Содержание

[Введение 5](#_Toc166794040)

[1 Анализ требований к программному средству 6](#_Toc166794041)

[1.1 Аналитический обзор аналогов 6](#_Toc166794042)

[1.1.1 Google Books 6](#_Toc166794043)

[1.1.2 Open Library 7](#_Toc166794044)

[1.2 Разработка функциональных требований 8](#_Toc166794045)

[1.3 Вывод 10](#_Toc166794046)

[2 Разработка архитектуры проекта 11](#_Toc166794047)

[2.1 Обобщенная структура управлением приложения 11](#_Toc166794048)

[2.2 Диаграммы UML, взаимосвязь всех компонентов 11](#_Toc166794049)

[3 Разработка объектов базы данных 12](#_Toc166794050)

[3.1 Таблицы 12](#_Toc166794051)

[3.2 Разработка пользовательских типов базы данных 14](#_Toc166794052)

[3.3 Разработка функций базы данных 15](#_Toc166794053)

[3.4 Разработка процедур базы данных 17](#_Toc166794054)

[3.5 Разработка триггеров базы данных 18](#_Toc166794055)

[3.6 Разработка пользователей базы данных 18](#_Toc166794056)

[3.7 Разработка ролей базы данных 19](#_Toc166794057)

[3.8 Разработка представлений базы данных 19](#_Toc166794058)

[3.9 Разработка индексов базы данных 19](#_Toc166794059)

[3.10 Вывод 20](#_Toc166794060)

[4 Тестирование производительности базы данных 21](#_Toc166794061)

[5 Описание импорта и экспорта JSON 23](#_Toc166794062)

[5.1 Экспорт данных в JSON 23](#_Toc166794063)

[5.2 Импорт данных из JSON 24](#_Toc166794064)

[6 Описание технологии и ее применение 27](#_Toc166794065)

[7 Краткое описание приложения для демонстрации 29](#_Toc166794066)

[7.1 Пример работы приложения от лица администратора 29](#_Toc166794067)

[7.2 Пример работы приложения от лица пользователя 30](#_Toc166794068)

[Заключение 31](#_Toc166794069)

[Список используемых источников 32](#_Toc166794070)

[Приложение A Листинг создания таблиц 33](#_Toc166794071)

[Приложение Б Листинг создания процедур 36](#_Toc166794072)

[Приложение В Листинг создания функций 49](#_Toc166794073)

[Приложение В Листинг создания триггеров 61](#_Toc166794074)

Введение

Цель данной работы заключается в создании реляционной базы данных для системы управления библиотекой, которая обеспечивает пользователя доступом к имеющимся на ней книг. База данных в данном контексте является неотъемлемой частью при разработке. Она позволит хранить, управлять данными и обеспечивать удобный доступ к ним.

Целью данного проекта является разработка реляционной базы данных для системы управления библиотекой книг, которая будет удовлетворять вышеуказанным требованиям и обеспечивать пользователям доступ к имеющимся на платформе книгам.

База данных - это организованное хранилище данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. В данной работе для управления базой данных была выбрана СУБД Oracle SQL. Она представляет собой структурированное хранилище информации, где данные организованы и связаны между собой. Выбранная мною база данных предоставляет эффективное хранение данных, обладает высокой надежностью и производительностью.

Основные требования к базе данных:

* Определение ролей (администратор, пользователь)
* Добавление, удаление и редактирование книг в библиотеке(администратор)
* Возможность указания авторов, жанров, издательств, годов выпуска и других характеристик книг(администратор)
* Поиск книг по названию(пользователь)
* Отслеживание экземпляров книг(администратор)
* Добавление книг в избранное (пользователь)

# 1 Анализ требований к программному средству

1.1 Аналитический обзор аналогов

Важным этапом в разработке, является анализ аналогов - существующий решений и платформ, предлагающих аналогичные функциональные возможности. Это позволяет изучить существующие подходы к управлению библиотечными данными, определить их преимущества и недостатки. На основе такого подхода мы можем так же определить требования и ожидания пользователей. Это позволит определить лучшие практики для дальнейшего внедрения в свой проект.

1.1.1 Google Books

Google Books — это онлайн-платформа, предоставляющая доступ к огромной коллекции книг в электронном формате. Платформа была запущена Google в 2004 году и с тех пор значительно расширилась, включая в себя миллионы книг в различных жанрах, от академической литературы до художественных произведений.

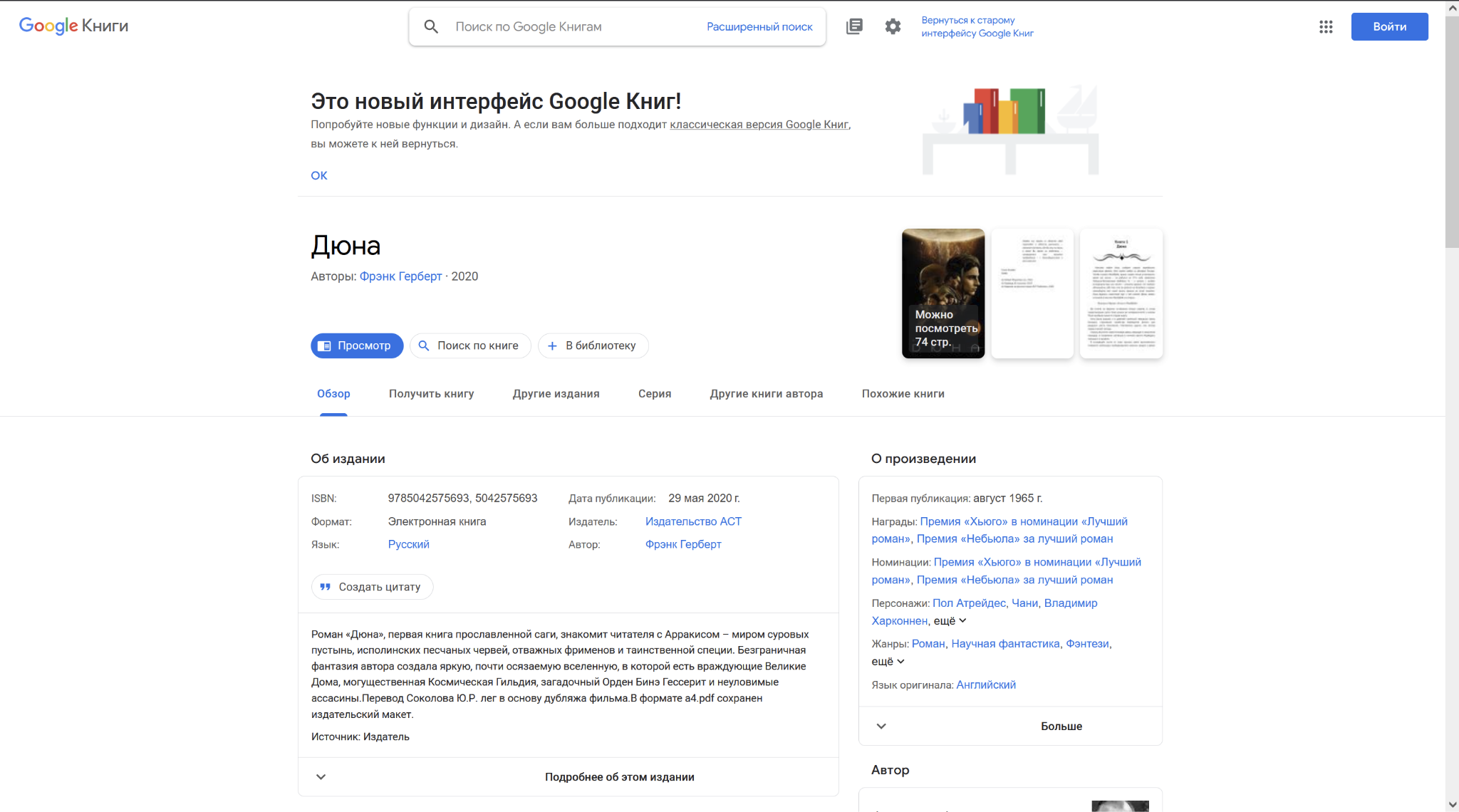


Рисунок 1.1 – Интерфейс Google Books

Google Books предлагает доступ к огромному количеству книг различных авторов и издательств. Возможность поиска современных бестселлеры и классический произведений со всего мира.

Пользователи могут осуществлять поиск книг по различным критериям, таким как название, автор, ключевые слова и т.д. Кроме того, часть книг доступна для онлайн-просмотра, что позволяет пользователям читать их прямо на платформе.

Основные преимущества Google Books:

* Большой выбор книг из различных областей знаний и жанров.
* Удобный интерфейс поиска и просмотра книг.
* Возможность чтения онлайн и загрузки книг для чтения офлайн.
* Бесплатный доступ к многим книгам, а также возможность покупки платных изданий.

Какие технологии и архитектуры Google Books не раскрывают публично, но можно предположить, что они используют распределения базы данных для хранения большого количества данных, для надежности и отказоустойчивости. Также для хранения данных они могут использовать в связке несколько баз данных, для хранения информации об авторах, книг и их информации.

В заключении к обзору данного аналога можно сделать вывод, что данная онлайн-платформа предлагает широкий выбор книг из различных областей знаний и обладает удобным интерфейсом. Однако существуют некоторые ограничения, такие как:

* Ограничения в доступе к книгам в зависимости от региона.
* Не всегда предоставляется возможность скачивания книг в электронном формате.

1.1.2 Open Library

Open Library — это проект интернет-архива, цель которого состоит в создании общедоступной электронной библиотеки, содержащей миллионы книг в различных форматах. Этот проект был запущен в 2006 году, и с тех пор он стал одним из крупнейших и наиболее известных ресурсов для доступа к книгам в электронном формате.

Одними из основных возможностей данного проекта является большой объем книг, как современных так и классический. Возможность читать книги онлайн и скачивать в популярных форматах, таких как PDF и EPUB.

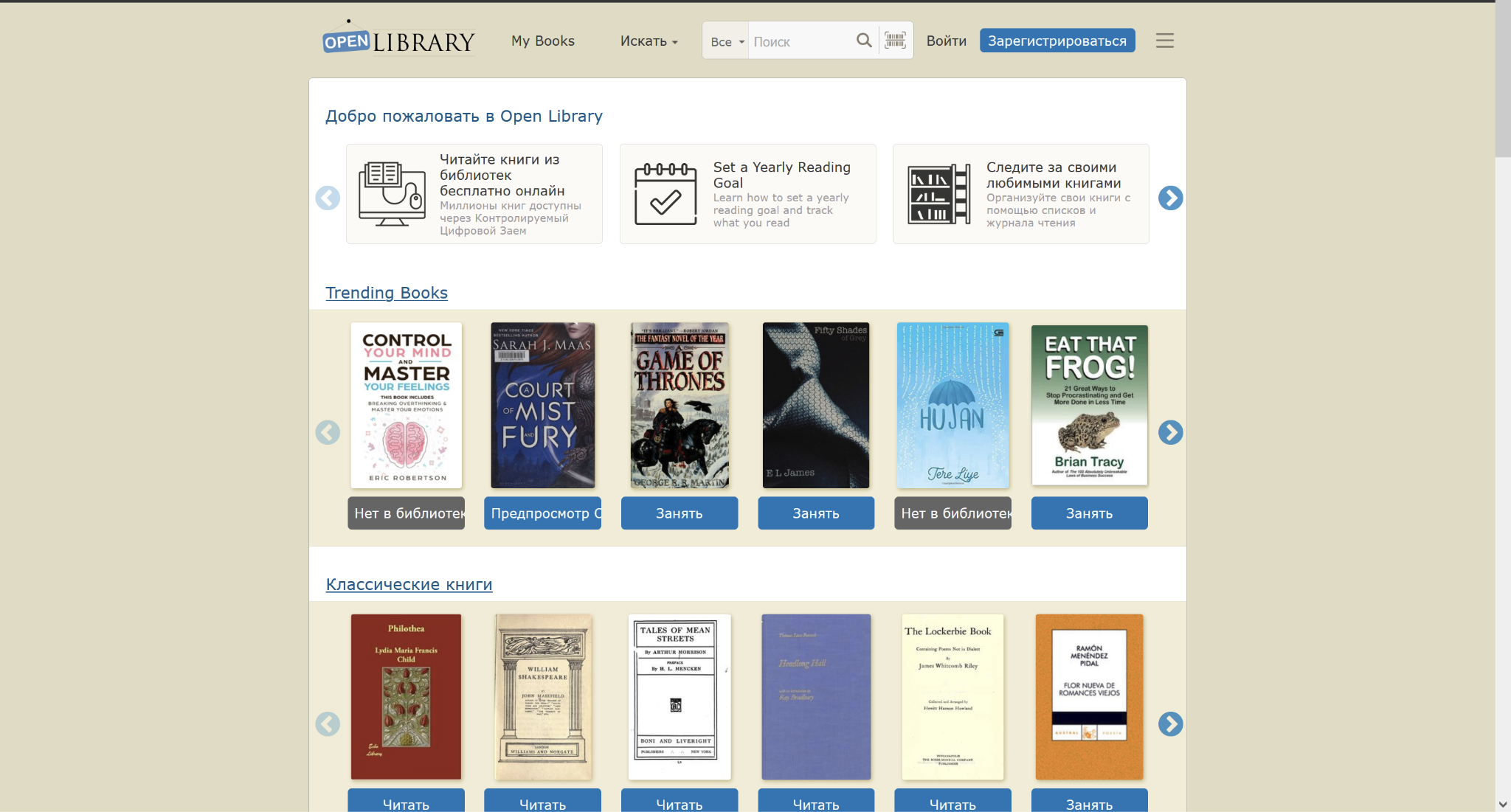


Рисунок 1.2 – Интерфейс Google Books

Open Library позволяет пользователям вносить изменения и дополнять информацию о книгах. Такая возможность дает большое преимущество перед другими аналогами, так как способствует созданию коллективной базы данных о книгах.

Недостатками Open Library также обладает, одним из крупных недостатков является в ограничении к полному тексту некоторых книг из-за авторских прав, но это не мешает быть данной платформе одной из самых популярных на ряду с Google Books.

1.2 Разработка функциональных требований

Разработка функциональных требований играет ключевую роль в процессе создания системы управления библиотекой. Этот этап представляет собой основу для построения базы данных, определяя функциональность и возможности системы, необходимые для эффективного управления библиотечными ресурсами. Это может включать в себя описание того, как данные должны храниться и организовываться, как происходит поиск и выборка данных, каким образом обновляются данные и какие механизмы используются для защиты данных. Кроме того, функциональные требования могут определять интеграцию базы данных с другими системами и программами.

Например, для системы управления библиотекой книг требования следующие:

* Добавлять новые книги.
* Указание основной информации, такой как название, автор, жанр, год издания.
* Для каждой книги должен вестись учет количества доступных экземпляров в библиотеке.
* Система должна поддерживать регистрацию пользователей, включая аутентификацию и управление учетными записями.
* Пользователи должны иметь возможность просматривать информацию о своем аккаунте, просмотр избранных книг и возможность оформлять заказ.
* Система должна обеспечивать возможность поиска книг.

Роли пользователя — это набор прав, которые пользователь может получить в системе.

В данном проекте роли пользователей будут следующими:

* Guest.
* User.
* Manager.
* Admin.

На основе предоставленного списка ролей необходимо построить варианты использование. Варианты использование изображена на рисунке 1.3.



Рисунок 1.2 – Диаграмма вариантов использования Admin

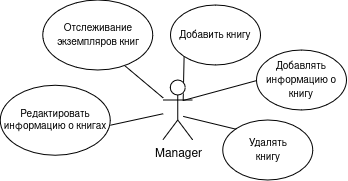


Рисунок 1.3 – Диаграмма вариантов использования Manager

Роль Manager управляет платформой, в его задачи относится следить за экземплярами книг, добавлять и изменять информацию о книга.

Роль Admin имеет полный доступ над системой, следит за ее работой, добавляет авторов книг и менеджеров на платформу.

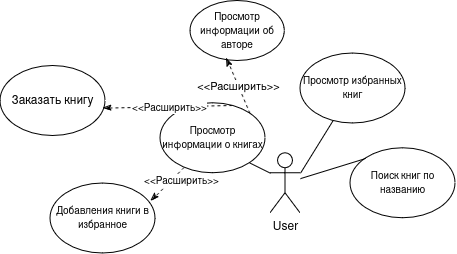


Рисунок 1.4 – Диаграмма вариантов использования User

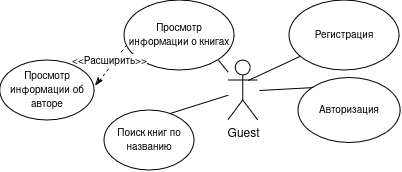


Рисунок 1.5 – Диаграмма вариантов использования Guest

В начале работы с приложением пользователь получает роль Guest, ему будет доступно просмотр базовой информации. После регистрации ему пользователя, он получает роль User.

User имеет возможности, просматривать и заказывать книгу, также может добавлять в свою библиотеку понравившиеся книги.

1.3 Вывод

В процессе проведения аналитического обзора были рассмотрены такие аналоги как Google Books и Open Library, которые уже существуют на рынке давно. Этот анализ позволил определить основные характеристики и функциональные возможности, которые необходимо предусмотреть при разработке собственного проекта. Также были определены ключевые функциональные требования, которые необходимы для управления библиотечными ресурсами. Была разработана диаграмма вариантов использования, на которой отображены основные функции ролей на платформе.

# 2 Разработка архитектуры проекта

2.1 Обобщенная структура управлением приложения

Разработка архитектуры проекта играет ключевую роль в обеспечении успешной реализации системы управления библиотекой книг. Для этого необходимо разработать функционал, который предоставляет возможности пользователю управлять данными. В частности для платформы, которая предоставляет возможности работать с книгами. Это может включать в себя разработку оптимизированных запросов, разработка механизма для извлечения и обработки информации из базы данных.

2.2 Диаграммы UML, взаимосвязь всех компонентов

Диаграмма базы данных таблиц (Database Table Diagram) - это визуальное представление структуры базы данных и отношений между таблицами, которые хранятся в этой базе данных. Диаграмма базы данных будет представлена на рисунке 2.1.

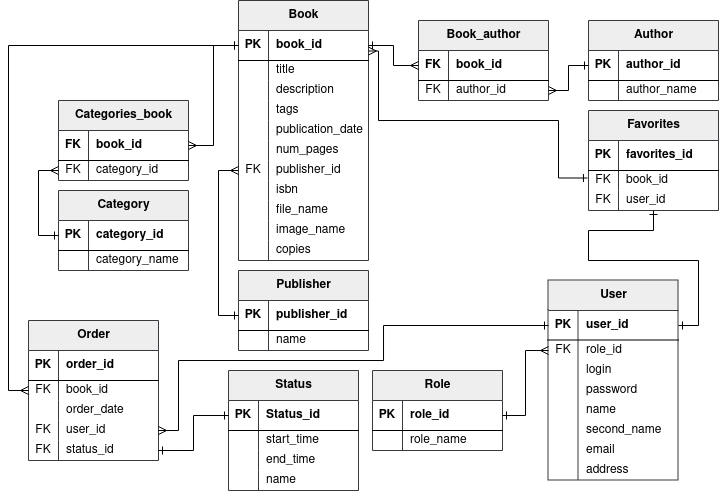


Рисунок 2.1 – Диаграммы базы данных

# 3 Разработка объектов базы данных

3.1 Таблицы

Для реализации базы данных «Система управлением библиотеки книг» было разработано 11 таблиц: Book, Book\_author, Author, Categories\_Book, Category, Order, Publisher, Status, Role, User, Favorites.

Таблица Book представляет книгу в библиотеки, состоит из столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы Book

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| book\_id | Идентификатор книги | NVARCHAR2(255) |
| title | Название книги | NVARCHAR2(255) |
| description | Описание книги | NVARCHAR2(255) |
| publication\_date | Дата публикации книги | DATE |
| publisher\_id | Идентификатор издателя книги | NVARCHAR2(255) |
| isbn | ISBN книги | NVARCHAR2(255) |
| tags | Теги книги | NVARCHAR2(60) |
| copies | Количество копий книги | INTEGER |
| num\_pages | Количество страниц книги | INTEGER |
| file\_name | Имя файла с содержанием книги | NVARCHAR2(255) |
| image\_name | Имя файла с обложкой книги | NVARCHAR2(255) |
| publisher\_id | Идентификатор издателя книги | NVARCHAR2(255) |

Таблица Author представляет список отзывов о магазине состоит из столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы Author

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| author\_id | Идентификатор автора | NVARCHAR2(255) |
| name | Имя автора | NVARCHAR2(255) |

Таблица Book\_author - связующая таблица между книгами и их авторами (таблица 3.2):

Таблица 3.2 – Столбцы таблицы Book\_author

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| author\_id | Идентификатор автора | NVARCHAR2(255) |
| book\_id | Идентификатор книги | NVARCHAR2(255) |

Таблица Category содержит информацию о категориях (таблица 3.3):

Таблица 3.3 – Столбцы таблицы Category

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| category\_id | Идентификатор категории | NVARCHAR2(255) |
| name | Название категории | NVARCHAR2(255) |

Таблица Categories\_book - связующая таблица между книгами и их категориями (таблица 3.4):

Таблица 3.4 – Столбцы таблицы Categories\_book

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| category\_id | Идентификатор категории | NVARCHAR2(255) |
| book\_id | Идентификатор книги | NVARCHAR2(255) |

Таблица Publisher содержит информацию об издателе(таблица 3.5):

Таблица 3.5 – Столбцы таблицы Publisher

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| publisher\_id | Идентификатор издателя | NVARCHAR2(255) |
| name | Имя издателя | NVARCHAR2(255) |

Таблица User содержит информацию об издателе(таблица 3.6):

Таблица 3.6 – Столбцы таблицы User

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| user\_id | Идентификатор пользователя | NVARCHAR2(255) |
| login | Логин пользователя | NVARCHAR2(255) |
| password | Пароль пользователя (хэшированный) | RAW(64) |
| role\_id | Идентификатор роли | NVARCHAR2(255) |
| name | Имя пользователя | NVARCHAR2(255) |
| second\_name | Фамилия пользователя | NVARCHAR2(255) |
| email | Email пользователя | NVARCHAR2(255) |
| address | Адрес пользователя | NVARCHAR2(255) |

Таблица Favorites содержит информацию об издателе(таблица 3.7):

Таблица 3.7 – Столбцы таблицы Favorites

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| user\_id | Идентификатор пользователя | NVARCHAR2(255) |
| book\_id | Идентификатор книги | NVARCHAR2(255) |

Таблица Order содержит информацию об издателе(таблица 3.8):

Таблица 3.8 – Столбцы таблицы Order

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| order\_id | Идентификатор заказа | NVARCHAR2(255) |
| user\_id | Идентификатор пользователя | NVARCHAR2(255) |
| order\_date | Дата заказа | DATE |
| book\_id | Идентификатор книги | NVARCHAR2(255) |
| status\_id | Идентификатор статуса | NVARCHAR2(255) |

Таблица Status содержит информацию об издателе(таблица 3.9):

Таблица 3.9 – Столбцы таблицы Status

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| status\_id | Идентификатор статуса | NVARCHAR2(255) |
| start\_time | Дата и время начала статуса | DATE |
| end\_time | Дата и время завершения статуса | DATE |
| name | Имя статуса | NVARCHAR2(255) |

Таблица Status содержит информацию об издателе(таблица 3.9):

Таблица 3.9 – Столбцы таблицы Status

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| role\_id | Идентификатор статуса | NVARCHAR2(255) |
| name | Имя роли | NVARCHAR2(255) |

3.2 Разработка пользовательских типов базы данных

Разработка пользовательских типов данных в Oracle SQL предоставляет возможность создавать собственные структуры данных. Это позволяет улучшить управление данными, упростить структурирование информации и повысить производительность запросов. В данном проекте было разработано 5 пользовательских типов данных:

* author\_info\_type - тип данных, который служит для хранения и отображения информации об авторе и его книгах.
* book\_result\_type - тип данных, который служит для хранения и отображения информации о книгах, ее авторов и категории.
* category\_info\_type - тип данных, который служит для хранения и отображения информации о категории и ее книгах.
* order\_with\_status\_type - тип данных, который служит для хранения и отображения информации о заказе и его статусе.
* user\_info\_type - тип данных, который служит для хранения и отображения информации о пользователе.

Реализация нескольких из пользовательских типов данных представлено в листинге 3.1. Остальные пользовательские типы будут аналогичны, работая с другими данными.

|  |
| --- |
| -- Пользовательский тип данных, который хранит информацию о заказах, книги и статусе.  create or replace TYPE order\_with\_status\_type AS OBJECT (  order\_id NVARCHAR2(255),  user\_id NVARCHAR2(255),  book\_id NVARCHAR2(255),  order\_date DATE,  status\_name NVARCHAR2(255),  start\_time DATE,  end\_time DATE  );  -- Пользовательская таблица которая состоит из пользовательского типа данных.  create or replace TYPE order\_with\_status\_table AS TABLE OF order\_with\_status\_type;  -- Пользовательский тип данных, который хранит информации о книгах вместе с авторам и категорией  create or replace TYPE book\_result\_type AS OBJECT (  book\_id NVARCHAR2(255),  title NVARCHAR2(255),  description NVARCHAR2(255),  publication\_date DATE,  publisher\_id RAW(16),  isbn NVARCHAR2(255),  tags NVARCHAR2(60),  copies INTEGER,  num\_pages INTEGER,  file\_name NVARCHAR2(255),  image\_name NVARCHAR2(255),  author\_name NVARCHAR2(255),  category\_name NVARCHAR2(255)  );  -- Пользовательская таблица которая состоит из пользовательского типа данных.  create or replace TYPE category\_info\_table AS TABLE OF category\_info\_type; |

Листинг 3.1 – Пользовательский тип данных order\_with\_status\_type и book\_result\_type

## 3.3 Разработка функций базы данных

Для управления данными через приложение все пользователи использует функции. Функция – объект базы данных, представляющий собой набор SQL-инструкций, который компилируется один раз и хранится на сервере. В моём приложении реализовано 17 функций.

* GET\_ALL\_AUTHORS –функция получения авторов.
* GET\_ALL\_BOOKS – функция получения книг.
* GET\_ALL\_CATEGORIES – функция получения категорий.
* GET\_ALL\_USERS\_INFO\_VIEW – функция получения пользователей
* GET\_AUTHOR\_BY\_ID – функция получения автора.
* GET\_BOOK\_BY\_ID – функция получения книг.
* GET\_BOOKS\_BY\_AUTHOR – функция получения книг автора.
* GET\_BOOKS\_BY\_CATEGORY – функция получения книг категории.
* GET\_BOOKS\_BY\_PUBLISHER – функция получения книг издателя.
* GET\_CATEGORY\_BY\_ID – функция получения категории.
* GET\_FAVORITE\_BOOKS\_BY\_USER – функция получения любимых книг
* GET\_ORDER\_BY\_ID – функция получения заказа.
* GET\_ORDERS\_BY\_USER\_OR\_BOOK – функция получения заказа по пользователя или книге.
* GET\_ORDERS\_WITH\_STATUS – функция получения заказа со статусом.
* GET\_USER\_BY\_ID – функция получения пользователя.
* GET\_USERS\_INFO – функция получения всех пользователей.
* HASH\_PASSWORD – функция хеширования пароля.
* SEARCH\_BOOKS – функция поиска книги.
* VALIDATE\_PASSWORD – функция проверки пароля

Пример создания некоторых функций представлен на листинге 3.2.

|  |
| --- |
| -- Функция хеширования пароля.  create or replace FUNCTION hash\_password (  p\_password IN NVARCHAR2  ) RETURN NVARCHAR2  AS  p\_hash\_raw RAW(32);  p\_hash\_hex NVARCHAR2(255);  BEGIN  p\_hash\_raw := DBMS\_CRYPTO.HASH(UTL\_I18N.STRING\_TO\_RAW(p\_password, 'AL32UTF8'), DBMS\_CRYPTO.HASH\_SH256);  p\_hash\_hex := UTL\_ENCODE.base64\_encode(p\_hash\_raw);  RETURN p\_hash\_hex;  END;  -- Функция проверки пароль на валидность.  create or replace FUNCTION validate\_password (  p\_password IN NVARCHAR2,  p\_hashed\_password IN NVARCHAR2  ) RETURN BOOLEAN  AS  l\_hash RAW(32);  l\_hash\_hex NVARCHAR2(255);  BEGIN  l\_hash := DBMS\_CRYPTO.HASH(  UTL\_I18N.STRING\_TO\_RAW(p\_password, 'AL32UTF8'),  DBMS\_CRYPTO.HASH\_SH256  );  l\_hash\_hex := UTL\_ENCODE.base64\_encode(l\_hash);  RETURN l\_hash\_hex = p\_hashed\_password;  END; |

Листинг 3.2 – Функции хеширования пароля и проверки пароля на валидность

3.4 Разработка процедур базы данных

По большому счёту, процедуры выполняют аналогичные действия с функциями. Всего было разработано в проекте 22 процедуры. Они использовались для создания, редактирования и удаление данных в моей базе данных.

* ADD\_FAVORITE\_BOOK – процедура добавления книги в таблицу Favorite.
* AUTHENTICATE\_USER – процедура проверки на аунтефикацию пользователя.
* CANCEL\_ORDER – процедура отмены заказа.
* CHANGE\_ORDER\_STATUS – процедура смены статуса заказа.
* CHANGE\_PASSWORD – процедура изменения пароля пользователя.
* CREATE\_AUTHOR – процедура создания автора.
* CREATE\_BOOK – процедура создания книги.
* CREATE\_CATEGORY – процедура создания книги.
* CREATE\_ORDER\_STATUS – процедура создания заказа со статусом.
* CREATE\_PUBLISHER – процедура создания издателя.
* CREATE\_USER – процедура создания пользователя.
* DELETE\_AUTHOR\_SET\_NULL – процедура удаления автора.
* DELETE\_BOOK\_SET\_NULL – процедура удаления книги.
* DELETE\_CATEGORY\_SET\_NULL – процедура удаления категории.
* DELETE\_FAVORITES – процедура удаления книги из таблицы Favorite.
* DELETE\_PUBLISHER – процедура удаления издателя.
* LINK\_BOOK\_AUTHOR – процедура для создания связи автора и книги.
* LINK\_BOOK\_CATEGORY – процедура для создания категории.
* UPDATE\_AUTHOR – процедура обновления автора.
* UPDATE\_BOOK – процедура обновления книги.
* UPDATE\_CATEGORY – процедура обновления категории.
* UPDATE\_PUBLISHER – процедура обновления издателя.
* UPDATE\_USER – процедура обновления пользователя.

На листинге 3.3 представлен пример реализации одной из процедур.

|  |
| --- |
| -- Процедура создания автора.  create or replace PROCEDURE create\_author (  p\_name IN NVARCHAR2  )  AS  p\_author\_id NVARCHAR2(255);  BEGIN  p\_author\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());  INSERT INTO Author (author\_id, name)  VALUES (p\_author\_id, p\_name);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Листинг 3.3 – Процедура создания автора

Данная процедура создает автора, входным параметром принимает имя автора, в процедуре генерируется GUID, который используется мною в проекте для генерации идентификаторов.

3.5 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.

|  |
| --- |
| -- Триггер на валидацию кол-во копий книг.  create or replace TRIGGER check\_book\_copies  BEFORE UPDATE OR INSERT ON Book  FOR EACH ROW  BEGIN  IF :new.copies < 0 THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Количество копий книги не может быть меньше или равно нулю.');  END IF;  END; |

Листинг 3.4 – Триггер на валидное количество копий книг

Разработанные триггеры в базе данных:

* check\_book\_copies – триггер на проверку целостности копий книг.
* check\_num\_pages – триггер на проверку целостности страниц книги.
* check\_title\_length – триггер на проверку длинны названия книги.
* check\_end\_time – триггер на проверку добавления правильной даты.
* check\_login\_length – тригер на проверку длинны логина пользователя.

В своей проекте я использовал триггеры для поддержки целостности данных, одна они так же могут использоваться для контроля доступа к данным и автоматической обработки данных при выполнении определенных операций в таблице. В листинге 3.4 приведен код одного из триггеров.

3.6 Разработка пользователей базы данных

В этом разделе создаются роли для ограничения доступа к базе данных. Создание ролей позволяет установить границы доступа к различным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации.

Будут созданы 4 роли для разграничения доступа к базе данных: user, manager, admin и guest. Это позволит ограничить доступ к определенным функциям базы данных и предотвратить несанкционированный доступ к конфиденциальной информации. Роли user\_role и manager\_role являются основными, в то время как роль admin будет иметь более широкий набор привилегий.

Роль admin имеет полный доступ ко всей базе данных и может выполнять любые операции, в том числе создавать и изменять таблицы, индексы, представления и триггеры. Выданные привилегии роли Admin можно увидеть на листинге 3.5.

|  |
| --- |
| -- Создание ролей базы данных.  CREATE USER ADMIN\_lib\_pdb IDENTIFIED BY 1234;  ALTER USER ADMIN\_lib\_pdb ACCOUNT UNLOCK;  CREATE USER MANAGER\_lib\_pdb IDENTIFIED BY 1234;  ALTER USER MANAGER\_lib\_pdb ACCOUNT UNLOCK;  CREATE USER USER\_lib\_pdb IDENTIFIED BY 1234;  ALTER USER USER\_lib\_pdb ACCOUNT UNLOCK;  CREATE USER GUEST\_lib\_pdb IDENTIFIED BY 1234;  ALTER USER GUEST\_lib\_pdb ACCOUNT UNLOCK; |

Листинг 3.5 – Создание пользователей базы данных

В дальнейшем для созданных пользователей были выданы необходимые для функционала привилегии. С помощью GRANT EXECUTE ON можно выдавать привилегии пользователям, которые работают в базе данных.

3.7 Разработка ролей базы данных

Создания ролей базы данных является одним из главных аспектов безопасности базы данных. Они контролируют доступ к объектам базы данных и позволяют ограничить к ним доступ неизвестным пользователем базы данных.

В своем проекте я создал роль conn\_role на подключения к базе данных, что бы обезопасить доступ неизвестным пользователям к моей PDB в которой хранятся объекты базы данных.

3.8 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 было создано одно представления user\_info\_view, которое предоставляет пользователем данные без наличия пароля, что уменьшает риски утечки информации.

3.9 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. В моем проекте было разработано 3 индекса:

|  |
| --- |
| CREATE INDEX idx\_book\_title ON Book (UPPER(title));  CREATE INDEX idx\_author\_name ON Author (UPPER(name));  CREATE INDEX idx\_category\_name ON Category (UPPER(name)); |

Листинг 3.6 – Разработка индексов

3.10 Вывод

В данном разделе были рассмотрены основные этапы настройки и использования Oracle. Были описаны процедуры создания таблиц, ролей для разграничения доступа к базе данных и процедур для выполнения различных операций с данными.

Можно сделать вывод, что Oracle является мощной и надежной системой управления базами данных, которая может быть использована для хранения и обработки больших объемов данных. Правильная установка и настройка Oracle, а также оптимизация запросов, позволят обеспечить высокую производительность и удобное поддержание базы данных разработчиком.

4 Тестирование производительности базы данных

В процессе разработки баз данных одной из важнейших задач является тестирование их производительности. Такие проверки позволяют оценить, насколько эффективно база данных обрабатывает запросы и возвращает результаты. Это важно для обеспечения высокой скорости работы приложения и выявления возможных проблем или узких мест, которые могут привести к задержкам или сбоям. Оценка производительности позволяет определить эффективность базы данных и принять меры для ее оптимизации.

Для проверки производительности базы данных необходимо заполнить ее большим количеством различных данных и узнать время выполнения одного запроса.

Для данного тестирования мы создаем анонимный блок с циклом на 100000 строк. Данный блок будет заполнять таблицу большим количеством данных. Чем меньше скорость выполнения данного блока, тем эффективна разработана база данных для поддержания большого количества строк в таблице(листинг 5.1).

|  |
| --- |
| DECLARE  v\_category\_id NVARCHAR2(255);  v\_name NVARCHAR2(255);  BEGIN  FOR i IN 1 .. 100000 LOOP  v\_name := 'Book test ' || i;  create\_book(v\_name, p\_isbn => i);  END LOOP;  COMMIT;  END; |

Листинг 4.1 – Заполнение таблицы 100000 строк

Для получения данных можно использовать раннее разработанную мною процедур для поиска книг(листинг 4.2).

|  |
| --- |
| SELECT \* FROM TABLE(search\_books('24')); |

Листинг 4.3 – Запрос к таблице Категории

Результат выполнения выборки данных из таблиц можно посмотреть выполнив запрос можно увидеть на рисунке 4.1.

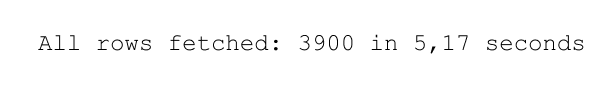


Рисунок 4.1 –получения строк без индекса

Теперь создадим индекс для тестирования запроса (листинг 4.4).

|  |
| --- |
| CREATE INDEX idx\_book\_title ON Book (UPPER(title));  CREATE INDEX idx\_author\_name ON Author (UPPER(name)); |

Листинг 4.4 – Запрос на показ плана выполнения

Результат выполнения запроса на анализ содержащий код в листинге 4.4, представлен можно увидеть на рисунке 4.2.

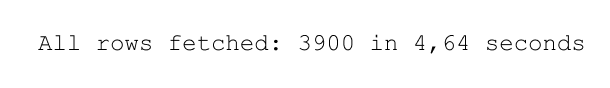


Рисунок 4.2 –получения строк с использованием индекса

После проведенного тестирования, можно убедить, что использования индекса в базе данных для выборки большого количества строк является важной частью базы данных. Он существенно сокращает выполнение запросов на получение строк из таблиц.

5 Описание импорта и экспорта JSON

Для экспорта данных в формате JSON использовались возможности, которые предоставляет нам sqldeveloper. В нем есть встренный эспорт данных.

5.1 Экспорт данных в JSON

Что бы произвести экспорт данных в json формат нам надо правой кнопкой мыши вызвать меню таблицы в базе данных и выбрать пункт Export, на рисунке 5.1 представлено меню таблицы.

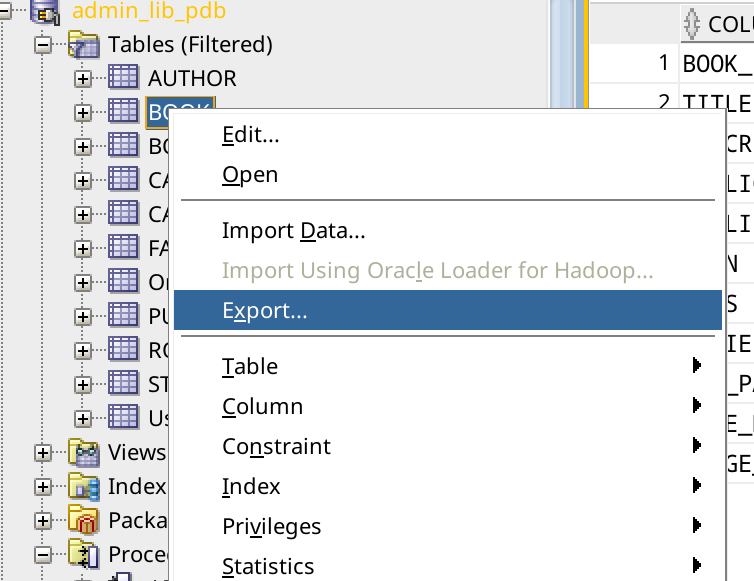


Рисунок 5.1 – Меню таблицы

Далее в появившемся окне, мы указываем формат JSON и путь куда мы экспортируем данные с указанием названия файла и формата, на рисунке 5.2 показана настройка экспорта.

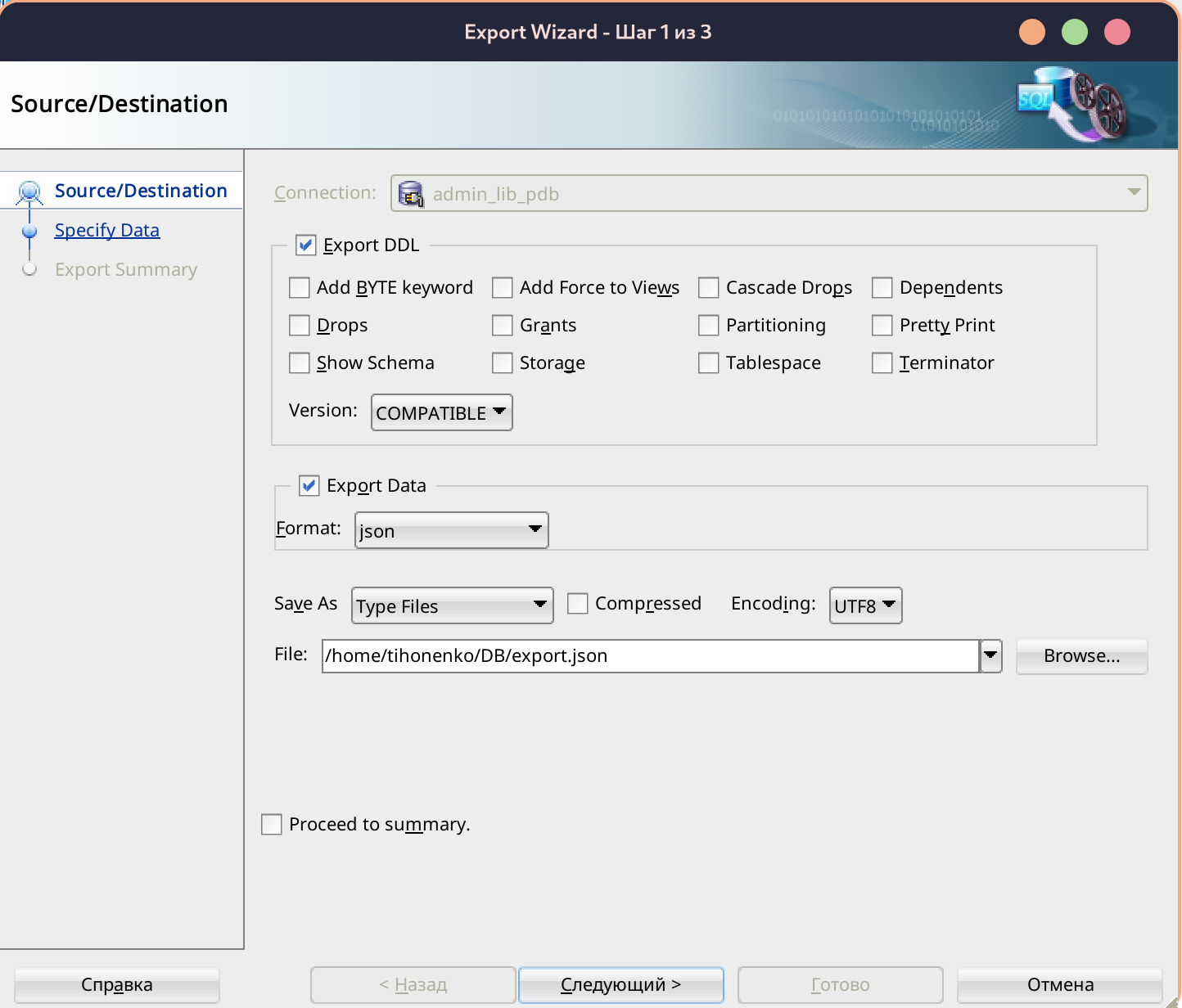


Рисунок 5.2 – Настройка экспорта

После выполнения у нас создается файл export.json, который содержит данные таблицы в формате JSON

5.2 Импорт данных из JSON

Для импорта JSON-данных в базу данных Oracle можно воспользоваться функцией JSON\_VALUE, которая позволяет преобразовать JSON-данные в строки и столбцы таблицы. Была использована функция MERGE для предотвращения вставки идентичных данных. Процедура для импорта категорий представлена в листинге 5.1.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE IMPORT\_CATEGORIES\_FROM\_JSON  IS  v\_file UTL\_FILE.FILE\_TYPE;  v\_line VARCHAR2(32767);  v\_json CLOB := EMPTY\_CLOB();  BEGIN  -- Открытие файла  v\_file := UTL\_FILE.FOPEN('JSON\_DIR', 'category.json', 'R');  LOOP  BEGIN  UTL\_FILE.GET\_LINE(v\_file, v\_line);  v\_json := v\_json || v\_line;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  EXIT;  END;  END LOOP;  UTL\_FILE.FCLOSE(v\_file);  FOR rec IN (  SELECT  category\_id,  name  FROM JSON\_TABLE(  v\_json, '$[\*]'  COLUMNS (  category\_id VARCHAR2(50) PATH '$.category\_id',  name VARCHAR2(255) PATH '$.name'  )  )  ) LOOP  MERGE INTO Category c  USING (  SELECT  rec.category\_id AS category\_id,  rec.name AS name  FROM DUAL  ) json\_data  ON (c.category\_id = json\_data.category\_id)  WHEN NOT MATCHED THEN  INSERT (category\_id, name)  VALUES (json\_data.category\_id, json\_data.name)  WHEN MATCHED THEN  UPDATE SET  name = json\_data.name;  END LOOP;    EXCEPTION  WHEN OTHERS THEN  IF UTL\_FILE.IS\_OPEN(v\_file) THEN  UTL\_FILE.FCLOSE(v\_file);  END IF;  RAISE;  END; |

Листинг 5.1 Функция импорта данных из JSON

В данной главе была представлена реализация импорта и экспорта данных в JSON формате. С помощью этих методов можно удобно применять данные в сложных структурах.

6 Описание технологии и ее применение

Технология email уведомлений при изменении базы данных (БД) обычно используется для автоматической отправки уведомлений на электронную почту при определенных событиях или изменениях данных в БД. Благодаря данной технологии мы можем получать информацию на почту о событиях, которые происходят с нашей базой данных, а также отсылать ее пользователем что бы поделиться новостями, информации о заказах.

В своем проекте я реализовал рассылку сообщений на почту о событиях, которые происходят с нашей базой данных. При добавлении, редактировании или удаление данных происходит отправка сообщений на почту.

Для реализации email рассылки я использовал smtp.yandex.ru сервис и простое приложение, которое вызывает некоторые процедуры с базы данных. В листинге 6.1 представлен код функции для отправки сообщения на почту.

|  |
| --- |
| sendEmailYandex = async (message: string, subject: string) => {  try {  let transporter = nodemailer.createTransport({  service: 'yandex',  auth: {  user: 'tihon.24.06',  pass: YANDEX\_PASS,  },  });  let mailOptions = {  from: 'tihon.24.06@yandex.ru',  to: 'nikireatz@gmail.com',  subject: subject,  text: message,  };  let info = await transporter.sendMail(mailOptions);  console.log('Email sent: ', info.response);  } catch (error) {  console.error('Error sending email: ', error);  }  }; |

Листинг 6.1 – Email уведомления на TypeScript

Вызов метода отправки письма показан в листинге 6.2.

|  |
| --- |
| createCategory = async (req: Request, res: Response) => {  try {  const { name } = req.body;  if (!name) {  return res.status(400).json({ message: 'Name is required' });  }  const category: ICategory = {  name,  };  const result = await this.\_service.createCategory(category);  if (result.code === 200) {  this.\_email.sendEmailYandex(  `была создана новая категория ${category.name}`,  'новые данные в таблице Category'  );  }  return res.status(result.code).json({  code: result.code,  message: result.message,  data: result.data,  });  } catch (err) {  console.log(' Error in createAuthor: ', err);  return res.status(500).json({ message: 'Internal server error' });  }  }; |

Листинг 6.2 – Вызов метода sendEmailYandex

На рисунке 6.1 показан результат выполенния нашего метода sendEmailYandex

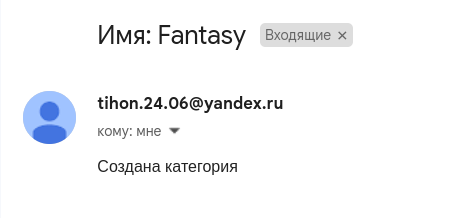
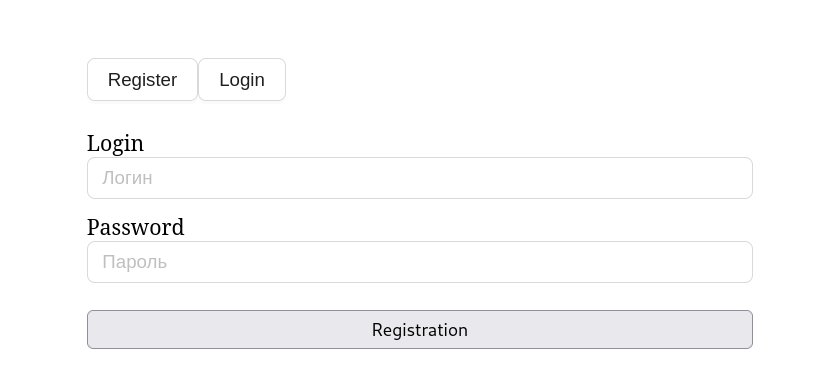


Рисунок 6.1 Результат выполнения метода sendEmailYandex

7 Краткое описание приложения для демонстрации

7.1 Пример работы приложения от лица администратора

Когда пользователь открывает приложение, на главном экране его просят авторизироваться. Форма авторизации представлена на рисунке 7.1.

Рисунок 7.1 - форма авторизации

После авторизации мы попадает в админ-панель. В данном приложении представлена простая небольшая реализация для работы с базой данных. В основном приложение требуется для демонстрации работы с email-рассылкой.

Возможности администратора показаны на рисунке 7.2

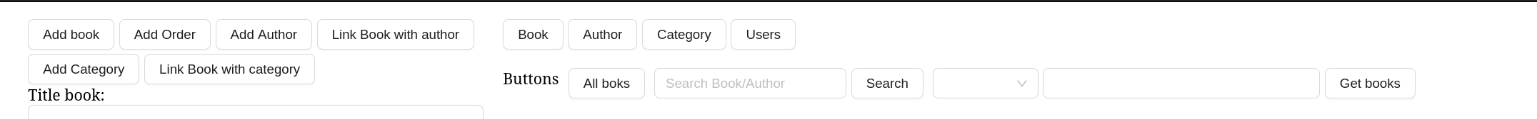


Рисунок 7.2 – Админ-панель

На рисунке 7.3 показан пример работы с админ-панелью на примере добавления книги. Также на изображении можно увидеть еще некоторые важные действия с данными, такие как добавления автора или категории к книге, создания заказа и так далее. Так же присутствуют кнопки на карточка, которые позволяют удалить или изменить книгу.

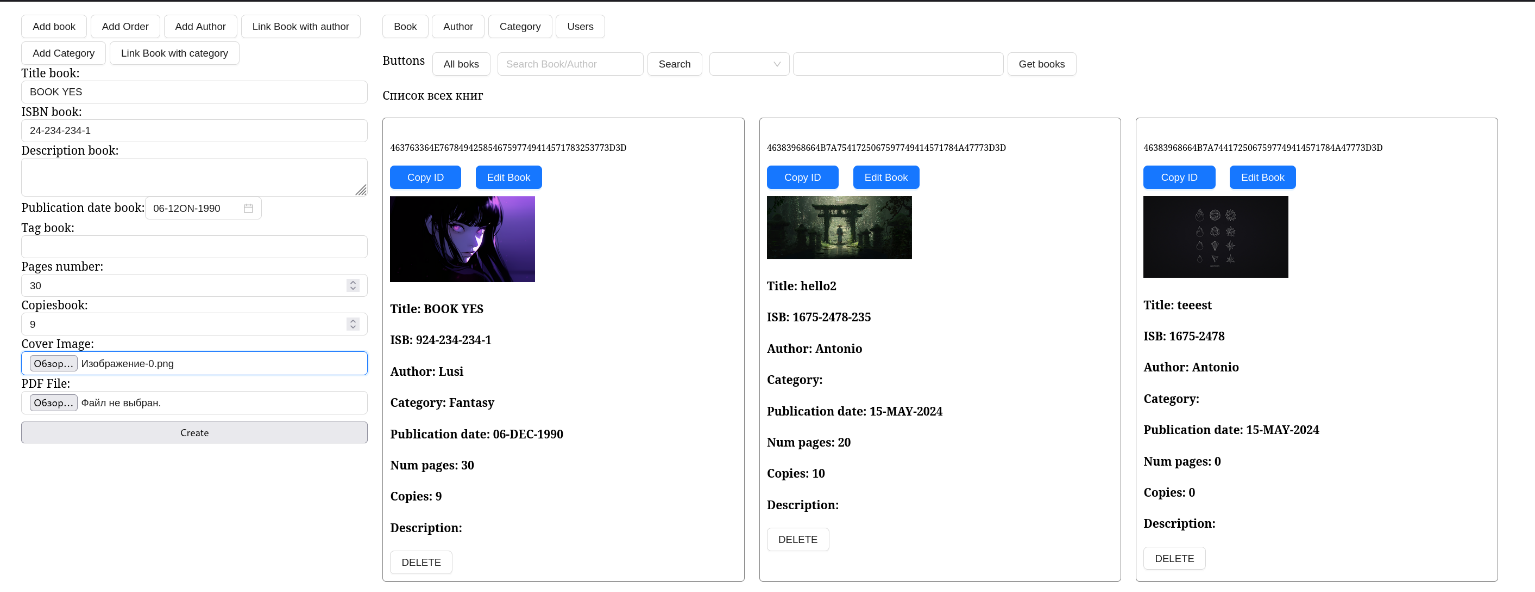


Рисунок 7.3 - Добавление книги

7.2 Пример работы приложения от лица пользователя

Если в приложение зайти под роль user, то пользователь попадет на страницу просмотр книг и добавления их в избранное, просмотр избранного и заказов, за исключением возможности удалить книгу, категорию или автора из базы данных. На рисунке 7.4 представлена страница с книгами для пользователя.

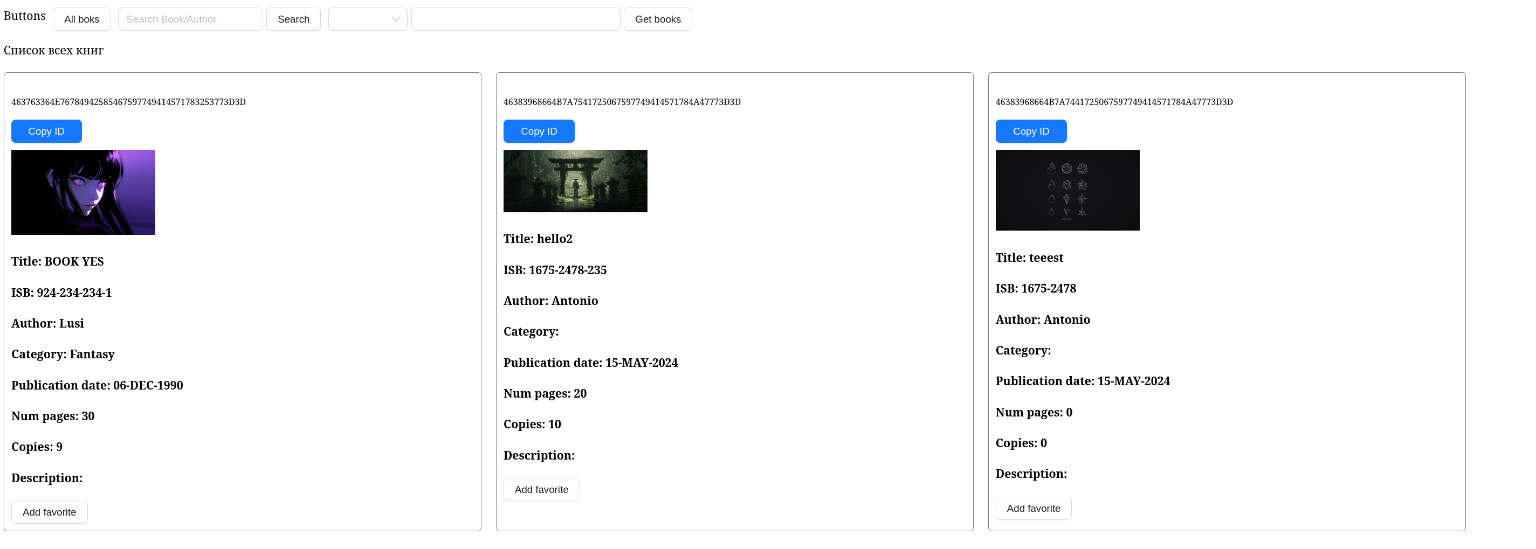


Рисунок 7.4 - Страница книг

Как видно на изображении пользователь может искать книги, добавлять их в избранное, так же есть поиск книг по название или имени автору, выборка книг по определенному автору или категории.

Заключение

В проекте была реализована база данных для управления библиотекой, которая включает создание заказа на книгу, управление книгами, авторами, категориями и пользователями. Были реализованы процедуры и функции комфортной манипуляциями данными. Важной частью проекта была реализация email-рассылки на события с базой данной. Были реализованы триггеры для проверки целостности данных, которые добавляют в таблицы. Реализация представлений и пользовательских типов помогло обеспечить уровень доступа данных при выполнении запросов на их получение. Были выданы привилегии и роли пользователям, которые могут взаимодействовать с объектами базы данных. Это позволило обеспечить доступ к тем объектам, которые нужны определенным пользователям. Главным аспектов эффективной базы данных является ее производительность и в рамках проекта, было реализовано тестирование на поиск данных, добавлены индексы для повышения скорости работы.

Список используемых источников

1. Работа с Oracle SQL [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/en/database/oracle/oracle-database/index.html> – Дата доступа: 27.04.2024.
2. Шифрование данных в Oracle [Электронный ресурс]. – Режим доступа: <https://www.nazmulhuda.info/encryption-decryption-using-dbms_crypto>. – Дата доступа: 30.04.2023.
3. Работа с данными формата JSON в Oracle [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/en/database/oracle/oracle-database/19/adjsn/loading-external-json-data.html> – Дата доступа: 2.05.2023.
4. Поставщик данных Oracle для .NET [Электронный ресурс]. – Режим доступа: <https://www.oracle.com/cis/database/technologies/appdev/dotnet/odp.html>. – Дата доступа: 28.04.2023.
5. Настройка почтовых программ [Электронный ресурс]. - Режим доступа: <https://yandex.ru/support/mail/mail-clients/others.html>. - Дата доступа: 30.04.2023.
6. Node Oracle db подключение Oracle к node - [Электронный ресурс]. - Режим доступа: <https://node-oracledb.readthedocs.io/en/latest/user_guide/introduction.html>. - Дата доступа: 27.04.2024

# Приложение A Листинг создания таблиц

|  |
| --- |
| --CREATE TABLE      ---- BOOK ----  CREATE TABLE Book (  book\_id NVARCHAR2(255) PRIMARY KEY,  title NVARCHAR2(255) NOT NULL UNIQUE,  description NVARCHAR2(255),  publication\_date DATE,  publisher\_id NVARCHAR2(255),  isbn NVARCHAR2(255) NOT NULL UNIQUE,  tags NVARCHAR2(60),  copies INTEGER,  num\_pages INTEGER,  file\_name NVARCHAR2(255),  image\_name NVARCHAR2(255)  );    ---- AUTHOR ----  CREATE TABLE Author (  author\_id NVARCHAR2(255) PRIMARY KEY,  name NVARCHAR2(255) NOT NULL UNIQUE  );    ---- BOOK\_AUTHOR ----  CREATE TABLE Book\_Author (  book\_id NVARCHAR2(255),  author\_id NVARCHAR2(255),  PRIMARY KEY (book\_id, author\_id)  );    ---- CATEGORY ----  CREATE TABLE Category (  category\_id NVARCHAR2(255) PRIMARY KEY,  name NVARCHAR2(255) NOT NULL UNIQUE  );    ---- CATEGORY\_BOOK ----  CREATE TABLE Categories\_Book (  book\_id NVARCHAR2(255),  category\_id NVARCHAR2(255),  PRIMARY KEY (book\_id, category\_id)  );    ---- PUBLISHER ----  CREATE TABLE Publisher (  publisher\_id NVARCHAR2(255) PRIMARY KEY,  name NVARCHAR2(255) NOT NULL UNIQUE  );    ---- USER ----  CREATE TABLE "User" (  user\_id NVARCHAR2(255) PRIMARY KEY,  login NVARCHAR2(255) NOT NULL UNIQUE,  password RAW(64) NOT NULL,  role\_id NVARCHAR2(255),  name NVARCHAR2(255) NOT NULL,  second\_name NVARCHAR2(255),  email NVARCHAR2(255),  address NVARCHAR2(255)  );    ---- FAVORITES ----  CREATE TABLE Favorites (  user\_id NVARCHAR2(255),  book\_id NVARCHAR2(255),  PRIMARY KEY (user\_id, book\_id)  );    ---- ORDER ----  CREATE TABLE "Order" (  order\_id NVARCHAR2(255) PRIMARY KEY,  user\_id NVARCHAR2(255),  order\_date DATE,  status\_id NVARCHAR2(255)  );    ---- STATUS ----  CREATE TABLE Status (  status\_id NVARCHAR2(255) PRIMARY KEY,  start\_time DATE,  end\_time DATE  );    ---- ROLE ----  CREATE TABLE Role (  role\_id NVARCHAR2(255) PRIMARY KEY,  name NVARCHAR2(255) NOT NULL UNIQUE    );      ---- KEY, CONSTRAINT ----    ---- Book table ----  ALTER TABLE Book ADD CONSTRAINT fk\_publisher FOREIGN KEY (publisher\_id) REFERENCES Publisher(publisher\_id);    ---- Book\_Author table ----  ALTER TABLE Book\_Author ADD CONSTRAINT fk\_book FOREIGN KEY (book\_id) REFERENCES Book(book\_id);  ALTER TABLE Book\_Author ADD CONSTRAINT fk\_author FOREIGN KEY (author\_id) REFERENCES Author(author\_id);    ---- Categories\_Book table ----  ALTER TABLE Categories\_Book ADD CONSTRAINT fk\_cat\_book FOREIGN KEY (book\_id) REFERENCES Book(book\_id);  ALTER TABLE Categories\_Book ADD CONSTRAINT fk\_cat FOREIGN KEY (category\_id) REFERENCES Category(category\_id);    ---- User table ----  ALTER TABLE "User" ADD CONSTRAINT fk\_user\_role FOREIGN KEY (role\_id) REFERENCES Role(role\_id);    ---- Favorites table ----  ALTER TABLE Favorites ADD CONSTRAINT fk\_fav\_user FOREIGN KEY (user\_id) REFERENCES "User"(user\_id);  ALTER TABLE Favorites ADD CONSTRAINT fk\_fav\_book FOREIGN KEY (book\_id) REFERENCES Book(book\_id);    ---- Order table ----  ALTER TABLE "Order" ADD CONSTRAINT fk\_order\_user FOREIGN KEY (user\_id) REFERENCES "User"(user\_id);  ALTER TABLE "Order" ADD CONSTRAINT fk\_order\_status FOREIGN KEY (status\_id) REFERENCES Status(status\_id);  ALTER TABLE "Order" ADD CONSTRAINT fk\_order\_book FOREIGN KEY (book\_id) REFERENCES Book(book\_id);    ---- STATUS ----  ALTER TABLE Status  ADD CONSTRAINT chk\_status\_name CHECK (name IN ('Выдан', 'Завершен', 'В процессе', 'Отменен')); |

Приложение Б Листинг создания процедур

|  |
| --- |
| --------------------------------------------------------  -- DDL for Procedure ADD\_FAVORITE\_BOOK  --------------------------------------------------------    CREATE OR REPLACE PROCEDURE "ADD\_FAVORITE\_BOOK" (  p\_book\_id IN NVARCHAR2,  p\_user\_id IN NVARCHAR2  )  AS  BEGIN  INSERT INTO Favorites (user\_id, book\_id)  VALUES (p\_user\_id, p\_book\_id);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure AUTHENTICATE\_USER  --------------------------------------------------------  CREATE OR REPLACE PROCEDURE "AUTHENTICATE\_USER" (  p\_login IN NVARCHAR2,  p\_password IN NVARCHAR2,  p\_is\_authenticated OUT BOOLEAN  )  AS  hashed\_password NVARCHAR2(255);  BEGIN  SELECT password INTO hashed\_password  FROM "User"  WHERE login = p\_login;    p\_is\_authenticated := validate\_password(p\_password, hashed\_password);    EXCEPTION  WHEN NO\_DATA\_FOUND THEN  p\_is\_authenticated := FALSE;  WHEN OTHERS THEN  p\_is\_authenticated := FALSE;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CANCEL\_ORDER  --------------------------------------------------------    CREATE OR REPLACE PROCEDURE "CANCEL\_ORDER" (  p\_order\_id IN NVARCHAR2  )  AS  v\_book\_id NVARCHAR2(255);  v\_status\_id NVARCHAR2(255);  v\_status\_count NUMBER;  BEGIN    SELECT book\_id, status\_id INTO v\_book\_id, v\_status\_id  FROM "Order"  WHERE order\_id = p\_order\_id;      SELECT COUNT(\*)  INTO v\_status\_count  FROM Status  WHERE status\_id = v\_status\_id  AND name IN ('Выдан', 'В процессе');    IF v\_status\_count = 0 THEN  RAISE\_APPLICATION\_ERROR(-20007, 'Заказ нельзя отменить, так как он уже завершен или отменен.');  END IF;      UPDATE "Order"  SET status\_id = (SELECT status\_id FROM Status WHERE name = 'Отменен')  WHERE order\_id = p\_order\_id;      UPDATE Book  SET copies = copies + 1  WHERE book\_id = v\_book\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20006, 'Заказ не найден.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CHANGE\_ORDER\_STATUS  --------------------------------------------------------    CREATE OR REPLACE PROCEDURE "CHANGE\_ORDER\_STATUS" (  p\_order\_id IN NVARCHAR2,  p\_new\_status\_name IN NVARCHAR2  )  AS  v\_order\_count NUMBER;  v\_status\_id NVARCHAR2(255);  BEGIN    SELECT COUNT(\*)  INTO v\_order\_count  FROM "Order"  WHERE order\_id = p\_order\_id;    IF v\_order\_count = 0 THEN  RAISE\_APPLICATION\_ERROR(-20006, 'Заказ не найден.');  END IF;    SELECT status\_id INTO v\_status\_id  FROM "Order"  WHERE order\_id = p\_order\_id;      UPDATE Status  SET name = p\_new\_status\_name  WHERE status\_id = v\_status\_id;    COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CHANGE\_PASSWORD  --------------------------------------------------------    CREATE OR REPLACE PROCEDURE "CHANGE\_PASSWORD" (  p\_user\_id IN NVARCHAR2,  p\_old\_password IN NVARCHAR2,  p\_new\_password IN NVARCHAR2  )  AS  stored\_password RAW(64);  is\_authenticated BOOLEAN;  BEGIN  SELECT password INTO stored\_password  FROM "User"  WHERE user\_id = p\_user\_id;    is\_authenticated := validate\_password(p\_old\_password, stored\_password);    IF is\_authenticated THEN  stored\_password := hash\_password(p\_new\_password);  UPDATE "User"  SET password = stored\_password  WHERE user\_id = p\_user\_id;  COMMIT;  ELSE  RAISE\_APPLICATION\_ERROR(-20001, 'Неверный старый пароль.');  END IF;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Пользователь не найден.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CREATE\_AUTHOR  --------------------------------------------------------    CREATE OR REPLACE PROCEDURE "CREATE\_AUTHOR" (  p\_name IN NVARCHAR2  )  AS  p\_author\_id NVARCHAR2(255);  BEGIN  p\_author\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());  INSERT INTO Author (author\_id, name)  VALUES (p\_author\_id, p\_name);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CREATE\_BOOK  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "CREATE\_BOOK" (  p\_title IN NVARCHAR2,  p\_description IN NVARCHAR2 DEFAULT NULL,  p\_publication\_date IN NVARCHAR2 DEFAULT NULL,  p\_publisher\_name IN NVARCHAR2 DEFAULT NULL,  p\_isbn IN NVARCHAR2,  p\_tags IN NVARCHAR2 DEFAULT NULL,  p\_num\_pages IN INTEGER DEFAULT NULL,  p\_copies IN INTEGER DEFAULT NULL,  p\_file\_name IN NVARCHAR2 DEFAULT NULL,  p\_image\_name IN NVARCHAR2 DEFAULT NULL  )  AS  l\_publisher\_id NVARCHAR2(255) DEFAULT NULL;  l\_publication\_date DATE;  l\_book\_id NVARCHAR2(255);  i\_copies INTEGER;  i\_num\_pages INTEGER;  BEGIN    IF p\_publisher\_name IS NOT NULL THEN  SELECT publisher\_id INTO l\_publisher\_id  FROM Publisher WHERE name = p\_publisher\_name;  ELSE  l\_publisher\_id := NULL;  END IF;    IF p\_publication\_date IS NOT NULL THEN  l\_publication\_date := TO\_DATE(p\_publication\_date, 'DD-MON-YYYY');  ELSE  l\_publication\_date := NULL;  END IF;    l\_book\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());    INSERT INTO Book (book\_id, title, description, publication\_date, publisher\_id, isbn, tags, num\_pages, copies, file\_name, image\_name)  VALUES (l\_book\_id, p\_title, p\_description, p\_publication\_date, l\_publisher\_id, p\_isbn, p\_tags, p\_num\_pages, p\_copies, p\_file\_name, p\_image\_name);  COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Издатель с таким именем уже существует.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CREATE\_CATEGORY  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "CREATE\_CATEGORY" (  p\_name IN NVARCHAR2  )  AS  p\_category\_id NVARCHAR2(255);  BEGIN  p\_category\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());  INSERT INTO Category (category\_id, name)  VALUES (p\_category\_id, p\_name);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CREATE\_ORDER\_STATUS  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "CREATE\_ORDER\_STATUS" (  p\_user\_id IN NVARCHAR2,  p\_book\_id IN NVARCHAR2,  p\_days IN INTEGER DEFAULT 0,  p\_hours IN INTEGER DEFAULT 0, -- Добавление часов  p\_minutes IN INTEGER DEFAULT 15, -- Добавление минут  p\_status\_name NVARCHAR2  )  AS  v\_date DATE;  v\_status\_id NVARCHAR2(255);  v\_order\_id NVARCHAR2(255);  v\_copies INTEGER;  v\_final\_end\_time DATE;  BEGIN  v\_date := SYSDATE;    v\_final\_end\_time := SYSDATE + p\_days + p\_hours/24 + p\_minutes/(24\*60);    IF v\_final\_end\_time < v\_date THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Дата окончания статуса не может быть раньше текущей даты.');  END IF;    SELECT copies INTO v\_copies  FROM Book  WHERE book\_id = p\_book\_id;    IF v\_copies IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20005, 'Информация о книге не найдена.');  END IF;    IF v\_copies <= 0 THEN  RAISE\_APPLICATION\_ERROR(-20005, 'Книг в наличии нету');  END IF;    UPDATE Book  SET copies = copies - 1  WHERE book\_id = p\_book\_id;    v\_order\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());  v\_status\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());    INSERT INTO Status (status\_id, start\_time, end\_time, name)  VALUES (v\_status\_id, v\_date, v\_final\_end\_time, p\_status\_name); -- Используем v\_final\_end\_time    INSERT INTO "Order" (order\_id, user\_id, order\_date, status\_id, book\_id)  VALUES (v\_order\_id, p\_user\_id, v\_date, v\_status\_id, p\_book\_id);    COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CREATE\_PUBLISHER  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "CREATE\_PUBLISHER" (  p\_name IN NVARCHAR2,  p\_publisher\_id OUT NVARCHAR2  )  AS  BEGIN  p\_publisher\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());    INSERT INTO Publisher (publisher\_id, name)  VALUES (p\_publisher\_id, p\_name);    COMMIT;  EXCEPTION  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Издатель с таким именем уже существует.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure CREATE\_USER  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "CREATE\_USER" (  p\_login IN NVARCHAR2,  p\_password IN NVARCHAR2,  p\_name IN NVARCHAR2,  p\_second\_name IN NVARCHAR2 DEFAULT NULL,  p\_email IN NVARCHAR2 DEFAULT NULL,  p\_address IN NVARCHAR2 DEFAULT NULL,  p\_user\_id OUT NVARCHAR2,  p\_role\_id OUT NVARCHAR2  )  AS    hashed\_password NVARCHAR2(255);  BEGIN  hashed\_password := hash\_password(p\_password);    SELECT role\_id INTO p\_role\_id from ROLE WHERE name = 'user';    p\_user\_id := UTL\_ENCODE.base64\_encode(SYS\_GUID());    INSERT INTO "User" (user\_id, login, name, second\_name, email, address, password, role\_id)  VALUES (p\_user\_id, p\_login, p\_name, p\_second\_name, p\_email, p\_address, hashed\_password, p\_role\_id);    COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure DELETE\_AUTHOR\_SET\_NULL  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "DELETE\_AUTHOR\_SET\_NULL" (  p\_author\_id IN NVARCHAR2  )  AS  BEGIN    DELETE FROM BOOK\_AUTHOR WHERE author\_id = p\_author\_id;    DELETE FROM Author WHERE author\_id = p\_author\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Автор не найден.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure DELETE\_BOOK\_SET\_NULL  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "DELETE\_BOOK\_SET\_NULL" (  p\_book\_id IN NVARCHAR2  )  AS  BEGIN  UPDATE "Order" SET book\_id = NULL WHERE book\_id = p\_book\_id;  UPDATE Favorites SET book\_id = NULL WHERE book\_id = p\_book\_id;    DELETE FROM Book\_Author WHERE book\_id = p\_book\_id;    DELETE FROM Categories\_Book WHERE book\_id = p\_book\_id;    DELETE FROM Book WHERE book\_id = p\_book\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Книга не найден.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure DELETE\_CATEGORY\_SET\_NULL  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "DELETE\_CATEGORY\_SET\_NULL" (  p\_category\_id IN NVARCHAR2  )  AS  BEGIN    DELETE FROM Categories\_Book WHERE category\_id = p\_category\_id;    DELETE FROM Category WHERE category\_id = p\_category\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Категория не найден.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure DELETE\_FAVORITES  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "DELETE\_FAVORITES" (  p\_book\_id IN NVARCHAR2,  p\_user\_id IN NVARCHAR2  )  AS  BEGIN  DELETE FROM Favorites  WHERE book\_id = p\_book\_id AND user\_id = p\_user\_id;    DELETE FROM Favorites  WHERE book\_id IS NULL AND user\_id = p\_user\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Запись в избранном не найдена.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure DELETE\_PUBLISHER  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "DELETE\_PUBLISHER" (  p\_publisher\_id IN NVARCHAR2  )  AS  BEGIN    UPDATE Book SET publisher\_id = NULL WHERE publisher\_id = p\_publisher\_id;    DELETE FROM Publisher  WHERE publisher\_id = p\_publisher\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Издатель не найден.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure LINK\_BOOK\_AUTHOR  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "LINK\_BOOK\_AUTHOR" (  p\_book\_id IN NVARCHAR2,  p\_author\_id IN NVARCHAR2  )  AS  BEGIN  INSERT INTO Book\_Author (book\_id, author\_id)  VALUES (p\_book\_id, p\_author\_id);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure LINK\_BOOK\_CATEGORY  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "LINK\_BOOK\_CATEGORY" (  p\_book\_id IN NVARCHAR2,  p\_category\_id IN NVARCHAR2  )  AS  BEGIN  INSERT INTO Categories\_Book (book\_id, category\_id)  VALUES (p\_book\_id, p\_category\_id);  COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure UPDATE\_AUTHOR  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "UPDATE\_AUTHOR" (  p\_author\_id IN NVARCHAR2,  p\_name IN NVARCHAR2  )  AS  BEGIN  UPDATE Author  SET  name = COALESCE(p\_name, name)  WHERE author\_id = p\_author\_id;    COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure UPDATE\_BOOK  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "UPDATE\_BOOK" (  p\_book\_id IN NVARCHAR2,  p\_title IN NVARCHAR2 DEFAULT NULL,  p\_description IN NVARCHAR2 DEFAULT NULL,  p\_publication\_date IN DATE DEFAULT NULL,  p\_publisher\_id IN NVARCHAR2 DEFAULT NULL,  p\_isbn IN NVARCHAR2 DEFAULT NULL,  p\_tags IN NVARCHAR2 DEFAULT NULL,  p\_copies IN INTEGER DEFAULT NULL,  p\_num\_pages IN INTEGER DEFAULT NULL,  p\_file\_name IN NVARCHAR2 DEFAULT NULL,  p\_image\_name IN NVARCHAR2 DEFAULT NULL  )  AS  l\_count INTEGER;  BEGIN    IF p\_title IS NOT NULL THEN  SELECT COUNT(\*) INTO l\_count  FROM Book  WHERE title = p\_title  AND book\_id != p\_book\_id;    IF l\_count > 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Название книги уже занято.');  END IF;  END IF;      UPDATE Book  SET  title = COALESCE(p\_title, title),  description = COALESCE(p\_description, description),  publication\_date = COALESCE(p\_publication\_date, publication\_date),  publisher\_id = COALESCE(p\_publisher\_id, publisher\_id),  tags = COALESCE(p\_tags, tags),  copies = COALESCE(p\_copies, copies),  num\_pages = COALESCE(p\_num\_pages, num\_pages),  file\_name = COALESCE(p\_file\_name, file\_name),  image\_name = COALESCE(p\_image\_name, image\_name)  WHERE book\_id = p\_book\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Книга не найден.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure UPDATE\_CATEGORY  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "UPDATE\_CATEGORY" (  p\_category\_id IN NVARCHAR2,  p\_name IN NVARCHAR2  )  AS  BEGIN  UPDATE Category  SET  name = COALESCE(p\_name, name)  WHERE category\_id = p\_category\_id;    COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure UPDATE\_PUBLISHER  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "UPDATE\_PUBLISHER" (  p\_publisher\_id IN NVARCHAR2,  p\_name IN NVARCHAR2  )  AS  BEGIN  UPDATE Publisher  SET name = p\_name  WHERE publisher\_id = p\_publisher\_id;    COMMIT;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RAISE\_APPLICATION\_ERROR(-20002, 'Издатель не найден.');  WHEN DUP\_VAL\_ON\_INDEX THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Издатель с таким именем уже существует.');  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END;  --------------------------------------------------------  -- DDL for Procedure UPDATE\_USER  --------------------------------------------------------      CREATE OR REPLACE PROCEDURE "UPDATE\_USER" (  p\_user\_id IN RAW,  p\_name IN NVARCHAR2 DEFAULT NULL,  p\_second\_name IN NVARCHAR2 DEFAULT NULL,  p\_email IN NVARCHAR2 DEFAULT NULL,  p\_address IN NVARCHAR2 DEFAULT NULL  )  AS  BEGIN  UPDATE "User"  SET  name = COALESCE(p\_name, name),  second\_name = COALESCE(p\_second\_name, second\_name),  email = COALESCE(p\_email, email),  address = COALESCE(p\_address, address)  WHERE user\_id = p\_user\_id;    COMMIT;  EXCEPTION  WHEN OTHERS THEN  ROLLBACK;  RAISE;  END; |

Приложение В Листинг создания функций

|  |
| --- |
| --------------------------------------------------------  -- DDL for Function GET\_ALL\_AUTHORS  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_ALL\_AUTHORS"  RETURN author\_info\_table  IS  v\_authors author\_info\_table := author\_info\_table();  BEGIN  SELECT author\_info\_type(author\_id, name)  BULK COLLECT INTO v\_authors  FROM Author;  RETURN v\_authors;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN author\_info\_table();  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_ALL\_BOOKS  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_ALL\_BOOKS" (  p\_page\_number IN NUMBER DEFAULT 1,  p\_page\_size IN NUMBER DEFAULT 10  )  RETURN book\_result\_table  IS  v\_books book\_result\_table := book\_result\_table();  v\_start\_row NUMBER;  v\_end\_row NUMBER;  BEGIN  v\_start\_row := (p\_page\_number - 1) \* p\_page\_size + 1;  v\_end\_row := p\_page\_number \* p\_page\_size;    SELECT book\_result\_type(  b.book\_id,  b.title,  b.description,  b.publication\_date,  b.publisher\_id,  b.isbn,  b.tags,  b.copies,  b.num\_pages,  b.file\_name,  b.image\_name,  a.name,  c.name  )  BULK COLLECT INTO v\_books  FROM Book b  LEFT JOIN Book\_Author ba ON b.book\_id = ba.book\_id  LEFT JOIN Author a ON ba.author\_id = a.author\_id  LEFT JOIN Categories\_Book cb ON b.book\_id = cb.book\_id  LEFT JOIN Category c ON cb.category\_id = c.category\_id  ORDER BY b.title  OFFSET v\_start\_row - 1 ROWS FETCH NEXT p\_page\_size ROWS ONLY;    RETURN v\_books;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN book\_result\_table();  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_ALL\_CATEGORIES  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_ALL\_CATEGORIES"  RETURN category\_info\_table  IS  v\_categories category\_info\_table := category\_info\_table();  BEGIN  SELECT category\_info\_type(category\_id, name)  BULK COLLECT INTO v\_categories  FROM Category  FETCH FIRST 100 ROWS ONLY;    RETURN v\_categories;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN category\_info\_table();  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_ALL\_USERS\_INFO\_VIEW  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_ALL\_USERS\_INFO\_VIEW"  RETURN user\_info\_table  IS  v\_users user\_info\_table := user\_info\_table();  BEGIN  SELECT user\_info\_type(  user\_id,  login,  role\_id,  name,  second\_name,  email,  address  )  BULK COLLECT INTO v\_users  FROM "User";    RETURN v\_users;  END;  --------------------------------------------------------  -- DDL for Function GET\_AUTHOR\_BY\_ID  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_AUTHOR\_BY\_ID" (  p\_author\_id IN NVARCHAR2  )  RETURN author\_info\_type  IS  v\_author author\_info\_type;  BEGIN  SELECT author\_info\_type(author\_id, name)  INTO v\_author  FROM Author  WHERE author\_id = p\_author\_id;    RETURN v\_author;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN NULL;  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_BOOK\_BY\_ID  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_BOOK\_BY\_ID" (  p\_book\_id IN NVARCHAR2  )  RETURN book\_result\_table  IS  v\_books book\_result\_table := book\_result\_table();  BEGIN  SELECT book\_result\_type(  b.book\_id,  b.title,  b.description,  b.publication\_date,  b.publisher\_id,  b.isbn,  b.tags,  b.copies,  b.num\_pages,  b.file\_name,  b.image\_name,  a.name,  c.name  )  BULK COLLECT INTO v\_books  FROM Book b  LEFT JOIN Book\_Author ba ON b.book\_id = ba.book\_id  LEFT JOIN Author a ON ba.author\_id = a.author\_id  LEFT JOIN Categories\_Book cb ON b.book\_id = cb.book\_id  LEFT JOIN Category c ON cb.category\_id = c.category\_id  WHERE b.book\_id = p\_book\_id;    RETURN v\_books;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN book\_result\_table(); -- Возвращаем пустую таблицу  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_BOOKS\_BY\_AUTHOR  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_BOOKS\_BY\_AUTHOR" (  p\_author\_id IN NVARCHAR2,  p\_page\_number IN INTEGER DEFAULT 1,  p\_page\_size IN INTEGER DEFAULT 10  )  RETURN book\_result\_table  IS  v\_books book\_result\_table := book\_result\_table();  v\_start\_row INTEGER;  v\_end\_row INTEGER;  BEGIN  v\_start\_row := (p\_page\_number - 1) \* p\_page\_size + 1;  v\_end\_row := p\_page\_number \* p\_page\_size;    SELECT book\_result\_type(  b.book\_id,  b.title,  b.description,  b.publication\_date,  b.publisher\_id,  b.isbn,  b.tags,  b.copies,  b.num\_pages,  b.file\_name,  b.image\_name,  a.name,  c.name  )  BULK COLLECT INTO v\_books  FROM Book b  JOIN Book\_Author ba ON b.book\_id = ba.book\_id  LEFT JOIN Author a ON ba.author\_id = a.author\_id  LEFT JOIN Categories\_Book cb ON b.book\_id = cb.book\_id  LEFT JOIN Category c ON cb.category\_id = c.category\_id  WHERE ba.author\_id = p\_author\_id  ORDER BY b.title  OFFSET v\_start\_row - 1 ROWS FETCH NEXT p\_page\_size ROWS ONLY;    RETURN v\_books;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN book\_result\_table(); -- Возвращаем пустую таблицу  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_BOOKS\_BY\_CATEGORY  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_BOOKS\_BY\_CATEGORY" (  p\_category\_id IN NVARCHAR2,  p\_page\_number IN INTEGER DEFAULT 1,  p\_page\_size IN INTEGER DEFAULT 10  )  RETURN book\_result\_table  IS  v\_books book\_result\_table := book\_result\_table();  v\_start\_row INTEGER;  v\_end\_row INTEGER;  BEGIN  v\_start\_row := (p\_page\_number - 1) \* p\_page\_size + 1;  v\_end\_row := p\_page\_number \* p\_page\_size;    SELECT book\_result\_type(  b.book\_id,  b.title,  b.description,  b.publication\_date,  b.publisher\_id,  b.isbn,  b.tags,  b.copies,  b.num\_pages,  b.file\_name,  b.image\_name,  a.name,  c.name  )  BULK COLLECT INTO v\_books  FROM Book b  JOIN Categories\_Book cb ON b.book\_id = cb.book\_id  LEFT JOIN Book\_Author ba ON b.book\_id = ba.book\_id  LEFT JOIN Author a ON ba.author\_id = a.author\_id  LEFT JOIN Category c ON cb.category\_id = c.category\_id  WHERE cb.category\_id = p\_category\_id  ORDER BY b.title  OFFSET v\_start\_row - 1 ROWS FETCH NEXT p\_page\_size ROWS ONLY;    RETURN v\_books;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN book\_result\_table();  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_BOOKS\_BY\_PUBLISHER  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_BOOKS\_BY\_PUBLISHER" (  p\_publisher\_id IN NVARCHAR2,  p\_page\_number IN INTEGER DEFAULT 1,  p\_page\_size IN INTEGER DEFAULT 10  )  RETURN book\_result\_table  IS  v\_books book\_result\_table := book\_result\_table();  v\_start\_row INTEGER;  v\_end\_row INTEGER;  BEGIN  v\_start\_row := (p\_page\_number - 1) \* p\_page\_size + 1;  v\_end\_row := p\_page\_number \* p\_page\_size;    SELECT book\_result\_type(  b.book\_id,  b.title,  b.description,  b.publication\_date,  b.publisher\_id,  b.isbn,  b.tags,  b.copies,  b.num\_pages,  b.file\_name,  b.image\_name,  a.name,  c.name  )  BULK COLLECT INTO v\_books  FROM Book b  LEFT JOIN Book\_Author ba ON b.book\_id = ba.book\_id  LEFT JOIN Author a ON ba.author\_id = a.author\_id  LEFT JOIN Categories\_Book cb ON b.book\_id = cb.book\_id  LEFT JOIN Category c ON cb.category\_id = c.category\_id  WHERE b.publisher\_id = p\_publisher\_id  ORDER BY b.title  OFFSET v\_start\_row - 1 ROWS FETCH NEXT p\_page\_size ROWS ONLY;    RETURN v\_books;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN book\_result\_table();  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_CATEGORY\_BY\_ID  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_CATEGORY\_BY\_ID" (  p\_category\_id IN NVARCHAR2  )  RETURN category\_info\_type  IS  v\_category category\_info\_type;  BEGIN  SELECT category\_info\_type(category\_id, name)  INTO v\_category  FROM Category  WHERE category\_id = p\_category\_id;    RETURN v\_category;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN NULL;  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_FAVORITE\_BOOKS\_BY\_USER  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_FAVORITE\_BOOKS\_BY\_USER" (  p\_user\_id IN NVARCHAR2  )  RETURN book\_result\_table  IS  v\_books book\_result\_table := book\_result\_table();  BEGIN  SELECT book\_result\_type(  b.book\_id,  b.title,  b.description,  b.publication\_date,  b.publisher\_id,  b.isbn,  b.tags,  b.copies,  b.num\_pages,  b.file\_name,  b.image\_name,  a.name,  c.name  )  BULK COLLECT INTO v\_books  FROM Book b  JOIN Favorites f ON b.book\_id = f.book\_id  LEFT JOIN Book\_Author ba ON b.book\_id = ba.book\_id  LEFT JOIN Author a ON ba.author\_id = a.author\_id  LEFT JOIN Categories\_Book cb ON b.book\_id = cb.book\_id  LEFT JOIN Category c ON cb.category\_id = c.category\_id  WHERE f.user\_id = p\_user\_id;    RETURN v\_books;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN book\_result\_table();  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_ORDER\_BY\_ID  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_ORDER\_BY\_ID" (  p\_order\_id IN NVARCHAR2  )  RETURN order\_with\_status\_type  IS  v\_order order\_with\_status\_type;  BEGIN  SELECT order\_with\_status\_type(  o.order\_id,  o.user\_id,  o.book\_id,  o.order\_date,  s.name,  s.start\_time,  s.end\_time  )  INTO v\_order  FROM "Order" o  JOIN Status s ON o.status\_id = s.status\_id  WHERE o.order\_id = p\_order\_id;    RETURN v\_order;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN NULL;  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_ORDERS\_BY\_USER\_OR\_BOOK  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_ORDERS\_BY\_USER\_OR\_BOOK" (  p\_user\_id IN NVARCHAR2 DEFAULT NULL,  p\_book\_id IN NVARCHAR2 DEFAULT NULL  )  RETURN order\_with\_status\_table  IS  v\_orders order\_with\_status\_table := order\_with\_status\_table();  BEGIN  SELECT order\_with\_status\_type(  o.order\_id,  o.user\_id,  o.book\_id,  o.order\_date,  s.name,  s.start\_time,  s.end\_time  )  BULK COLLECT INTO v\_orders  FROM "Order" o  JOIN Status s ON o.status\_id = s.status\_id  WHERE (p\_user\_id IS NULL OR o.user\_id = p\_user\_id)  AND (p\_book\_id IS NULL OR o.book\_id = p\_book\_id);    RETURN v\_orders;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN order\_with\_status\_table(); -- Возвращаем пустую таблицу  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_ORDERS\_WITH\_STATUS  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_ORDERS\_WITH\_STATUS"  RETURN order\_with\_status\_table  IS  v\_orders order\_with\_status\_table := order\_with\_status\_table();  BEGIN  SELECT order\_with\_status\_type(  o.order\_id,  o.user\_id,  o.book\_id,  o.order\_date,  s.name,  s.start\_time,  s.end\_time  )  BULK COLLECT INTO v\_orders  FROM "Order" o  JOIN Status s ON o.status\_id = s.status\_id;    RETURN v\_orders;  END;  --------------------------------------------------------  -- DDL for Function GET\_USER\_BY\_ID  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_USER\_BY\_ID" (  p\_user\_id IN NVARCHAR2  )  RETURN user\_info\_type  IS  v\_user user\_info\_type;  BEGIN  SELECT user\_info\_type(  user\_id,  login,  role\_id,  name,  second\_name,  email,  address  )  INTO v\_user  FROM "User"  WHERE user\_id = p\_user\_id;    RETURN v\_user;  EXCEPTION  WHEN NO\_DATA\_FOUND THEN  RETURN NULL;  WHEN OTHERS THEN  RAISE;  END;  --------------------------------------------------------  -- DDL for Function GET\_USERS\_INFO  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "GET\_USERS\_INFO"  RETURN user\_info\_table  IS  v\_users user\_info\_table := user\_info\_table();  BEGIN  SELECT user\_info\_type(  user\_id,  login,  role\_id,  name,  second\_name,  email,  address  )  BULK COLLECT INTO v\_users  FROM "User";    RETURN v\_users;  END;  --------------------------------------------------------  -- DDL for Function HASH\_PASSWORD  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "HASH\_PASSWORD" (  p\_password IN NVARCHAR2  ) RETURN NVARCHAR2  AS  p\_hash\_raw RAW(32);  p\_hash\_hex NVARCHAR2(255);  BEGIN  p\_hash\_raw := DBMS\_CRYPTO.HASH(UTL\_I18N.STRING\_TO\_RAW(p\_password, 'AL32UTF8'), DBMS\_CRYPTO.HASH\_SH256);  p\_hash\_hex := UTL\_ENCODE.base64\_encode(p\_hash\_raw);  RETURN p\_hash\_hex;  END;  --------------------------------------------------------  -- DDL for Function SEARCH\_BOOKS  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "SEARCH\_BOOKS" (  p\_search\_term IN NVARCHAR2  )  RETURN book\_result\_table  IS  v\_results book\_result\_table := book\_result\_table();  BEGIN  SELECT book\_result\_type(  b.book\_id,  b.title,  b.description,  b.publication\_date,  b.publisher\_id,  b.isbn,  b.tags,  b.copies,  b.num\_pages,  b.file\_name,  b.image\_name,  a.name,  c.name  )  BULK COLLECT INTO v\_results  FROM Book b  LEFT JOIN Book\_Author ba ON b.book\_id = ba.book\_id  LEFT JOIN Author a ON ba.author\_id = a.author\_id  LEFT JOIN Categories\_Book cb ON b.book\_id = cb.book\_id  LEFT JOIN Category c ON cb.category\_id = c.category\_id  WHERE UPPER(b.title) LIKE '%' || UPPER(p\_search\_term) || '%' OR UPPER(a.name) LIKE '%' || UPPER(p\_search\_term) || '%'  AND ROWNUM <= 100;    RETURN v\_results;  END;  --------------------------------------------------------  -- DDL for Function VALIDATE\_PASSWORD  --------------------------------------------------------    CREATE OR REPLACE FUNCTION "VALIDATE\_PASSWORD" (  p\_password IN NVARCHAR2,  p\_hashed\_password IN NVARCHAR2  ) RETURN BOOLEAN  AS  l\_hash RAW(32);  l\_hash\_hex NVARCHAR2(255);  BEGIN  l\_hash := DBMS\_CRYPTO.HASH(  UTL\_I18N.STRING\_TO\_RAW(p\_password, 'AL32UTF8'),  DBMS\_CRYPTO.HASH\_SH256  );  l\_hash\_hex := UTL\_ENCODE.base64\_encode(l\_hash);    RETURN l\_hash\_hex = p\_hashed\_password;  END; |

Приложение В Листинг создания триггеров

|  |
| --- |
| CREATE OR REPLACE TRIGGER check\_book\_copies  BEFORE UPDATE OR INSERT ON Book  FOR EACH ROW  BEGIN  IF :new.copies < 0 THEN  RAISE\_APPLICATION\_ERROR(-20003, 'Количество копий книги не может быть меньше или равно нулю.');  END IF;  END;      CREATE OR REPLACE TRIGGER complete\_orders\_on\_expiration  AFTER UPDATE OF end\_time ON Status  FOR EACH ROW  DECLARE  v\_order\_id "Order".order\_id%TYPE;  BEGIN  -- Получаем ID заказа по ID статуса  SELECT order\_id INTO v\_order\_id  FROM "Order"  WHERE status\_id = :old.status\_id;    -- Проверяем, истекло ли время статуса  IF :old.end\_time < SYSDATE THEN  -- Обновляем статус заказа на "Завершен"  UPDATE "Order"  SET status\_id = (SELECT status\_id FROM Status WHERE name = 'Завершен')  WHERE order\_id = v\_order\_id;    -- Увеличиваем количество копий книги на 1  UPDATE Book  SET copies = copies + 1  WHERE book\_id = (SELECT book\_id FROM "Order" WHERE order\_id = v\_order\_id);  END IF;  END;    CREATE OR REPLACE TRIGGER check\_num\_pages  BEFORE INSERT OR UPDATE ON Book  FOR EACH ROW  BEGIN  IF :NEW.num\_pages < 0 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Количество страниц не может быть меньше 0');  END IF;  END;    CREATE OR REPLACE TRIGGER check\_title\_length  BEFORE INSERT OR UPDATE ON Book  FOR EACH ROW  BEGIN  IF LENGTH(:NEW.title) < 3 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Название должно быть больше 3ех символов');  END IF;  END;    CREATE OR REPLACE TRIGGER check\_end\_time  BEFORE INSERT OR UPDATE ON Status  FOR EACH ROW  BEGIN  IF :NEW.end\_time < :NEW.start\_time THEN  RAISE\_APPLICATION\_ERROR(-20001, 'End time не можеть быть раньше start time');  END IF;  END;    CREATE OR REPLACE TRIGGER check\_login\_length  BEFORE INSERT OR UPDATE ON "User"  FOR EACH ROW  BEGIN  IF LENGTH(:NEW.login) < 4 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'Логин не можеть быть меньше 4ех символов');  END IF;  END; |