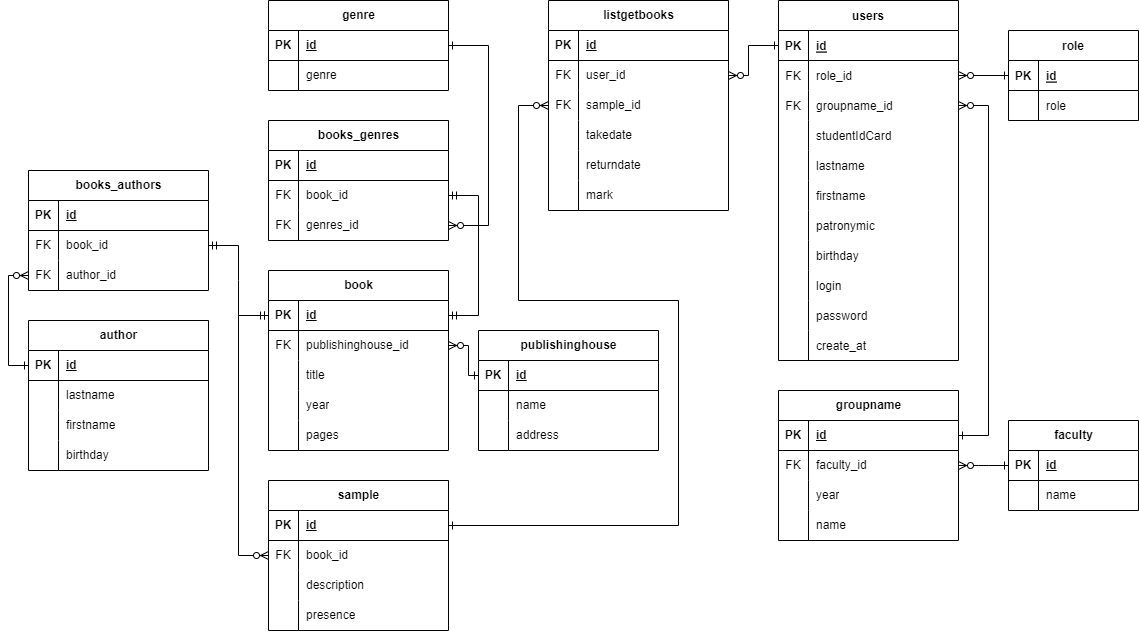


Рисунок 2.1 – Схема базы данных

Таким образом, схема показывает связи между таблицами и полями, а также отношения между ними. Одна из основных связей – это связь "один ко многим" (one-to-many), которая определяет отношение одной записи в таблице к нескольким записям в другой таблице.

Например, таблица users связана с таблицами role и groupname через внешние ключи role\_id и groupname\_id соответственно. Также видно, что таблицы book и sample связаны друг с другом через внешний и первичный ключ book\_id, а таблицы sample и listgetbooks – через sample\_id. Все связи представлены на рисунке 2.1, который располагается выше.

## 2.2 Описание информационных объектов и ограничений целостности

Поле первичного ключа служит уникальным определением записи. Оно также служит для связи таблиц. В связанных таблицах первичный ключ родительской таблицы становится внешним ключом в дочерней таблице. Внешний ключ дочерней таблицы отсылает к сведениям родительской таблицы.

Первичные ключи будут иметь постфикс PK, вторичные – FK. Также в подавляющем числе случаев названия вторичных ключей будут соответствовать названию связанных таблиц (без множественного числа), при несоответствии укажем название связанной таблицы во избежание путаницы.

Таблица 2.1 – Первичные и внешние ключи отношений

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Ключ | Описание |
| users | id | PK | Идентификационный номер таблицы «users» |
| role\_id | FK | Идентификационный номер таблицы «role» |
| groupname\_id | FK | Идентификационный номер таблицы «groupname» |
| studentIDCard |  | Номер студенческого билета |
| lastname |  | Фамилия пользователя |
| firstname |  | Имя пользователя |
| patronymic |  | Отчество пользователя |
| birthday |  | Дата рождения пользователя |
| login |  | Логин учетной записи |
| password |  | Пароль учетной записи |
| create\_at |  | Дата создания учетной записи |
| role | id | PK | Идентификационный номер таблицы «role» |
| role |  | Имя роли пользователя |
| groupname | id | PK | Идентификационный номер таблицы «groupname» |
| faculty\_id | FK | Идентификационный номер таблицы «faculty» |
| year |  | Год поступления группы в университет |
| name |  | Название группы |
| faculty | id | PK | Идентификационный номер таблицы «faculty» |
| name |  | Название факультета |
| listgetbooks | id | PK | Идентификационный номер таблицы «listgetbooks» |
| user\_id | FK | Идентификационный номер таблицы «user» |
| sample\_id | FK | Идентификационный номер таблицы «sample» |
| takedate |  | Дата получения книги |
| returndate |  | Дата возврата книги |
| mark |  | Оценка полученной книги |
| sample | id | PK | Идентификационный номер таблицы «sample» |
| book\_id | FK | Идентификационный номер таблицы «book» |

Окончание таблицы 2.1

|  |  |  |  |
| --- | --- | --- | --- |
| Таблица | Поле | Ключ | Описание |
| sample | description |  | Описание состояния экземпляра книги |
| presence |  | Наличие в библиотеке экземпляра книги |
| author | id | PK | Идентификационный номер таблицы «author» |
| lastname |  | Фамилия автора |
| firstname |  | Имя автора |
| birthday |  | День рождения автора |
| books\_authors | id | PK | Составной ключ полей book\_id и author\_id |
| book\_id | FK | Идентификационный номер таблицы «book» |
| author\_id | FK | Идентификационный номер таблицы «author» |
| book | id | PK | Идентификационный номер таблицы «book» |
| author\_id | FK | Идентификационный номер таблицы «author» |
| publishinghouse\_id | FK | Идентификационный номер таблицы «publishinghouse» |
| title |  | Название книги |
| year |  | Год написания книги |
| pages |  | Количество страниц книги |
| publishinghouse | id | PK | Идентификационный номер таблицы «publishinghouse» |
| name |  | Название издательства |
| address |  | Адрес издательства |
| genre | id | PK | Идентификационный номер таблицы «genre» |
| genre |  | Название жанра |
| books\_genres | id | PK | Составной ключ полей book\_id и genre\_id |
| book\_id | FK | Идентификационный номер таблицы «book» |
| genre\_id | FK | Идентификационный номер таблицы «genre» |

Таким образом, для реализации базы данных было разработано 12 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: role, users, groupname, faculty, listgetbooks, sample, book, books\_genres, books\_authors, authors, genres, publishinghouse. Ниже будет описание про каждую из них более подробно.

## 2.3 Вывод

Такая архитектура базы данных предоставляет гибкость в управлении данными, высокую производительность и безопасность. Структурированное разделение на авторизацию и доменную логику позволяет оптимально использовать ресурсы системы и обеспечивать эффективное взаимодействие с данными.

# 3 Разработка объектов базы данных

## 3.1 Разработка таблиц базы данных

Для реализации спроектированной базы данных была выбрана система управления базами данных «MySQL». Это обусловлено тем, что, во-первых, данная СУБД получилась широкое распространение, а во-вторых, имеются свободно распространяемые сборки.

Краткая характеристика каждой из таблиц была предоставлена во втором разделе, а код их создания можно увидеть в [приложении А](#_Приложение_А). Произведем описание структуру каждой из таблиц с описанием типа данных полей.

Таблица users хранит данные учетных записей. Ее структура приведена в таблице 3.1.

Таблица 3.1 – Столбцы таблицы users

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «users» |
| role\_id | INT | Идентификационный номер таблицы «role» |
| groupname\_id | INT | Идентификационный номер таблицы «groupname» |
| studentIDCard | INT | Номер студенческого билета |
| lastname | VARCHAR(255) | Фамилия пользователя |
| firstname | VARCHAR(255) | Имя пользователя |
| patronymic | VARCHAR(255) | Отчество пользователя |
| birthday | DATE | Дата рождения пользователя |
| login | VARCHAR(255) | Логин учетной записи |
| password | VARCHAR(255) | Пароль учетной записи |
| create\_at | DATETIME | Дата создания учетной записи |

Таблица roleсодержит основные роли, предоставленные для работы с приложением. Ее структура приведена в таблице 3.2.

Таблица 3.2 – Столбцы таблицы role

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «role» |
| role | VARCHAR(50) | Имя роли пользователя |

Таблица groupname содержит сведения о группах. Ее структура приведена в таблице 3.3.

Таблица 3.3 – Столбцы таблицы groupname

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «groupname» |
| faculty\_id | INT | Идентификационный номер таблицы «faculty» |
| year | INT | Год поступления группы в университет |
| name | VARCHAR(50) | Название группы |

Таблица faculty содержит сведения о факультетах. Ее структура приведена в таблице 3.4.

Таблица 3.4 – Столбцы таблицы faculty

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «faculty» |
| name | VARCHAR(255) | Название факультета |

Таблица listgetbooks содержит сведения о книгах, которые взяты пользователями. Ее структура приведена в таблице 3.5.

Таблица 3.5 – Столбцы таблицы listgetbooks

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «listgetbooks» |
| user\_id | INT | Идентификационный номер таблицы «user» |
| sample\_id | INT | Идентификационный номер таблицы «sample» |
| takedate | DATE | Дата получения книги |
| returndate | DATE | Дата возврата книги |
| mark | INT | Оценка полученной книги |

Таблица sample содержит сведения о экземплярах книг библиотеки. Ее структура приведена в таблице 3.6.

Таблица 3.6 – Столбцы таблицы sample

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «sample» |
| book\_id | INT | Идентификационный номер таблицы «book» |
| description | VARCHAR(255) | Описание состояния экземпляра книги |
| presence | TINYINT | Наличие в библиотеке экземпляра книги |

Таблица author содержит сведения об авторах книг. Ее структура приведена в таблице 3.7.

Таблица 3.7 – Столбцы таблицы author

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «author» |
| lastname | VARCHAR(255) | Фамилия автора |
| firstname | VARCHAR(255) | Имя автора |
| birthday | DATE | День рождения автора |

Таблица book содержит сведения о книгах. Ее структура приведена в таблице 3.8.

Таблица 3.8 – Столбцы таблицы book

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «book» |

Окончание таблицы 3.8

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| publishinghouse\_id | INT | Идентификационный номер таблицы «publishinghouse» |
| title | VARCHAR(255) | Название книги |
| year | INT | Год написания книги |
| pages | SMALLINT | Количество страниц книги |

Таблица publishinghouse содержит сведения о издательствах. Ее структура приведена в таблице 3.9.

Таблица 3.9 – Столбцы таблицы publishinghouse

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «publishinghouse» |
| name | VARCHAR(255) | Название издательства |
| address | VARCHAR(255) | Адрес издательства |

Таблица genre содержит сведения о жанрах книги. Ее структура приведена в таблице 3.10.

Таблица 3.10 – Столбцы таблицы genre

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Идентификационный номер таблицы «genre» |
| genre | VARCHAR(255) | Название жанра |

Таблица books\_genres предоставляет хранение несколько жанров книги. Ее структура приведена в таблице 3.11.

Таблица 3.11 – Столбцы таблицы books\_genres

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Составной ключ полей book\_id и genre\_id |
| book\_id | INT | Идентификационный номер таблицы «book» |
| genre\_id | INT | Идентификационный номер таблицы «genre» |

Таблица books\_authors предоставляет хранение несколько авторов книги. Ее структура приведена в таблице 3.12.

Таблица 3.12 – Столбцы таблицы books\_authors

|  |  |  |
| --- | --- | --- |
| Название | Тип данных | Описание |
| id | INT | Составной ключ полей book\_id и author\_id |
| book\_id | INT | Идентификационный номер таблицы «book» |
| author\_id | INT | Идентификационный номер таблицы «author» |

Перед созданием этих таблиц для лучшей структуры было принято решение использовать файловые группы.

Файловые группы– это механизм, который позволяет логически организовать файлы базы данных и распределять объекты базы данных, такие как таблицы и индексы, по различным физическим дискам или устройствам. Это может улучшить производительность и управляемость базы данных, особенно в случае больших объемов данных. Скрипт для создания файловых групп будет также представлен в [приложении А](#_Приложение_А).

## 3.2 Разработка представлений базы данных

Представление в базе данных – это виртуальная таблица, которая формируется на основе выполнения запроса к одной или нескольким таблицам в базе данных. Представления позволяют объединять данные из нескольких таблиц в одну, не изменяя исходных таблиц и их структуры. Представления используются для обеспечения удобства доступа и управления данными в базе данных, а также для улучшения производительности запросов. В данном проекте были созданы четыре представления:

* ReaderDetails содержит информацию о читателях;
* LibrarianDetails содержит информацию о библиотекарях;
* BookDetails содержит информацию о списках книг с выводом авторов и жанров.
* AvailableBooks содержит информацию о списках книг, которые находятся в наличии.

Реализация нескольких из представлений представлено в листинге 3.1. Остальные представления будут аналогичны, работая с другими данными.

|  |
| --- |
| CREATE VIEW ReaderDetails AS  SELECT u.id AS user\_id, u.lastname, u.firstname, u.patronymic, u.birthday,  r.role, g.name AS group\_name, f.name AS faculty\_name  FROM users u  INNER JOIN role r ON u.role\_id = r.id  LEFT JOIN groupname g ON u.groupname\_id = g.id  LEFT JOIN faculty f ON g.faculty\_id = f.id  WHERE u.role\_id = 3;  CREATE VIEW BookDetails AS  SELECT b.id AS book\_id, b.title, b.year, b.pages,  p.name AS publishing\_house,  a.lastname AS author\_lastname, a.firstname AS author\_firstname,  STRING\_AGG(g.genre, ', ') AS genres  FROM book b  INNER JOIN publishinghouse p ON b.publishinghouse\_id = p.id  LEFT JOIN books\_authors ba ON b.id = ba.book\_id  LEFT JOIN author a ON ba.author\_id = a.id  LEFT JOIN books\_genres bg ON b.id = bg.book\_id  LEFT JOIN genre g ON bg.genre\_id = g.id  GROUP BY b.id, b.title, b.year, b.pages, p.name, a.lastname, a.firstname; |

Листинг 3.1 – Представления ReaderDetails и BookDetails

Были созданы основные представления по доступу к данным таблиц, с последующей манипуляцией выборки.

## 3.3 Разработка индексов базы данных

Индекс в базе данных представляет собой объект, который используется для ускорения поиска данных. Если таблица содержит большое количество строк, то последовательный поиск данных может занимать много времени. Индекс создается на основе значений одного или нескольких столбцов таблицы и указывает на соответствующие строки таблицы. Использование индексов помогает улучшить производительность базы данных, поскольку они имеют оптимизированную структуру для поиска, например, сбалансированное дерево. Один из наиболее распространенных алгоритмов индексации – это B-дерево (B-tree). B-дерево – это сбалансированное дерево поиска, которое обеспечивает эффективный поиск элементов, используя ключи для сортировки данных в индексе.

Например, для того, чтобы быстро находить пользователей по их параметрам, были созданы два индекса: idx\_users\_lastname\_firstname по полям lastname и firstname и idx\_users\_studentIDCard по полю studentIDCard. Создание индексов представлено в листинге 3.2.

|  |
| --- |
| CREATE INDEX idx\_faculty\_name ON faculty (name);  CREATE INDEX idx\_groupname\_name ON groupname (name);  CREATE INDEX idx\_role\_role ON role (role);  CREATE INDEX idx\_users\_lastname\_firstname ON users(lastname, firstname);  CREATE INDEX idx\_users\_studentIDCard ON users(studentIDCard);  CREATE INDEX idx\_book\_title ON book(title);  CREATE INDEX idx\_publishinghouse\_name ON publishinghouse(name);  CREATE INDEX idx\_author\_lastname\_firstname ON author(lastname, firstname);  CREATE INDEX idx\_genre\_genre ON genre(genre); |

Листинг 3.2 – Индексы таблиц

Кроме этого, в базе данных присутствуют индексы в таблицах: groupname, faculty, role, book, genre, author и publishinghouse.

Применение индексов в базе данных значительно повышает скорость выполнения операций поиска, сортировки и фильтрации данных, особенно в случае больших объемов информации. Однако создание индексов может занять дополнительное время при добавлении или изменении данных в таблицах. Поэтому необходимо балансировать количество и типы индексов, чтобы обеспечить оптимальную производительность базы данных.

## 3.4 Разработка функций базы данных

Для управления данными через приложение все пользователи используют функции. Функция – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. Стоит отметить, что функции не могут содержать группы операторов DML и DDL. В моем приложении реализовано 11 функций. Пример создания функции представлен на листинге 3.3.

|  |
| --- |
| CREATE FUNCTION SearchReaderByStudentIDCard(@studentIDcard VARCHAR(6))  RETURNS TABLE  AS  RETURN (  SELECT  u.id,  r.role,  u.groupname\_id,  u.studentIDcard,  u.lastname,  u.firstname,  u.patronymic,  u.birthday,  u.create\_at  FROM users u  INNER JOIN role r ON u.role\_id = r.id  WHERE u.role\_id = 3 AND u.studentIDcard = @studentIDcard  ); |

Листинг 3.3 – Функция для поиска по студенческой карте

Функция SearchReaderByStudentIDCard() выполняет запрос к базе данных для извлечения информации о читателе по номеру студенческого билета. Запрос объединяет данные из таблиц users и role с использованием оператора INNER JOIN. В результате выполнения SELECT-запроса выбираются следующие колонки: идентификатор пользователя, роль пользователя, идентификатор группы, номер студенческого билета, фамилия, имя, отчество, дата рождения и дата создания записи.

Таким образом, функция SearchReaderByStudentIDCard() предназначена для получения детальной информации о читателе, имеющем определенный номер студенческого билета, из базы данных. Выборка из таблиц реализована с помощью функций, код приведен в [приложении Б](#_Приложение_Б).

## 3.5 Разработка процедур базы данных

По большому счету, процедуры выполняют аналогичные действия с функциями, за тем исключением, что они не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Всего было разработано 16 процедур. В листинге 3.4 представлен пример реализации одной из процедур.

|  |
| --- |
| CREATE PROCEDURE AddBookSample  @book\_id int,  @description varchar(255)  AS  BEGIN  BEGIN TRY  IF NOT EXISTS (SELECT 1 FROM book WHERE id = @book\_id)  BEGIN  THROW 50007, 'Книга с указанным ID не существует', 1;  RETURN;  END  INSERT INTO sample (book\_id, description, presence)  VALUES (@book\_id, @description, 1);  PRINT 'Экземпляр книги успешно добавлен';  END TRY  BEGIN CATCH  PRINT 'При добавлении экземпляра книги произошла ошибка: ' + ERROR\_MESSAGE();  PRINT 'Код ошибки: ' + CAST(ERROR\_NUMBER() AS VARCHAR(10));  PRINT 'Уровень ошибки: ' + CAST(ERROR\_SEVERITY() AS VARCHAR(10));  PRINT 'Строка, на которой произошла ошибка: ' + CAST(ERROR\_LINE() AS VARCHAR(10));  PRINT 'Процедура, в которой произошла ошибка: ' + ERROR\_PROCEDURE();  END CATCH  END; |

Листинг 3.4 – Процедура добавления экземпляра книги

Данная процедура добавляет экземпляр книги в библиотеку, в качестве параметров принимает id книги и описание экземпляра. Скрипт других процедур приведен в [приложении В](#_Приложение_В).

## 3.6 Разработка триггеров базы данных

Триггер в базе данных – это объект, который автоматически выполняет определенные действия при возникновении определенных событий в таблице или представлении базы данных. Триггер может быть настроен на срабатывание при вставке, обновлении или удалении строк в таблице.

Триггеры используются для поддержки целостности данных, контроля доступа к данным и автоматической обработки данных при выполнении определенных операций в таблице.

Всего было сделано 8 триггеров, в листинге 3.5 приведен код одного из триггеров.

|  |
| --- |
| CREATE TRIGGER trg\_check\_age  ON users  AFTER INSERT, UPDATE  AS  BEGIN  DECLARE @TodayDate DATETIME  SET @TodayDate = GETDATE()  IF EXISTS (  SELECT 1  FROM inserted  WHERE DATEDIFF(YEAR, inserted.birthday, @TodayDate) <= 16  )  BEGIN  THROW 50001, 'Возраст пользователя должен быть больше 16 лет.', 1  ROLLBACK TRANSACTION  END  END; |

Листинг 3.5 – Скрипт триггера trg\_check\_age

Данный триггер создается в таблице users, который проверяет дату, а именно проверка на то, что возраст пользователя должен быть больше 16 лет. Скрипт других триггеров представлен в [приложении Г](#_Приложение_Г).

## 3.7 Разработка пользователей

Пользователь базы данных — это физическое или юридическое лицо, которое имеет доступ к БД и пользуется услугами информационной системы для получения информации. На каждом этапе развития БД (проектирование, реализация, эксплуатация, модернизация и развитие, полная реорганизация) с ней связаны разные категории пользователей.

При проектировании базы данных было создано 3 типа пользователей: читатель, библиотекарь и администратор библиотеки. Создание пользователей представлено в листинге 3.6.

|  |
| --- |
| CREATE LOGIN ReaderUser WITH PASSWORD = '321';  CREATE USER ReaderUser FOR LOGIN ReaderUser;  CREATE LOGIN AdminUser WITH PASSWORD = '111';  CREATE USER AdminUser FOR LOGIN AdminUser;  CREATE LOGIN LibrarianUser WITH PASSWORD = '123';  CREATE USER LibrarianUser FOR LOGIN LibrarianUser; |

Листинг 3.6 – Создание пользователей

Далее для созданного пользователя базы данных были созданы роли и выданы необходимые привилегии, код предоставлен в [приложении Д.](#_Приложение_Д_2)

## 3.8 Описание технологии и применение в базе данных

Резервное копирование и восстановление в базах данных представляет собой систему методов и инструментов, предназначенных для сохранения и восстановления данных в случае их потери или повреждения. В данной базе данных используется технология резервного копирования для обеспечения надежности и безопасности хранения информации.

Применение резервного копирования и восстановления обладает несколькими ключевыми преимуществами:

* 1. Сохранение целостности данных. Регулярные резервные копии помогают предотвратить потерю информации в случае аварийных ситуаций, таких как сбои оборудования или программного обеспечения.
  2. Обеспечение безопасности. Резервные копии данных могут служить важным инструментом в защите от угроз безопасности, таких как вирусы или злонамеренные атаки, позволяя быстро восстановить информацию после инцидента.
  3. Восстановление в случае необходимости. Благодаря наличию резервных копий базы данных возможно оперативное восстановление данных до момента их утраты или повреждения.
  4. Эффективное управление данными. Возможность регулярного создания резервных копий и их последующего использования дает возможность контролировать и управлять данными с большей уверенностью.

Технология резервного копирования позволяет обеспечить сохранность и доступность данных, играя ключевую роль в обеспечении непрерывности работы базы данных и предоставлении средств для восстановления информации в случае необходимости.

## 3.9 Вывод

В данном разделе была рассмотрена разработка объектов базы данных библиотеки университета. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе. Благодаря хорошо спроектированным объектам базы данных система способна обеспечивать надежность, своевременное быстродействие и целостность данных.

# 4. Описание процедур импорта и экспорта

## 4.1 Экспорт данных в JSON

Для всех таблиц базы данных были разработаны процедуры экспорта и импорта данных в формате JSON. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в JSON, была разработана процедура, результатом которой стало создание JSON файла, с записанными строками из определенной таблицы в JSON формат. Например, скрипт процедуры для экспорта таблицы author представлен в листинге 4.1.

|  |
| --- |
| CREATE OR ALTER PROCEDURE ExportAuthorDataToJson  @filePath NVARCHAR(500)  AS  BEGIN  DECLARE @jsonOutput NVARCHAR(MAX);  DECLARE @cmd NVARCHAR(4000);  DECLARE @psCmd NVARCHAR(4000);  SELECT @jsonOutput = (  SELECT \*  FROM author  FOR JSON AUTO, ROOT('Author')  );  SET @jsonOutput = REPLACE(@jsonOutput, '"', '\"');  SET @psCmd = 'powershell.exe -Command "$jsonContent = ''' + @jsonOutput + '''; $filePath = ''' + @filePath + '''; $jsonContent | Out-File -FilePath $filePath -Encoding default"';  IF LEN(@psCmd) > 2048  BEGIN  RAISERROR('Длина команды превышает допустимый предел в 2048 символов', 16, 1);  RETURN;  END  EXEC xp\_cmdshell @psCmd;  END; |

Листинг 4.1 – Процедура экспорта ExportAuthorDataToJson

Для каждой таблицы вызывается процедура с соответствующим запросом и именем файла, чтобы экспортировать данные этой таблицы в отдельный JSON-файл. Кроме таблицы author, эти процедуры экспортируют данные из таблиц role, users, groupname, faculty, listgetbooks, sample, book, genre, publishinghouse, books\_genres, books\_authors.

## 4.2 Импорт данных из JSON

Для импорта JSON-данных в базу данных MSSQL можно воспользоваться процедурой ImportAuthorDataFromJson которая предназначена для импорта данных об авторах. Вначале создается временная таблица #TempAuthor для временного хранения данных авторов. Содержимое файла author.json загружается с помощью OPENROWSET и сохраняется в переменной @jsonInput. Данные из JSON-файла извлекаются с помощью функции OPENJSON и вставляются во временную таблицу #TempAuthor. Наконец, с помощью оператора MERGE данные из временной таблицы #TempAuthor объединяются с таблицей author, добавляя новые записи о авторах. После завершения операции временная таблица #TempAuthor удаляется.

Процедура для импорта информации об авторах представлена в листинге 4.2.

|  |
| --- |
| CREATE OR ALTER PROCEDURE ImportAuthorDataFromJson  AS  BEGIN  CREATE TABLE #TempAuthor (  lastname VARCHAR(255) NOT NULL,  firstname VARCHAR(255) NOT NULL,  birthday DATE  );  DECLARE @jsonInput NVARCHAR(MAX);  SELECT @jsonInput = BulkColumn  FROM OPENROWSET (BULK 'C:\localhost\export and import\author.json', SINGLE\_CLOB) AS j;  INSERT INTO #TempAuthor (lastname, firstname, birthday)  SELECT lastname, firstname, birthday  FROM OPENJSON(@jsonInput, '$.Author')  WITH (  lastname VARCHAR(255) '$.lastname',  firstname VARCHAR(255) '$.firstname',  birthday DATE '$.birthday'  );  MERGE author AS target  USING #TempAuthor AS source  ON target.lastname = source.lastname AND target.firstname = source.firstname  WHEN NOT MATCHED BY TARGET THEN  INSERT (lastname, firstname, birthday)  VALUES (source.lastname, source.firstname, source.birthday);  DROP TABLE #TempAuthor;  END; |

Листинг 4.2 – Процедура импорта ImportAuthorDataFromJson

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в формат JSON.

# 5. Тестирование производительности

Одной из ключевых задач в процессе разработки является тестирование производительности базы данных. Такое тестирование позволяет оценить способность базы данных обрабатывать запросы и возвращать результаты с высокой скоростью. Оценка производительности важна для определения эффективности базы данных и выявления возможных узких мест, которые могут вызывать задержки или проблемы в работе приложения.

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса. Для данной задачи была создана процедура для заполнения таблицы users 100000 строк, скрипт заполнения таблицы представлен в [приложении Е.](#_Приложение_Е)

Время на выполнение процедуры вставки строк в таблицу представлено на рисунке 5.1.



Рисунок 5.1 – Скорость выполнения процедуры вставки

Результат выборки всех строк из таблицы можно посмотреть на рисунке 5.2.

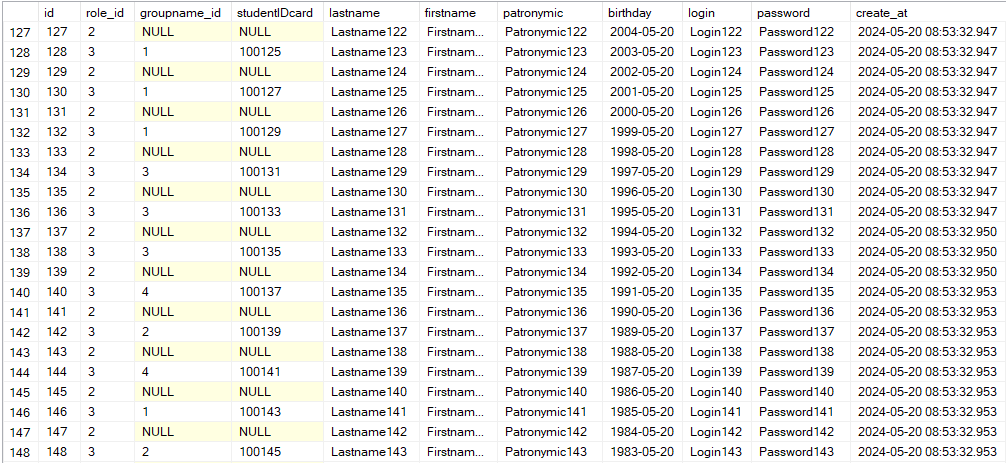


Рисунок 5.2 – Выборка всех строк из таблицы users

Для получения выборки данных использовался запрос, который представлен в листинге 5.1.

|  |
| --- |
| SELECT lastname, firstname FROM users WHERE lastname = 'Lastname80000' and firstname = 'Firstname80000'; |

Листинг 5.1 – Запрос к таблице users

План выполнения – это большое дерево, на котором много объектов. Это дерево – инструкция, что нужно делать, чтобы получить результат запроса. Операторы в плане вызываются по порядку слева направо (с левого верхнего угла), а данные в плане перемещаются справа налево.

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Результат будет представлен на рисунке 5.3.

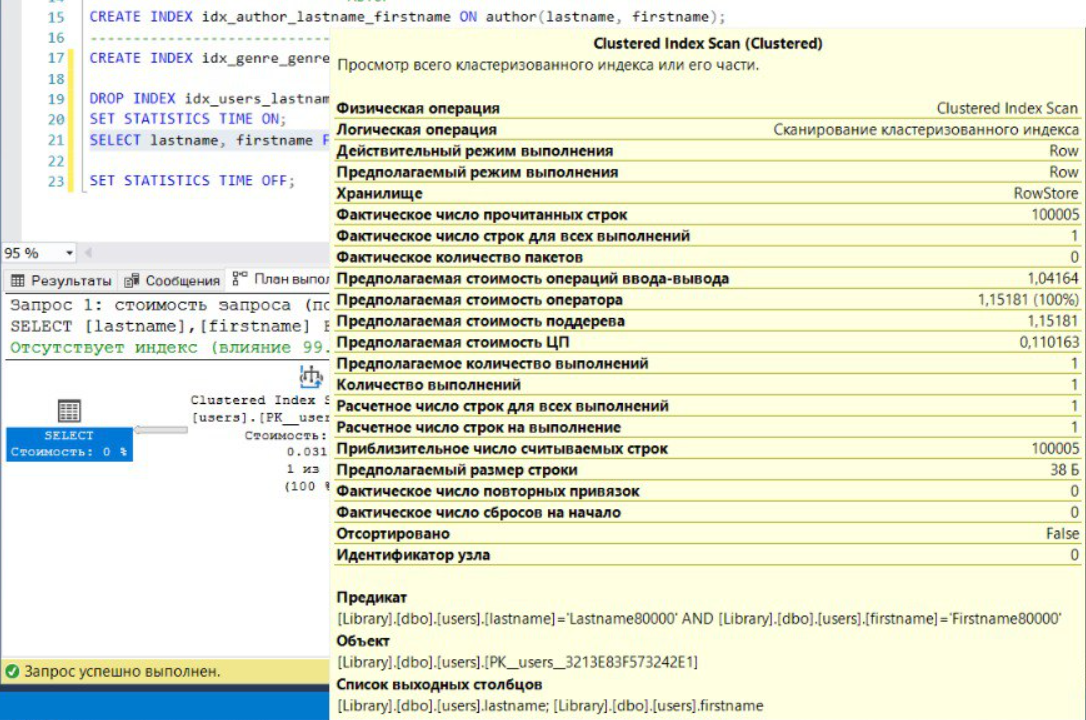


Рисунок 5.3 – Результат выполнения запроса без индекса

Для ускорения данного процесса создадим индекс idx\_users\_lastname\_firstname на поля lastname и firstname, так как именно по этим полям выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. В результате время поиска уменьшилось, это показано на рисунке 5.4.

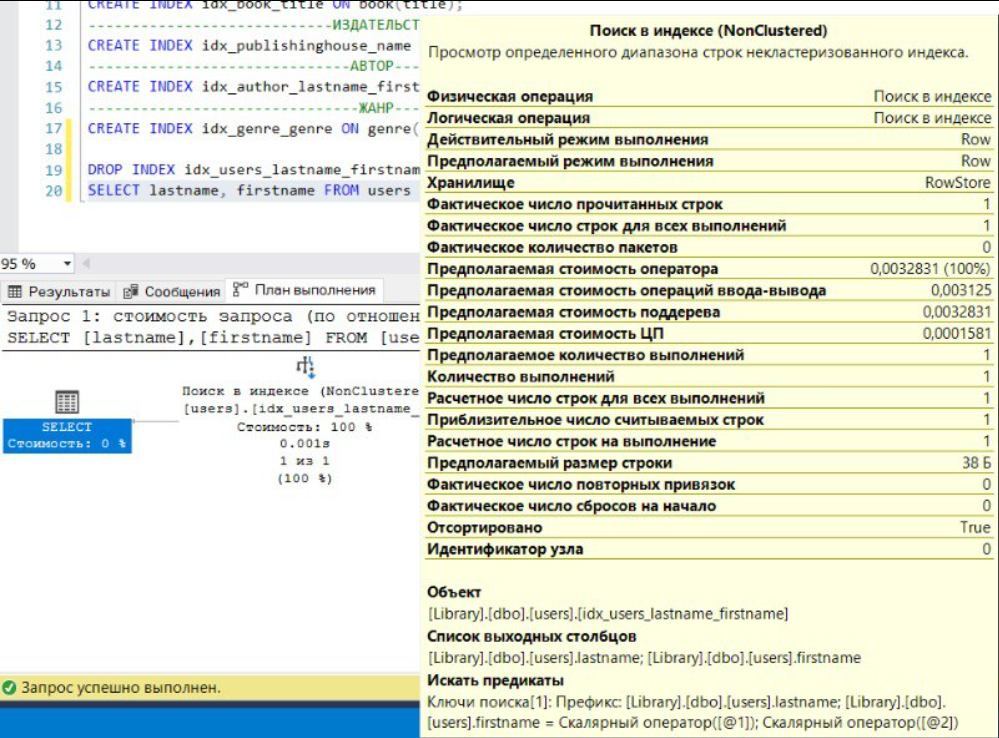


Рисунок 5.4 – Результат выполнения запроса с индексом

Результаты тестирования говорят о том, что создание индексов на полях, по которым выполняются частые запросы, может значительно повысить производительность базы данных.

# 6. Руководство пользователя

При входе на сайт отображается форма авторизации с полями ввода логина и пароля. Читатель может или зарегистрироваться, или войти с уже существующего аккаунта, так же есть возможность войти с администратора и библиотекаря, если ввести конкретный логин и пароль. Это показано на рисунке 6.1.

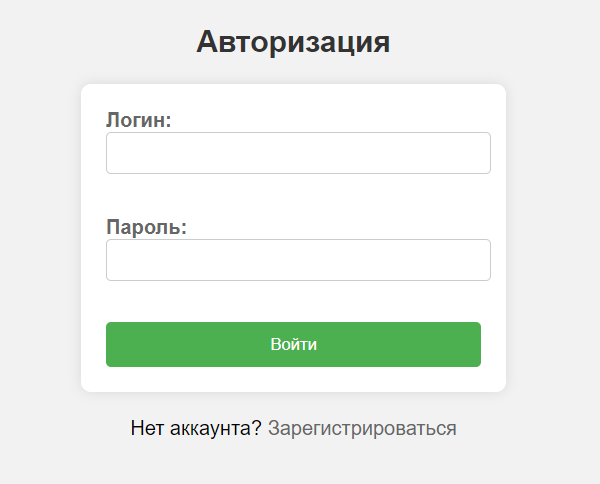


Рисунок 6.1 – Вид формы авторизации

При нажатии на «Зарегистрироваться» открывается форма, где читатель заполняет информацию о себе. Форма представлена на рисунке 6.2.

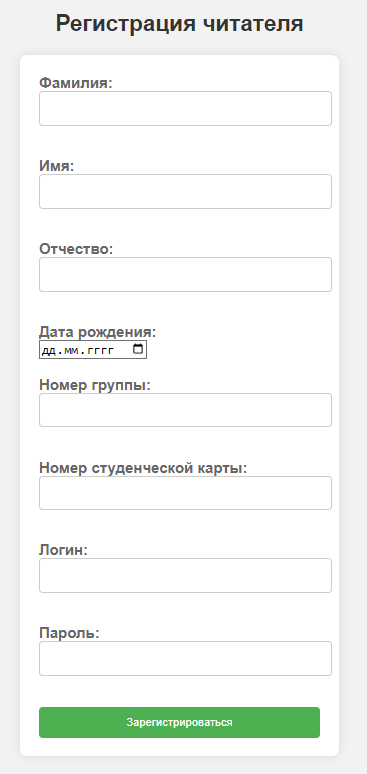


Рисунок 6.2 – Вид формы регистрации читателя

После регистрации и последующей авторизации, у читатель открывается страница с возможностями просмотреть список взятых книг, найти книгу по критериям, просмотреть список книг, а также взять их. Вид главной страницы читателя представлен на рисунке 6.3.

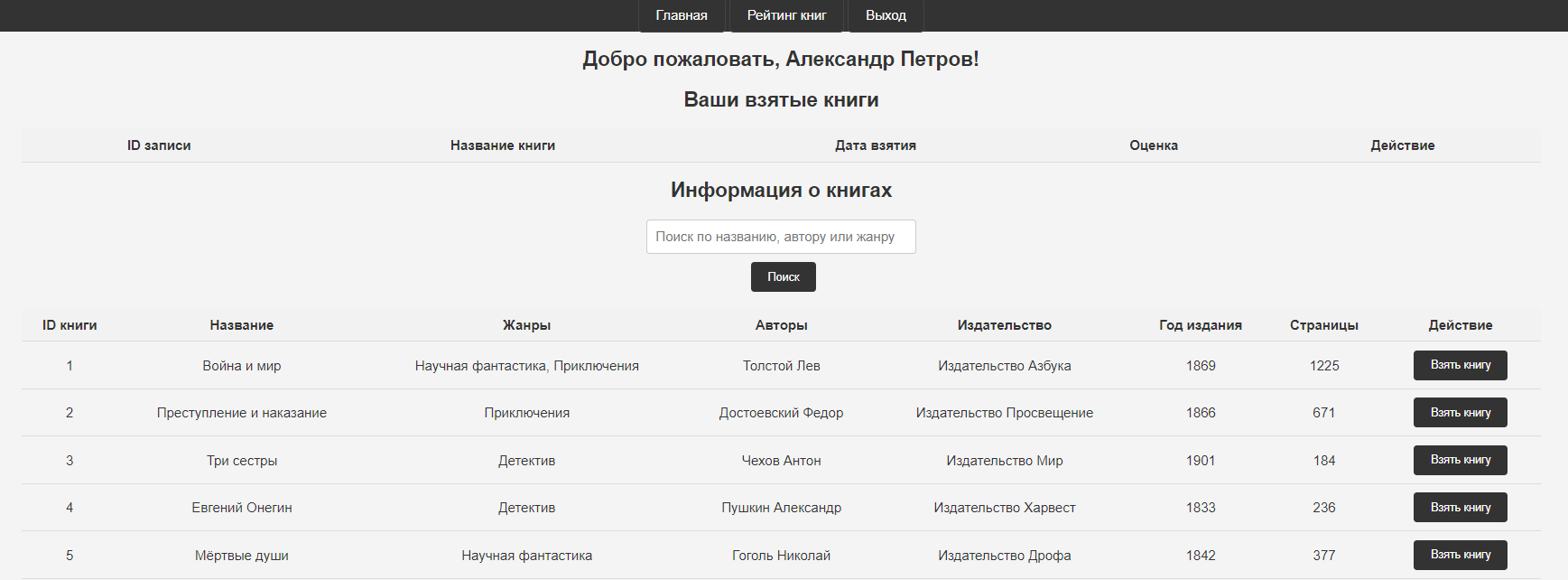


Рисунок 6.3 – Вид главной страницы

На странице рейтинга читатель может просмотреть средние оценки книг от читателей. Страница показана на рисунке 6.4.

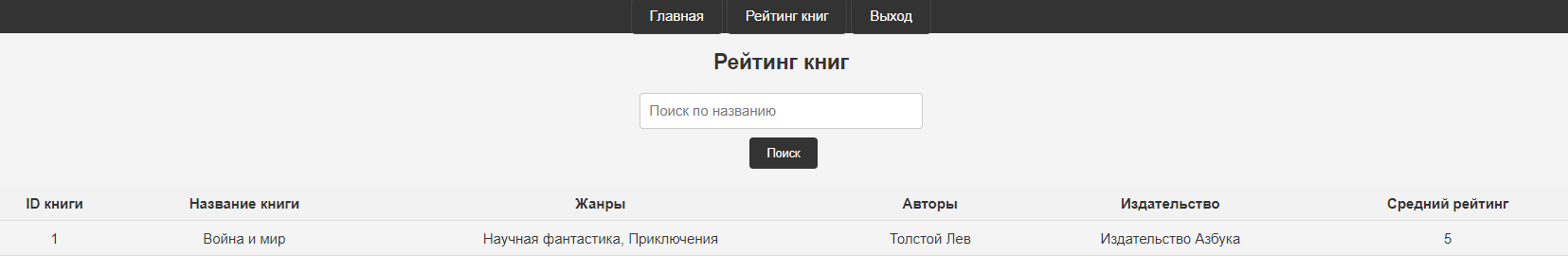


Рисунок 6.4 – Вид страницы рейтинга книг

Если авторизироваться от роли библиотекаря, то сразу можно увидеть страницу с поиском книги. Вид страницы с поиском книги показан на рисунке 6.5.

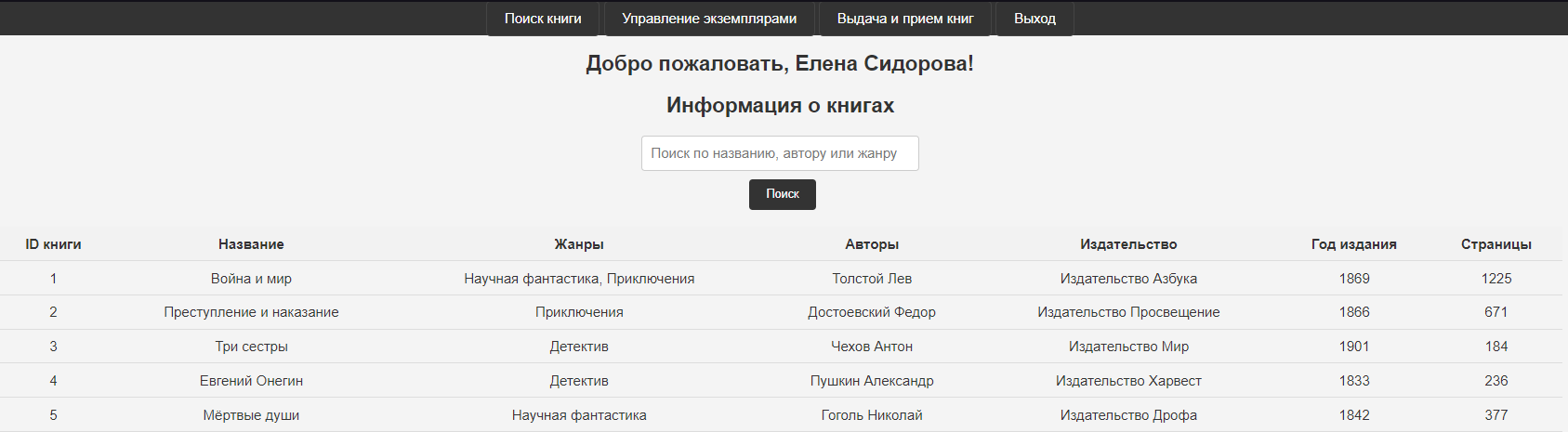


Рисунок 6.5 – Вид страницы поиска книг

Если нажать на навигационном меню «Управление экземплярами», то можно увидеть, что на это странице есть две формы с возможностями добавить и удалить экземпляр книги. Также присутствует поиск по названию книги, подсчет экземпляров книги по ее ID, а также вывод списка экземпляров книг. Вид этой страницы представлен на рисунке 6.6.

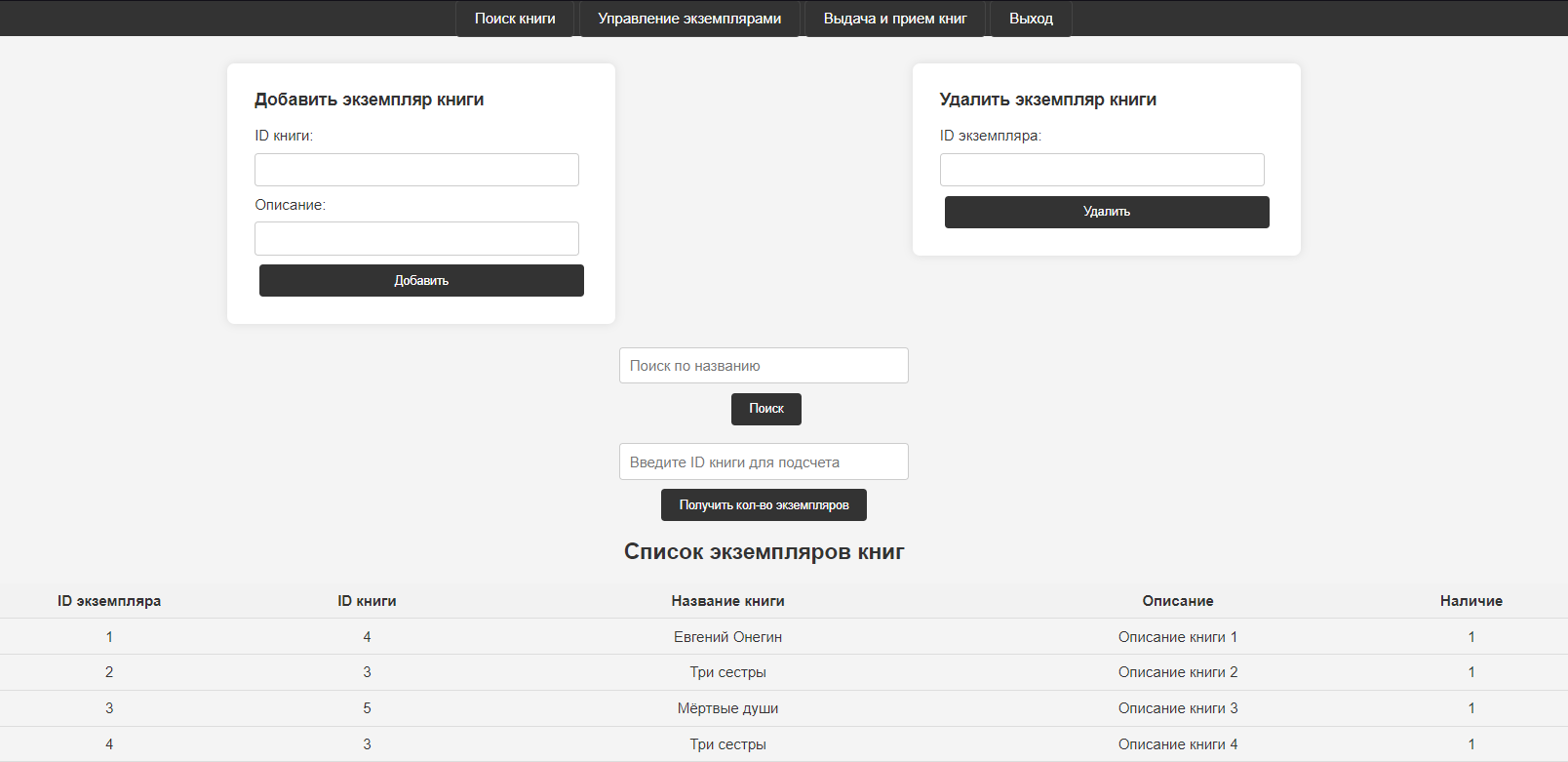


Рисунок 6.6 – Вид страницы управление экземплярами

На сайте для библиотекаря присутствует возможность выдавать и принимать книги от читателей, а также поиск читателей по ID студенческой карты из таблицы «Список книг на руках». Страница показана на рисунке 6.7.

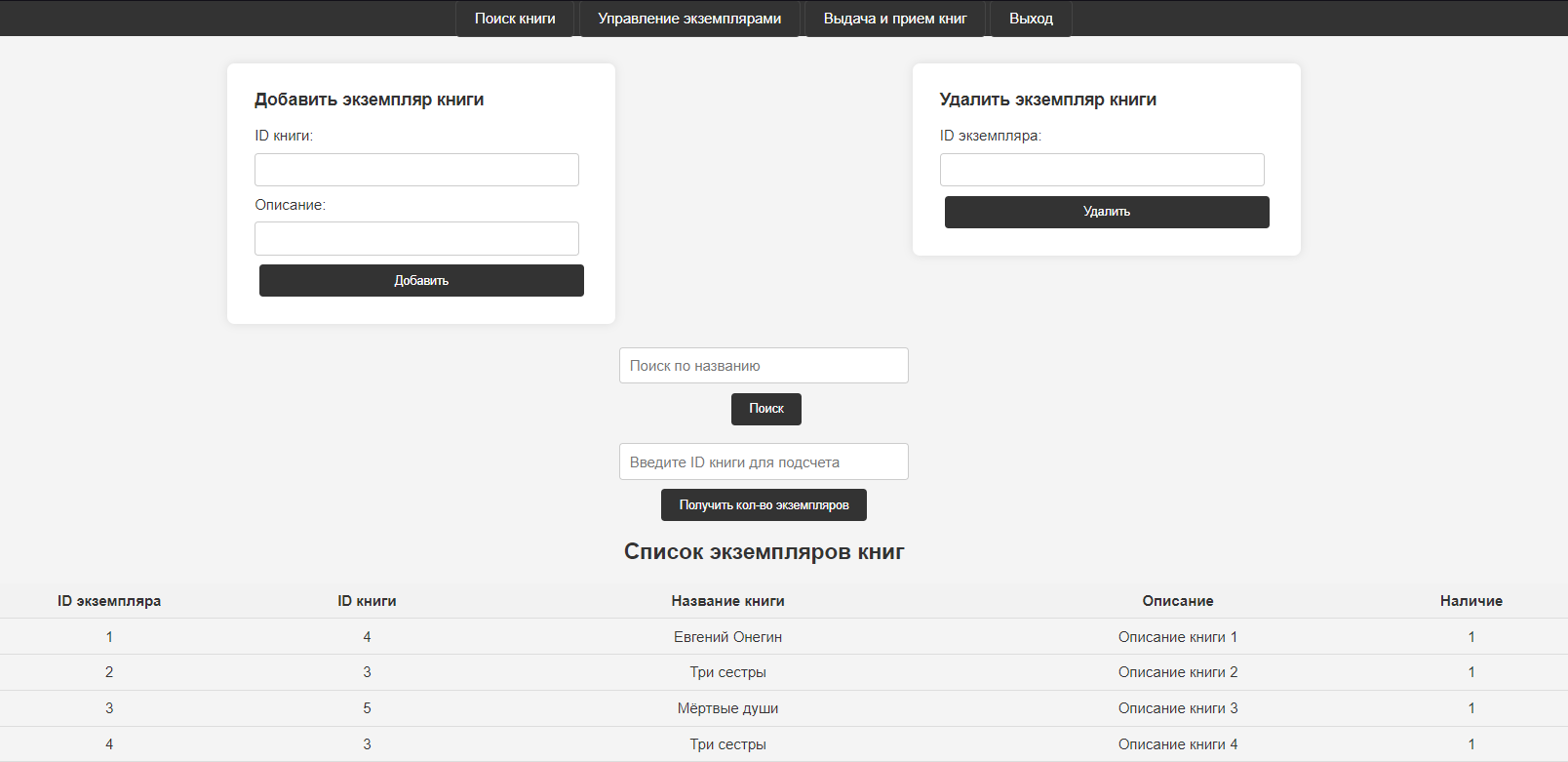


Рисунок 6.7 – Вид страницы выдача и прием книг

Для того, чтобы войти в приложение под администратором, необходимо ввести admin в оба поля формы входа. Вход с учетной записью администратора дает нам управление библиотекарями, читателями и книгами. выполняя процедуры, функции. Здесь можно как добавлять библиотекаря, так и изменять данные существующих. Вид страницы с управлением библиотекарями представлена на рисунке 6.8.

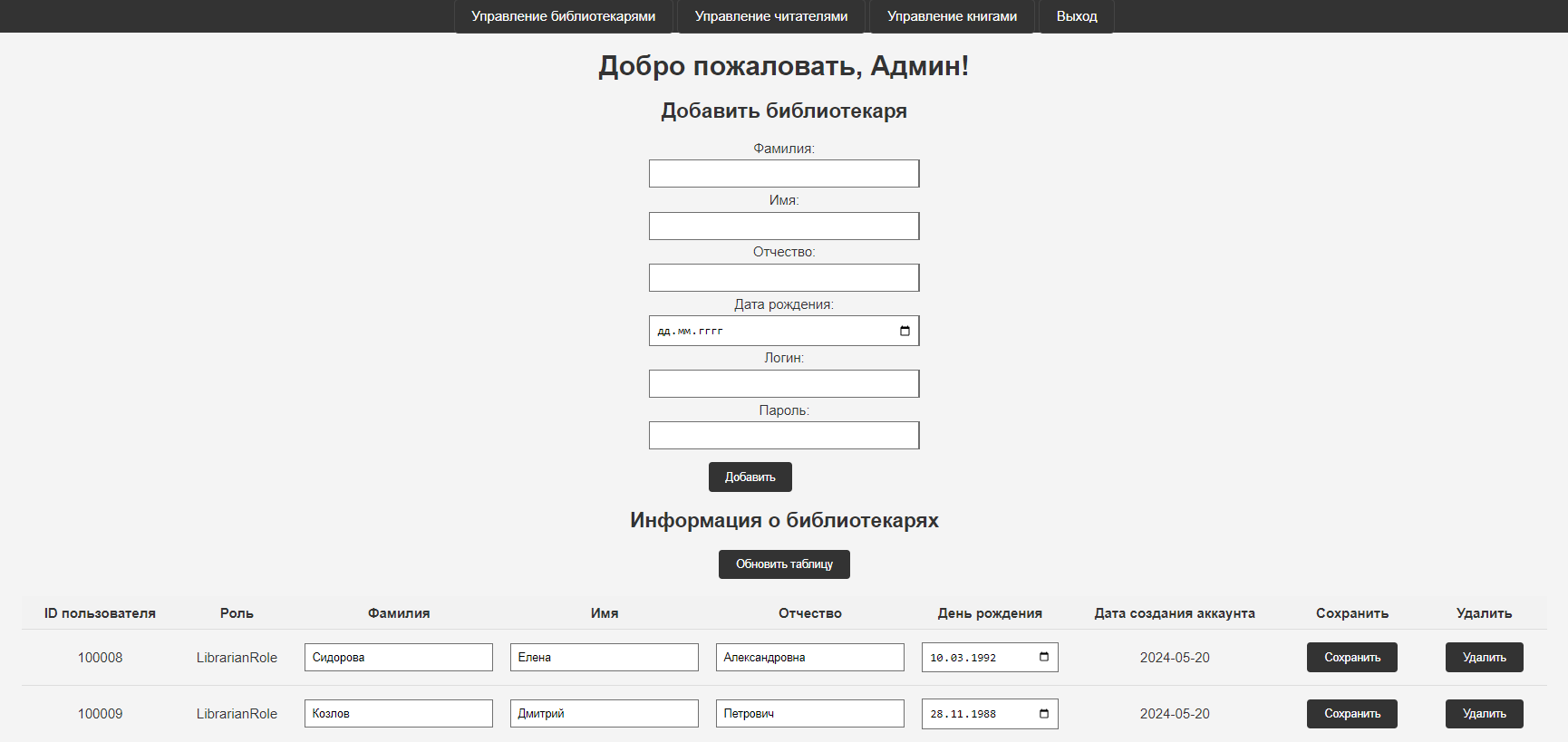


Рисунок 6.8 – Вид страницы управление библиотекарями

На странице управление читателями администратор может выполнять такие действия, как добавление читателя, поиск по ID студенческой карты, так и изменять данные читателей. Страница показана на рисунке 6.9.

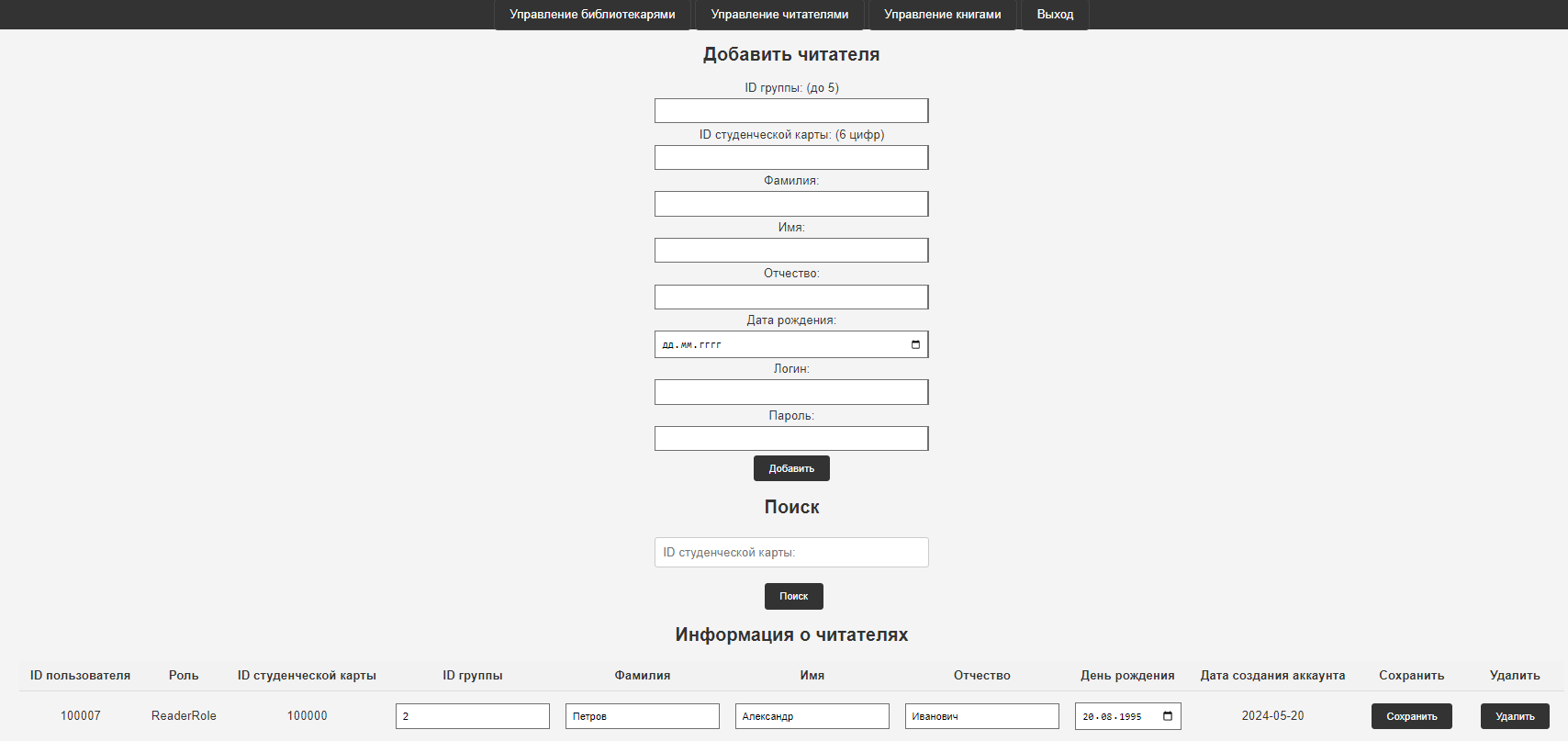


Рисунок 6.9 – Вид страницы управление читателями

На странице управление книгой администратор имеет возможность добавлять, искать по критериям и удалять книгу. Вид страницы с управлением книгой продемонстрирован на рисунке 6.10.

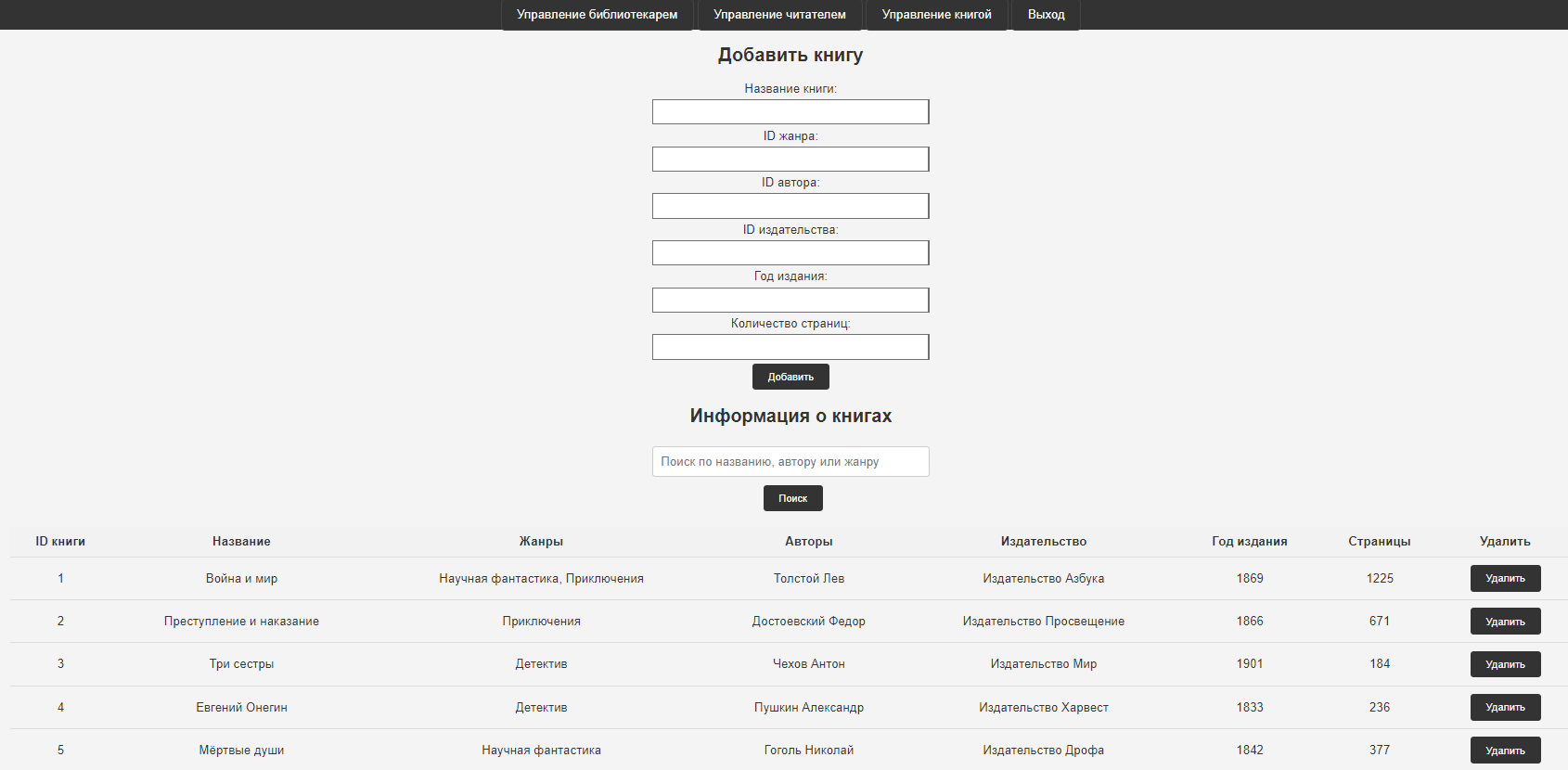


Рисунок 6.9 – Вид страницы управление книгой

Также для всех ролей присутствует возможность выхода из аккаунта.

# Заключение

В данном проекте была поставлена задача разработки базы данных библиотеки университета с использованием технологии резервного копирования и восстановления в СУБД «MSSQL».

В ходе выполнения проекта были использованы различные объекты, такие как таблицы, представления, процедуры, индексы, триггеры, функции, чтобы обеспечить структурированное хранение данных и удобный доступ к ним. В итоге, поставленная цель была успешно достигнута, и на выходе получена готовая база данных.

В рамках тестирования базы данных был использован поток данных, и результаты тестирования оказались положительными. Также были разработаны процедуры импорта и экспорта данных в формате JSON, что значительно облегчило управление данными и повысило удобство использования базы данных.

Разработанная база данных библиотеки университета имеет ряд важных особенностей, которые позволяют эффективно управлять ей. Одной из главных особенностей является технология резервного копирования и восстановления, которая обеспечивает сохранность данных в случае любых сбоев или ошибок. Это означает, что все данные библиотеки, включая информацию о пользователях, книгах и экземплярах книг могут быть быстро восстановлены, минимизируя потерю информации и обеспечивая непрерывность работы сайта.

Кроме того, база данных обладает гибкой системой привилегий, которая позволяет разграничивать доступ к различным функциям и данным в зависимости от роли пользователя. Это обеспечивает безопасность данных и предотвращает несанкционированный доступ.

Делая вывод, можно сказать, что поставленные цели были выполнены. База данных библиотеки университета выполняет необходимую функцию хранения и управления данными.

# Список используемых источников

1. Yale Library [Электронный ресурс] – Режим доступа: https://library.yale.edu – Дата доступа 10.02.2024.
2. Universiteit Leiden [Электронный ресурс] – Режим доступа: https://www.universiteitleiden.nl/en – Дата доступа 10.02.2024.
3. Документация MSSQL [Электронный ресурс] – https://learn.microsoft.com/ru-ru/sql/sql-server/?view=sql-server-ver16 – Дата доступа 15.02.2024.
4. MSSQL Сайт с информацией [Электронный ресурс] – Режим доступа: https://metanit.com/sql/sqlserver/ – Дата доступа 15.02.2024.
5. Технология резервного копирования и восстановления в MSSQL [Электронный ресурс] – Режим доступа: https: //infostart.ru/1c/articles/101210// – Дата доступа 06.05.2024.

# Приложение А

|  |
| --- |
| CREATE DATABASE Library ON PRIMARY  (  NAME = N'Library\_mdf',  FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\Library\_mdf.mdf',  SIZE = 10240KB,  MAXSIZE = UNLIMITED,  FILEGROWTH = 1024KB  ),  FILEGROUP FG1  (  NAME = N'Library\_fg1',  FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\Library\_fg1.ndf',  SIZE = 10240KB,  MAXSIZE = 1GB,  FILEGROWTH = 25%  ),  FILEGROUP FG2  (  NAME = N'Library\_fg2',  FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\Library\_fg2.ndf',  SIZE = 10240KB,  MAXSIZE = 1GB,  FILEGROWTH = 25%  )  LOG ON  (  NAME = N'Library\_log',  FILENAME = N'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\DATA\Library\_log.ldf',  SIZE = 10240KB,  MAXSIZE = 2048GB,  FILEGROWTH = 10%  );  CREATE TABLE role (  id int IDENTITY(1,1) PRIMARY KEY,  role varchar(50) NOT NULL UNIQUE  ) ON FG1;  CREATE TABLE users (  id int IDENTITY(1,1) PRIMARY KEY,  role\_id int NOT NULL,  groupname\_id int DEFAULT NULL,  studentIDcard int DEFAULT NULL,  lastname varchar(255) NOT NULL,  firstname varchar(255) NOT NULL,  patronymic varchar(255) NOT NULL,  birthday date NOT NULL,  login varchar(255) NOT NULL UNIQUE,  password varchar(255) NOT NULL,  create\_at datetime NOT NULL DEFAULT GETDATE()  ) ON FG1;  CREATE TABLE groupname (  id int IDENTITY(1,1) PRIMARY KEY,  faculty\_id int NOT NULL,  year int NOT NULL,  name varchar(50) NOT NULL  ) ON FG1;  CREATE TABLE faculty (  id int IDENTITY(1,1) PRIMARY KEY,  name varchar(255) NOT NULL UNIQUE  ) ON FG1;  CREATE TABLE listgetbooks (  id int IDENTITY(1,1) PRIMARY KEY,  user\_id int NOT NULL,  sample\_id int NOT NULL,  takedate date NOT NULL,  returndate date DEFAULT NULL,  mark int DEFAULT NULL  ) ON FG2;  CREATE TABLE sample (  id int IDENTITY(1,1) PRIMARY KEY,  book\_id int NOT NULL,  description varchar(255) NOT NULL,  presence tinyint CHECK (presence IN (0, 1))  ) ON FG2;  CREATE TABLE book (  id int PRIMARY KEY,  publishinghouse\_id int NOT NULL,  title varchar(255) NOT NULL,  year int NOT NULL,  pages smallint NOT NULL  );  CREATE TABLE publishinghouse (  id int IDENTITY(1,1) PRIMARY KEY,  name varchar(255) NOT NULL UNIQUE,  address varchar(255) NOT NULL  ) ON FG2;  CREATE TABLE genre (  id int IDENTITY(1,1) PRIMARY KEY,  genre varchar(255) NOT NULL UNIQUE  ) ON FG2;  CREATE TABLE books\_genres (  book\_id int,  genre\_id int,  PRIMARY KEY (book\_id, genre\_id),  FOREIGN KEY (book\_id) REFERENCES book (id) ON DELETE NO ACTION,  FOREIGN KEY (genre\_id) REFERENCES genre (id) ON DELETE NO ACTION  ) ON FG2;  CREATE TABLE author (  id int IDENTITY(1,1) PRIMARY KEY,  lastname varchar(255) NOT NULL,  firstname varchar(255) NOT NULL,  birthday date  ) ON FG2;  CREATE TABLE books\_authors (  book\_id int,  author\_id int,  PRIMARY KEY (book\_id, author\_id),  FOREIGN KEY (book\_id) REFERENCES book (id) ON DELETE NO ACTION,  FOREIGN KEY (author\_id) REFERENCES author (id) ON DELETE NO ACTION  ) ON FG2;  ALTER TABLE users  ADD CONSTRAINT fk\_users\_role  FOREIGN KEY (role\_id) REFERENCES role(id) ON DELETE NO ACTION;  ALTER TABLE users  ADD CONSTRAINT fk\_users\_groupname  FOREIGN KEY (groupname\_id) REFERENCES groupname(id) ON DELETE NO ACTION;  ALTER TABLE groupname  ADD CONSTRAINT fk\_groupname\_faculty  FOREIGN KEY (faculty\_id) REFERENCES faculty(id) ON DELETE NO ACTION;  ALTER TABLE listgetbooks  ADD CONSTRAINT fk\_listgetbooks\_users  FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE NO ACTION;  ALTER TABLE listgetbooks  ADD CONSTRAINT fk\_listgetbooks\_sample  FOREIGN KEY (sample\_id) REFERENCES sample(id) ON DELETE NO ACTION;  ALTER TABLE sample  ADD CONSTRAINT fk\_sample\_book  FOREIGN KEY (book\_id) REFERENCES book(id) ON DELETE NO ACTION;  ALTER TABLE book  ADD CONSTRAINT fk\_book\_publishinghouse  FOREIGN KEY (publishinghouse\_id) REFERENCES publishinghouse(id) ON DELETE NO ACTION;  ALTER TABLE books\_authors  ADD CONSTRAINT fk\_books\_authors\_book  FOREIGN KEY (book\_id) REFERENCES book(id) ON DELETE NO ACTION;  ALTER TABLE books\_authors  ADD CONSTRAINT fk\_books\_authors\_author  FOREIGN KEY (author\_id) REFERENCES author(id) ON DELETE NO ACTION;  ALTER TABLE books\_genres  ADD CONSTRAINT fk\_books\_genres\_book  FOREIGN KEY (book\_id) REFERENCES book(id) ON DELETE NO ACTION;  ALTER TABLE books\_genres  ADD CONSTRAINT fk\_books\_genres\_genre  FOREIGN KEY (genre\_id) REFERENCES genre(id) ON DELETE NO ACTION; |

# Приложение Б

|  |
| --- |
| CREATE FUNCTION GetBorrowedBooks(@userId INT)  RETURNS TABLE  AS  RETURN (  SELECT l.id, b.title, l.takedate  FROM listgetbooks l  JOIN sample s ON l.sample\_id = s.id  JOIN book b ON s.book\_id = b.id  WHERE l.user\_id = @userId AND l.returndate IS NULL  );  CREATE FUNCTION dbo.SearchBooksByTitleInRating (@title NVARCHAR(MAX))  RETURNS TABLE  AS  RETURN  (  SELECT b.id, b.title,  STUFF((SELECT DISTINCT ', ' + g.genre  FROM books\_genres bg  JOIN genre g ON bg.genre\_id = g.id  WHERE bg.book\_id = b.id FOR XML PATH('')), 1, 2, '') AS genres,  STUFF((SELECT DISTINCT ', ' + a.lastname + ' ' + a.firstname  FROM books\_authors ba  JOIN author a ON ba.author\_id = a.id  WHERE ba.book\_id = b.id FOR XML PATH('')), 1, 2, '') AS authors,  p.name AS publishinghouse, AVG(l.mark) AS average\_rating  FROM book b  JOIN sample s ON b.id = s.book\_id  JOIN listgetbooks l ON l.sample\_id = s.id  LEFT JOIN publishinghouse p ON b.publishinghouse\_id = p.id  WHERE b.title LIKE '%' + @title + '%'  GROUP BY b.id, b.title, p.name  );  CREATE FUNCTION dbo.GetBookRatings()  RETURNS TABLE  AS  RETURN  (  SELECT b.id, b.title,  STUFF((SELECT DISTINCT ', ' + g.genre  FROM books\_genres bg  JOIN genre g ON bg.genre\_id = g.id  WHERE bg.book\_id = b.id FOR XML PATH('')), 1, 2, '') AS genres,  STUFF((SELECT DISTINCT ', ' + a.lastname + ' ' + a.firstname  FROM books\_authors ba  JOIN author a ON ba.author\_id = a.id  WHERE ba.book\_id = b.id FOR XML PATH('')), 1, 2, '') AS authors,  p.name AS publishinghouse, AVG(l.mark) AS average\_rating  FROM book b  JOIN sample s ON b.id = s.book\_id  JOIN listgetbooks l ON l.sample\_id = s.id  LEFT JOIN publishinghouse p ON b.publishinghouse\_id = p.id  GROUP BY b.id, b.title, p.name  );  CREATE FUNCTION SearchBooksByTitle(@Title NVARCHAR(255))  RETURNS TABLE  AS  RETURN  (  SELECT s.id as id, s.book\_id, b.title, s.description, s.presence  FROM book AS b  INNER JOIN sample AS s ON b.id = s.book\_id  WHERE b.title LIKE '%' + @Title + '%'  )  CREATE FUNCTION GetBookSampleCount(@book\_id int)  RETURNS INT  AS  BEGIN  DECLARE @sample\_count INT;  SELECT @sample\_count = COUNT(id) FROM sample WHERE book\_id = @book\_id;  RETURN @sample\_count;  END;  CREATE FUNCTION GetBookSamples()  RETURNS TABLE  AS  RETURN  (  SELECT sample.id AS id, sample.book\_id, book.title, sample.description, sample.presence  FROM sample  INNER JOIN book ON sample.book\_id = book.id  );  CREATE FUNCTION ViewBooks()  RETURNS TABLE  AS  RETURN  (  SELECT DISTINCT  b.id, b.title,  b.year, b.pages,  p.name AS publishinghouse,  STUFF((  SELECT ', ' + g.genre  FROM books\_genres bg  JOIN genre g ON bg.genre\_id = g.id  WHERE bg.book\_id = b.id  FOR XML PATH('')), 1, 2, '') AS genres,  STUFF((  SELECT ', ' + CONCAT(a.lastname, ' ', a.firstname)  FROM books\_authors ba  JOIN author a ON ba.author\_id = a.id  WHERE ba.book\_id = b.id  FOR XML PATH('')), 1, 2, '') AS authors  FROM book b  INNER JOIN publishinghouse p ON b.publishinghouse\_id = p.id  LEFT JOIN books\_genres bg ON b.id = bg.book\_id  LEFT JOIN genre g ON bg.genre\_id = g.id  LEFT JOIN books\_authors ba ON b.id = ba.book\_id  LEFT JOIN author a ON ba.author\_id = a.id  GROUP BY b.id, b.title, b.year, b.pages, p.name  );  CREATE FUNCTION SearchBooks (@searchString NVARCHAR(MAX))  RETURNS TABLE  AS  RETURN  (  SELECT DISTINCT  b.id,  b.title,  b.year,  b.pages,  p.name AS publishinghouse,  STUFF((  SELECT ', ' + g.genre  FROM books\_genres bg  JOIN genre g ON bg.genre\_id = g.id  WHERE bg.book\_id = b.id  FOR XML PATH('')), 1, 2, '') AS genres,  STUFF((  SELECT ', ' + CONCAT(a.lastname, ' ', a.firstname)  FROM books\_authors ba  JOIN author a ON ba.author\_id = a.id  WHERE ba.book\_id = b.id  FOR XML PATH('')), 1, 2, '') AS authors  FROM book b  INNER JOIN publishinghouse p ON b.publishinghouse\_id = p.id  LEFT JOIN books\_genres bg ON b.id = bg.book\_id  LEFT JOIN genre g ON bg.genre\_id = g.id  LEFT JOIN books\_authors ba ON b.id = ba.book\_id  LEFT JOIN author a ON ba.author\_id = a.id  WHERE b.title LIKE '%' + @searchString + '%' OR a.firstname LIKE '%' + @searchString + '%' OR a.lastname LIKE '%' + @searchString + '%' OR g.genre LIKE '%' + @searchString + '%'  GROUP BY b.id, b.title, b.year, b.pages, p.name  ); |

# Приложение В

|  |
| --- |
| CREATE PROCEDURE BorrowBook  @userId INT,  @bookId INT  AS  BEGIN  BEGIN TRY  DECLARE @sampleId INT;  DECLARE @AlreadyBorrowed BIT;  SELECT @AlreadyBorrowed = CASE WHEN COUNT(lg.id) > 0 THEN 1 ELSE 0 END  FROM listgetbooks lg  JOIN sample s ON lg.sample\_id = s.id  WHERE lg.user\_id = @userId AND s.book\_id = @bookId AND lg.returndate IS NULL;  IF @AlreadyBorrowed = 1  BEGIN  PRINT 'Пользователь уже взял экземпляр этой книги';  END  ELSE  BEGIN  SELECT TOP 1 @sampleId = id FROM sample WHERE book\_id = @bookId AND presence = 1;  IF (@sampleId IS NOT NULL)  BEGIN  INSERT INTO listgetbooks (user\_id, sample\_id, takedate)  VALUES (@userId, @sampleId, GETDATE());    UPDATE sample  SET presence = 0  WHERE id = @sampleId;  PRINT 'Книга успешно взята в аренду';  END  ELSE  BEGIN  PRINT 'Нет доступных экземпляров книги';  END  END  END TRY  BEGIN CATCH  PRINT 'При аренде книги произошла ошибка: ' + ERROR\_MESSAGE();  PRINT 'Код ошибки: ' + CAST(ERROR\_NUMBER() AS VARCHAR(10));  PRINT 'Уровень ошибки: ' + CAST(ERROR\_SEVERITY() AS VARCHAR(10));  PRINT 'Строка, на которой произошла ошибка: ' + CAST(ERROR\_LINE() AS VARCHAR(10));  PRINT 'Процедура, в которой произошла ошибка: ' + ERROR\_PROCEDURE();  END CATCH  END  CREATE PROCEDURE ReturnBookReader  @listGetBookId INT,  @mark INT  AS  BEGIN  BEGIN TRY  DECLARE @sampleId INT;  DECLARE @UserId INT;  DECLARE @BookId INT;  DECLARE @AlreadyRated BIT;  SELECT @sampleId = sample\_id, @UserId = user\_id  FROM listgetbooks  WHERE id = @listGetBookId;  SELECT @BookId = book\_id  FROM sample  WHERE id = @sampleId;  IF @sampleId IS NOT NULL AND @UserId IS NOT NULL AND @BookId IS NOT NULL  BEGIN  SELECT @AlreadyRated = CASE WHEN COUNT(lg.mark) > 0 THEN 1 ELSE 0 END  FROM listgetbooks lg  JOIN sample s ON lg.sample\_id = s.id  WHERE lg.user\_id = @UserId AND s.book\_id = @BookId AND lg.mark IS NOT NULL;  IF @AlreadyRated = 1  BEGIN  UPDATE listgetbooks  SET returndate = GETDATE(), mark = NULL  WHERE id = @listGetBookId AND returndate IS NULL;  END  ELSE  BEGIN  UPDATE listgetbooks  SET returndate = GETDATE(), mark = @mark  WHERE id = @listGetBookId AND returndate IS NULL;  END  UPDATE sample  SET presence = 1  WHERE id = @sampleId;  PRINT 'Книга успешно возвращена';  END  ELSE  BEGIN  PRINT 'При возврате книги произошла ошибка: Некорректные данные';  END;  END TRY  BEGIN CATCH  PRINT 'При возврате книги произошла ошибка: ' + ERROR\_MESSAGE();  PRINT 'Код ошибки: ' + CAST(ERROR\_NUMBER() AS VARCHAR(10));  PRINT 'Уровень ошибки: ' + CAST(ERROR\_SEVERITY() AS VARCHAR(10));  PRINT 'Строка, на которой произошла ошибка: ' + CAST(ERROR\_LINE() AS VARCHAR(10));  PRINT 'Процедура, в которой произошла ошибка: ' + ERROR\_PROCEDURE();  END CATCH  END  CREATE PROCEDURE AddLibrarian  @lastname varchar(255),  @firstname varchar(255),  @patronymic varchar(255),  @birthday date,  @login varchar(255),  @password varchar(255)  AS  BEGIN  BEGIN TRY  INSERT INTO users (role\_id, lastname, firstname, patronymic, birthday, login, password)  VALUES (2, @lastname, @firstname, @patronymic, @birthday, @login, @password);  PRINT 'Библиотекарь успешно добавлен';  END TRY  BEGIN CATCH  PRINT 'При добавлении читателя произошла ошибка: ' + ERROR\_MESSAGE();  PRINT 'Код ошибки: ' + CAST(ERROR\_NUMBER() AS VARCHAR(10));  PRINT 'Уровень ошибки: ' + CAST(ERROR\_SEVERITY() AS VARCHAR(10));  PRINT 'Строка, на которой произошла ошибка: ' + CAST(ERROR\_LINE() AS VARCHAR(10));  PRINT 'Процедура, в которой произошла ошибка: ' + ERROR\_PROCEDURE();  END CATCH  END;  CREATE PROCEDURE AddReader  @groupname\_id int,  @studentIDcard int,  @lastname varchar(255),  @firstname varchar(255),  @patronymic varchar(255),  @birthday date,  @login varchar(255),  @password varchar(255)  AS  BEGIN  BEGIN TRY  INSERT INTO users (role\_id, groupname\_id, studentIDcard, lastname, firstname, patronymic, birthday, login, password)  VALUES (3, @groupname\_id, @studentIDcard, @lastname, @firstname, @patronymic, @birthday, @login, @password);  PRINT 'Читатель успешно добавлен';  END TRY  BEGIN CATCH  PRINT 'При добавлении читателя произошла ошибка: ' + ERROR\_MESSAGE();  PRINT 'Код ошибки: ' + CAST(ERROR\_NUMBER() AS VARCHAR(10));  PRINT 'Уровень ошибки: ' + CAST(ERROR\_SEVERITY() AS VARCHAR(10));  PRINT 'Строка, на которой произошла ошибка: ' + CAST(ERROR\_LINE() AS VARCHAR(10));  PRINT 'Процедура, в которой произошла ошибка: ' + ERROR\_PROCEDURE();  END CATCH  END;  CREATE PROCEDURE AddBook  @publishinghouse\_id INT,  @title NVARCHAR(255),  @year INT,  @pages INT,  @genre\_id INT,  @author\_id INT  AS  BEGIN  IF NOT EXISTS (SELECT 1 FROM publishinghouse WHERE id = @publishinghouse\_id)  BEGIN  THROW 50003, 'Издательство с указанным ID не найдено', 1;  RETURN;  END;  IF NOT EXISTS (SELECT 1 FROM genre WHERE id = @genre\_id)  BEGIN  THROW 50004, 'Жанр с указанным ID не найден', 1;  RETURN;  END;  IF NOT EXISTS (SELECT 1 FROM author WHERE id = @author\_id)  BEGIN  THROW 50005, 'Автор с указанным ID не найден', 1;  RETURN;  END;  DECLARE @max\_book\_id INT;  SELECT @max\_book\_id = ISNULL(MAX(id), 0) FROM book;  SET @max\_book\_id = @max\_book\_id + 1;  INSERT INTO book (id, publishinghouse\_id, title, year, pages)  VALUES (@max\_book\_id, @publishinghouse\_id, @title, @year, @pages);  INSERT INTO books\_genres (book\_id, genre\_id)  VALUES (@max\_book\_id, @genre\_id);  INSERT INTO books\_authors (book\_id, author\_id)  VALUES (@max\_book\_id, @author\_id);  SELECT 'Книга успешно добавлена' AS Result;  END;  CREATE PROCEDURE DeleteBook  @book\_id int  AS  BEGIN  BEGIN TRY  IF NOT EXISTS (SELECT 1 FROM book WHERE id = @book\_id)  BEGIN  THROW 51006, 'Книги с таким ID не существует', 1;  END  ELSE IF EXISTS (SELECT 1 FROM listgetbooks WHERE sample\_id IN (SELECT id FROM sample WHERE book\_id = @book\_id) AND returndate IS NULL)  BEGIN  THROW 51020, 'Невозможно удалить книгу, так как она находится в распоряжении читателя', 1;  END  ELSE  BEGIN  DELETE FROM listgetbooks WHERE sample\_id IN (SELECT id FROM sample WHERE book\_id = @book\_id);  DELETE FROM sample WHERE book\_id = @book\_id;  DELETE FROM books\_genres WHERE book\_id = @book\_id;  DELETE FROM books\_authors WHERE book\_id = @book\_id;  DELETE FROM book WHERE id = @book\_id;  PRINT 'Книга успешно удалена';  END;  END TRY  BEGIN CATCH  PRINT 'При удалении книги произошла ошибка: ' + ERROR\_MESSAGE();  PRINT 'Код ошибки: ' + CAST(ERROR\_NUMBER() AS VARCHAR(10));  PRINT 'Уровень ошибки: ' + CAST(ERROR\_SEVERITY() AS VARCHAR(10));  PRINT 'Строка, на которой произошла ошибка: ' + CAST(ERROR\_LINE() AS VARCHAR(10));  PRINT 'Процедура, в которой произошла ошибка: ' + ERROR\_PROCEDURE();  END CATCH  END; |

# Приложение Г

|  |
| --- |
| CREATE TRIGGER trg\_check\_year  ON book  AFTER INSERT, UPDATE  AS  BEGIN  DECLARE @CurrentYear INT;  SET @CurrentYear = YEAR(GETDATE());  IF EXISTS (SELECT 1 FROM inserted WHERE year > @CurrentYear OR year < 1000)  BEGIN  RAISERROR ('Год издания книги должен быть больше тысячного года и не превышать текущий год', 16, 1);  ROLLBACK TRANSACTION;  RETURN;  END;  END;  CREATE TRIGGER trg\_check\_studentIDcard\_length  ON users  AFTER INSERT, UPDATE  AS  BEGIN  IF EXISTS (  SELECT 1  FROM inserted  WHERE LEN(CAST(studentIDcard AS nvarchar)) <> 6 AND studentIDcard IS NOT NULL  )  BEGIN  THROW 50000, 'ID cтуденческой карты должно состоять из 6 цифр', 1;  END  END;  CREATE TRIGGER trg\_check\_unique\_studentIDcard  ON users  AFTER INSERT, UPDATE  AS  BEGIN  IF EXISTS (  SELECT 1  FROM inserted i  JOIN users u ON i.studentIDcard = u.studentIDcard  WHERE i.studentIDcard IS NOT NULL AND i.id <> u.id  )  BEGIN  THROW 50002, 'ID cтуденческой карты дублируется', 1;  END  END; |

# Приложение Д

|  |
| --- |
| CREATE ROLE ReaderRole;  ALTER ROLE ReaderRole ADD MEMBER ReaderUser;  GRANT SELECT ON dbo.author TO ReaderUser;  GRANT SELECT ON dbo.books\_authors TO ReaderUser;  GRANT SELECT ON dbo.books\_genres TO ReaderUser;  GRANT SELECT ON dbo.genre TO ReaderUser;  GRANT SELECT ON dbo.publishinghouse TO ReaderUser;  GRANT SELECT ON dbo.book TO ReaderUser;  GRANT SELECT ON dbo.users TO ReaderUser;  GRANT SELECT, UPDATE, INSERT ON dbo.listgetbooks TO ReaderUser;  GRANT SELECT, UPDATE ON dbo.sample TO ReaderUser;  GRANT EXECUTE ON dbo.RegisterReader TO ReaderRole;  GRANT EXECUTE ON dbo.BorrowBook TO ReaderRole;  GRANT EXECUTE ON dbo.ReturnBookReader TO ReaderRole;  GRANT SELECT ON dbo.GetBorrowedBooks TO ReaderRole;  GRANT SELECT ON dbo.SearchBooksByTitleInRating TO ReaderRole;  GRANT SELECT ON dbo.GetBookRatings TO ReaderRole;  CREATE ROLE AdminRole;  ALTER ROLE AdminRole ADD MEMBER AdminUser;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.author TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.book TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.books\_authors TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.books\_genres TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.faculty TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.genre TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.groupname TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.listgetbooks TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.publishinghouse TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.role TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.sample TO AdminRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.users TO AdminRole;  GRANT EXECUTE ON dbo.AddLibrarian TO AdminRole;  GRANT EXECUTE ON dbo.AddReader TO AdminRole;  GRANT EXECUTE ON dbo.UpdateLibrarian TO AdminRole;  GRANT EXECUTE ON dbo.UpdateReader TO AdminRole;  GRANT EXECUTE ON dbo.DeleteUser TO AdminRole;  GRANT EXECUTE ON dbo.AddBook TO AdminRole;  GRANT EXECUTE ON dbo.DeleteBook TO AdminRole;  GRANT SELECT ON dbo.SearchReaderByStudentIDCard TO AdminRole;  GRANT SELECT ON dbo.GetAllReaders TO AdminRole;  CREATE ROLE LibrarianRole;  ALTER ROLE LibrarianRole ADD MEMBER LibrarianUser;  GRANT SELECT ON dbo.author TO LibrarianRole;  GRANT SELECT ON dbo.book TO LibrarianRole;  GRANT SELECT ON dbo.books\_authors TO LibrarianRole;  GRANT SELECT ON dbo.books\_genres TO LibrarianRole;  GRANT SELECT ON dbo.genre TO LibrarianRole;  GRANT SELECT ON dbo.users TO LibrarianRole;  GRANT SELECT ON dbo.book TO LibrarianRole;  GRANT SELECT ON dbo.publishinghouse TO LibrarianRole;  GRANT SELECT, UPDATE, INSERT ON dbo.listgetbooks TO LibrarianRole;  GRANT SELECT, UPDATE, INSERT, DELETE ON dbo.sample TO LibrarianRole;  GRANT EXECUTE ON dbo.AddBookSample TO LibrarianRole;  GRANT EXECUTE ON dbo.DeleteBookSample TO LibrarianRole;  GRANT EXECUTE ON dbo.IssueBook TO LibrarianRole;  GRANT EXECUTE ON dbo.ReturnBookLibrarian TO LibrarianRole;  GRANT EXECUTE ON dbo.SearchIssuedBooksByStudentID TO LibrarianRole;  GRANT SELECT ON dbo.SearchBooksByTitle TO LibrarianRole;  GRANT EXECUTE ON dbo.GetBookSampleCount TO LibrarianRole;  GRANT SELECT ON dbo.GetBookSamples TO LibrarianRole;  GRANT SELECT ON dbo.GetIssuedBooks TO LibrarianRole;  GRANT EXECUTE ON dbo.LoginUser TO AdminRole, LibrarianRole, ReaderRole;  GRANT SELECT ON dbo.ViewBooks TO ReaderRole, LibrarianRole, AdminRole;  GRANT SELECT ON dbo.SearchBooks TO ReaderRole, LibrarianRole, AdminRole;  GRANT SELECT, UPDATE ON dbo.ReaderDetails TO AdminRole;  GRANT SELECT, UPDATE ON dbo.LibrarianDetails TO AdminRole;  GRANT SELECT ON dbo.BookDetails TO AdminRole, LibrarianRole, ReaderRole;  GRANT SELECT ON dbo.AvailableBooks TO ReaderRole, LibrarianRole; |

# Приложение Е

|  |
| --- |
| CREATE OR ALTER PROCEDURE InsertUsers  AS  BEGIN  DECLARE @counter INT = 1  DECLARE @role\_id INT  DECLARE @groupname\_id INT  DECLARE @studentIDcard INT  DECLARE @lastname VARCHAR(255)  DECLARE @firstname VARCHAR(255)  DECLARE @patronymic VARCHAR(255)  DECLARE @birthday DATE  DECLARE @login VARCHAR(255)  DECLARE @password VARCHAR(255)  WHILE @counter <= 100000  BEGIN  SET @role\_id = CASE WHEN @counter % 2 = 0 THEN 2 ELSE 3 END  SET @groupname\_id = CASE WHEN @role\_id = 3 THEN CAST(RAND() \* 4 + 1 AS INT) ELSE NULL END  SET @studentIDcard = CASE WHEN @role\_id = 3 THEN @counter + 100002 ELSE NULL END  SET @lastname = 'Lastname' + CAST(@counter AS VARCHAR)  SET @firstname = 'Firstname' + CAST(@counter AS VARCHAR)  SET @patronymic = 'Patronymic' + CAST(@counter AS VARCHAR)  SET @birthday = DATEADD(year, -(@counter % 30 + 18), GETDATE())  SET @login = 'Login' + CAST(@counter AS VARCHAR)  SET @password = 'Password' + CAST(@counter AS VARCHAR)  INSERT INTO users (role\_id, groupname\_id, studentIDcard, lastname, firstname, patronymic, birthday, login, password)  VALUES (@role\_id, @groupname\_id, @studentIDcard, @lastname, @firstname, @patronymic, @birthday, @login, @password)  SET @counter = @counter + 1  END  END |