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 для сервиса для публикации и чтения электронных книг с применение Full Text Search- технологии»

Выполнил студент Дребезов И.В.

(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

Минск 2024

Содержание

[Введение 5](#_Toc166816627)

[1 Постановка задачи 6](#_Toc166816628)

[1.1 Аналитический обзор аналогов 6](#_Toc166816629)

[1.1.1 Аналог FicBook 6](#_Toc166816630)

[1.1.2 Аналог Author.today 7](#_Toc166816631)

[1.2 Вывод 8](#_Toc166816632)

[2 Проектирование базы данных 9](#_Toc166816633)

[2.1 Определение вариантов использования 9](#_Toc166816634)

[2.2 Диаграммы UML, взаимодействие всех компонентов 10](#_Toc166816635)

[2.3 Вывод 11](#_Toc166816636)

[3 Разработка объектов базы данных 12](#_Toc166816637)

[3.1 Разработка таблиц базы данных 12](#_Toc166816638)

[3.2 Разработка представлений базы данных 15](#_Toc166816639)

[3.3 Разработка индексов базы данных 16](#_Toc166816640)

[3.4 Разработка процедур базы данных 16](#_Toc166816641)

[3.5 Разработка функций базы данных 18](#_Toc166816642)

[3.6 Разработка триггеров базы данных 19](#_Toc166816643)

[3.7 Создание пользователей 20](#_Toc166816644)

[3.8 Вывод 21](#_Toc166816645)

[4 Описание процедур импорта и экспорта 22](#_Toc166816646)

[5 Тестирование производительности 23](#_Toc166816647)

[6 Описание технологии и ее применение в базе данных 25](#_Toc166816648)

[7 Руководство пользователя 26](#_Toc166816649)

[Заключение 27](#_Toc166816650)

[Список используемых источников 28](#_Toc166816651)

[Приложение А. Листинг создания таблиц 29](#_Toc166816652)

[Приложение Б. Листинг процедур базы данных 31](#_Toc166816653)

[Приложение В. Листинг функций базы данных 38](#_Toc166816654)

[Приложение Г. Листинг представлений базы данных 39](#_Toc166816655)

[Приложение Д. Листинг кода создания пользователей 41](#_Toc166816656)

[Приложение Е. Листинг процедур для импорта и экспорта в xml 43](#_Toc166816657)

[Приложение Ж. Листинг Триггеров 44](#_Toc166816658)

# Введение

Цель данной работы заключается в создании реляционной базы данных для онлайн библиотеки, которая обеспечивает пользователя доступом к имеющимся на ней книгам и позволяет добавлять свои. Создание такой базы данных позволит организовать удобное хранение, и быстрый доступ к хранящейся информации.

База данных - это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Oracle, поскольку эта система обладает высокой надежностью и производительностью, а также позволяет работать с большим объёмом данных.

База данных должна быть защищена от несанкционированного доступа. Для этого данные изолированы от прямого вмешательства пользователей с помощью процедур, которые в свою очередь распределены по пользователям в зависимости от необходимого для их роли набора привилегий.

Для повышения точности и эффективности поиска была реализована технология Full Text Search

Для повышения скорости чтения данных из базы данных были созданы индексы для самых часто используемых наборов столбцов.

В пояснительной записке содержится информация о структуре и реализации проекта.

# 1 Постановка задачи

## Аналитический обзор аналогов

Электронные книги – удобный формат документов, который позволяет хранить большой объем книг в цифровом формате.

В настоящие дни электронные книги активно набирают популярно. С каждым днём все больше людей предпочитаю цифровую версию книг в связи с их удобством. Вместе с этим сам процесс публикации и распространения стал доступнее и проще, позволяя любому желающему попробовать себя в роле писателя.

На фоне подобной тенденции появляется всё больше новых сервисов, предоставляющих возможность для свободной публикации и чтения электронных книг.

## 1.1.1 Аналог FicBook

FicBook – популярный сервис для свободной публикации и чтения книг. Изначально сайт был создан для публикации фанфиков, но быстро набрал популярность и теперь совмещает самые разные жанры и форматы.

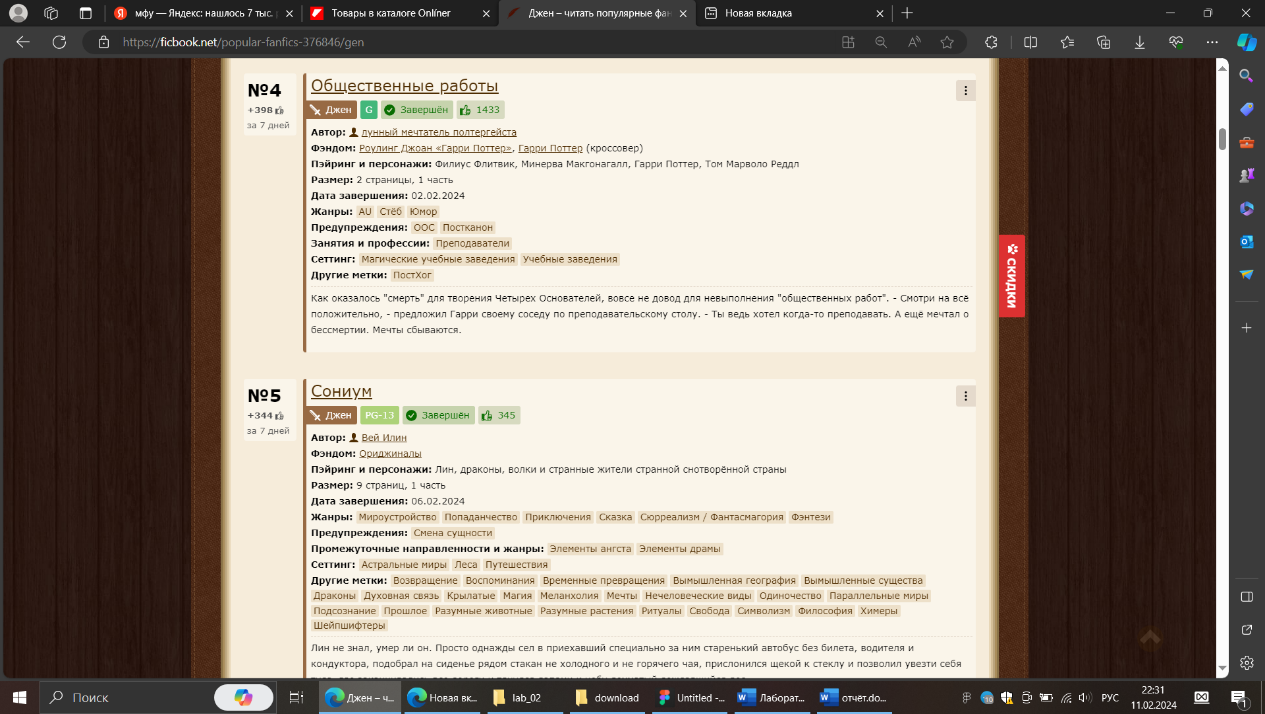


Рисунок 1.1 – Каталог книг на FicBook

Главная особенность сервиса – гибка и широкая система жанров и специальных тэгов. FicBook предоставляет большое разнообразие жанров и тэгов, которые регулярно пополняются. Это позволяет эффективнее находить книгу по вкусу, а писателям не нужно беспокоится об описании, так как тэги помогают передать содержание.

Так же сервис предоставляет удобную систему сборников. Пользователь может в любой момент создать сборник и добавлять в него любые книги. Это позволяет удобно хранить и сортировать понравившиеся книги

Ещё на FicBook есть возможность писать комментарий отдельно к каждой главе книги, что позволяет удобно делится мнение. А в профиле пользователя всегда можно посмотреть все его оставленные отзывы.

## 1.1.2 Аналог Author.today

Author.today – популярный сервис для публикации и чтения книг. Сервис ориентирован на оригинальные работы, но пользователи имеют возможность публиковать любое своё творчество.

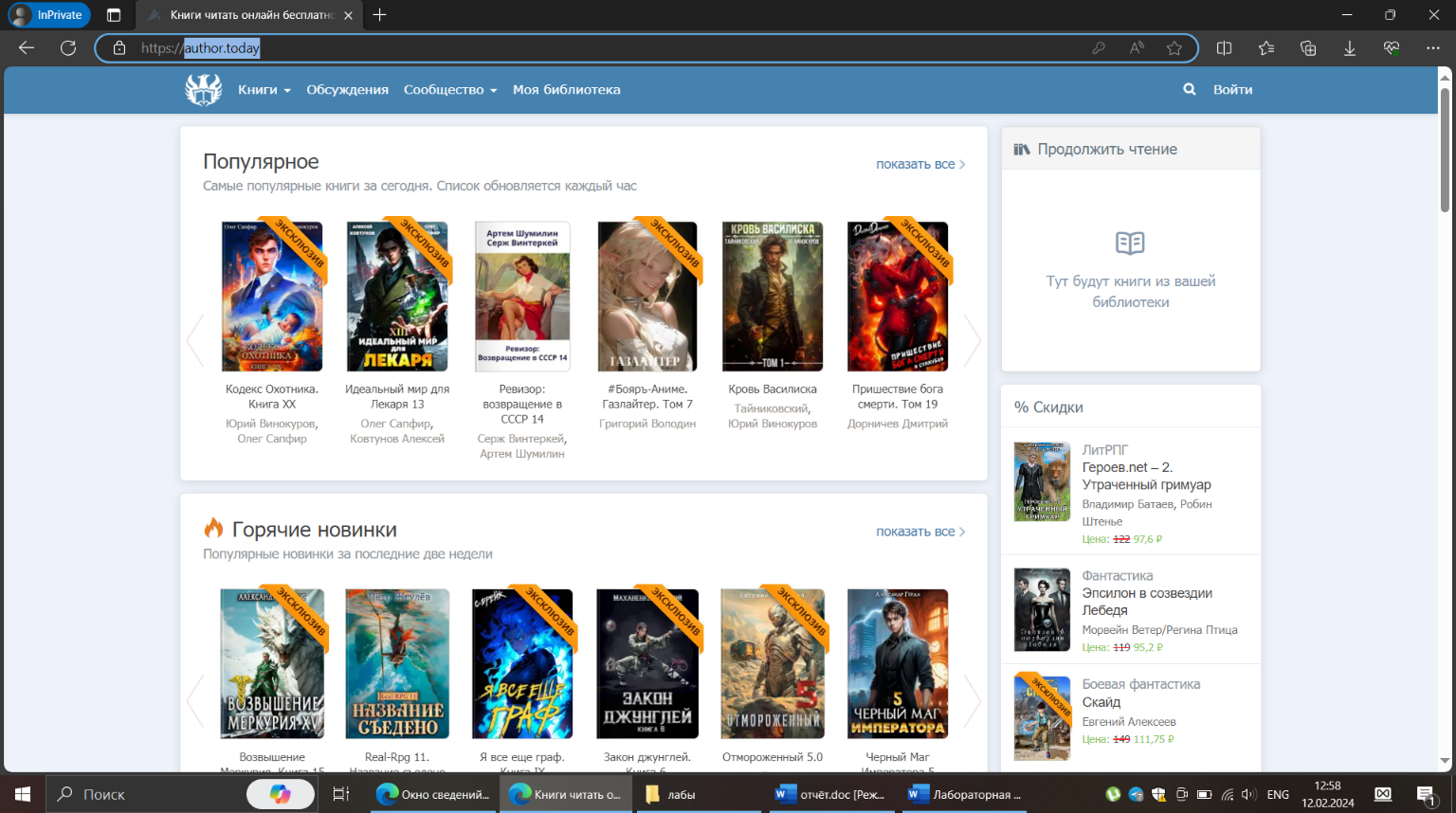


Рисунок 1.2 – Главная страница Author.today

При публикации книг пользователи имеют возможность создавать свои собственные жанры, что приводит к образованию большого количества однотипных тэгов. Из-за этого многие работы просто теряются, а читателям сложно находить новые книги.

Пользователи могут писать комментарии под всей книгой. Но из-за этого сложнее ссылаться на конкретную главу из-за чего под книгой скапливается куча неотсортированных комментариев.

Также сервис предоставляет заготовленные сборники для разных случаев, для сохранения книг. Есть и пользовательские сборники, но имеют другой вид и минимум три книги для их создания, что делает их неудобными для обычного использования.

## 1.2 Вывод

Основной задачей проекта является построение надежной базы данных, которая будет хранить информацию о пользователях, их книгах, коллекция, отзывах и доступных жанрах.

В рамках разработки планируются следующие роли: администратор, читатель, писатель.

При разработке базы данных должны быть выполнены следующие требования:

˗ база данных должна быть реализована в СУБД Oracle.

˗ доступ к данным должен осуществляться только через соответствующие процедуры, функции и представления;

˗ должен быть проведен импорт данных из формата XML, экспорт данных в формат XML;

˗ протестировать производительность базы данных на таблице, содержащей не менее 100 000 строк, и внести изменения в структуру в случае необходимости.

˗ проанализировать планы запросов к таблице;

˗ применить технологию. Full Text Search

# 2 Проектирование базы данных

## 2.1 Определение вариантов использования

Варианты 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 должны быть доступны для каждой роли, а также как должна быть организована навигация в системе. Варианты использования обычно представляются в виде UML диаграмм, которые позволяют наглядно отобразить взаимодействие между пользователями и системой.

Диаграмма показана на рисунке 2.1.

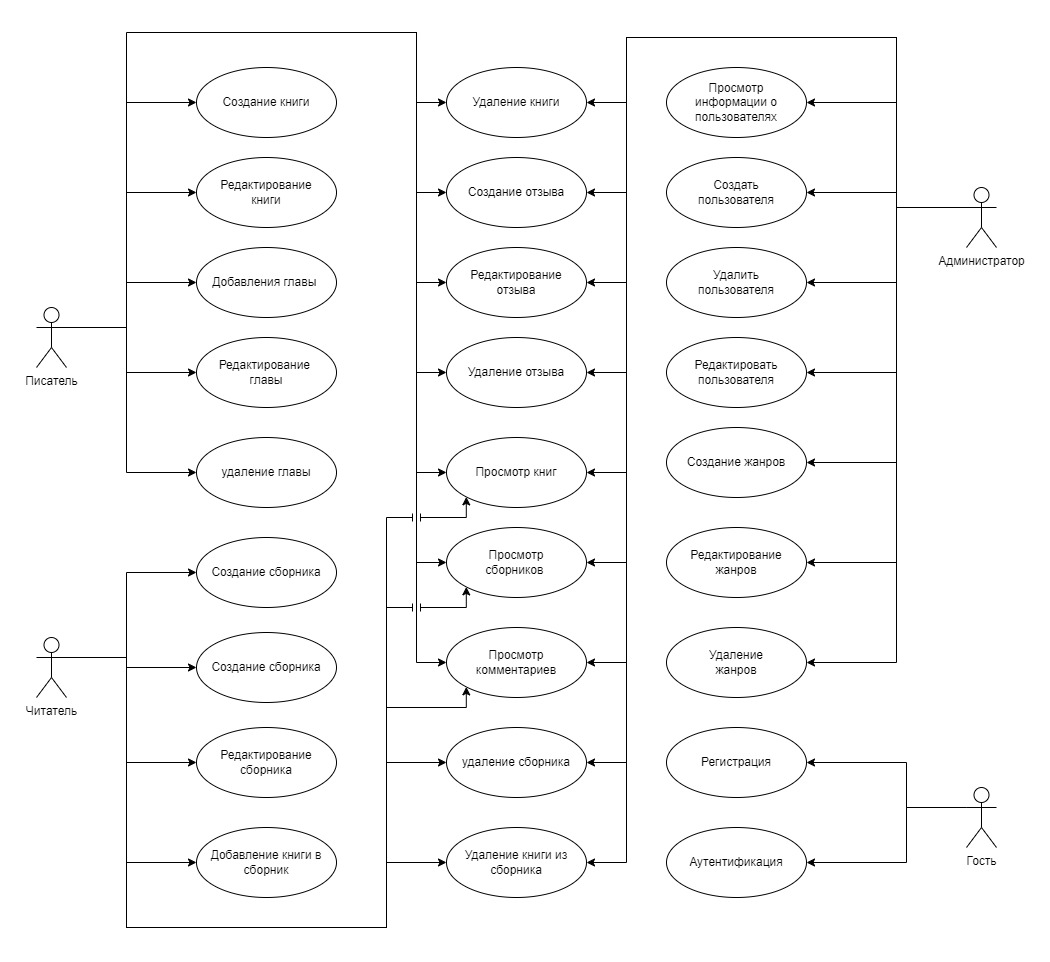


Рисунок 2.1 – UML диаграмма вариантов использования

Роли пользователя — это набор прав, которые пользователь может получить в системе. Роли обычно назначаются пользователям в зависимости от их позиции, роли или ответственности в организации. В зависимости от роли пользователя, он может иметь доступ к различным функциям системы. В данном проекте роли пользователей будут следующими:

* Гость;
* Администратор;
* Читатель;
* Писатель.

В начале работы с приложением пользователь является гостем. После регистрации гость становится либо Писателем, либо Читателем в зависимом от выбранного типа. В роли Читателя пользователь получает возможность создавать свои коллекции для хранения книг. В роли Писателя пользовать получает возможность создавать и управлять книгами.

Просмотр доступных книг, жанров и коллекций доступен всем всем авторизованным пользователям.

## 2.2 Диаграммы UML, взаимодействие 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 представлена на рисунке 2.2.

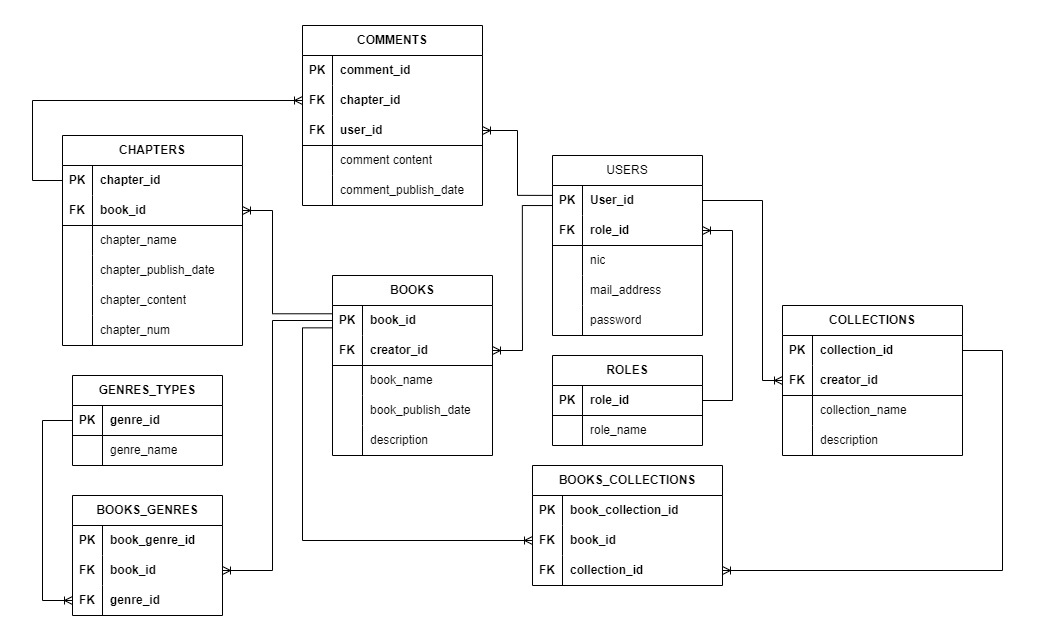


Рисунок 2.2 – Диаграмма базы данных

Видно, что диаграмма 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 BOOKS связана с таблицей USERS, через внешний ключ creator\_id. Также видно, что таблицы USERS и ROLES связаны друг с другом через внешний и первичный ключ role\_id. Все связи представлены на рисунке 2.2, который располагается выше.

Для обеспечения безопасности данных необходимо реализовать обращение к данным через процедуры.

## 2.3 Вывод

Для реализации базы данных было разработано 9 таблиц.

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

При разработке приложения для курсового проекта была использована база данных Oracle.

Для реализации базы данных было разработано 9 таблиц. В структуру схемы базы данных для проекта входят следующие таблицы: ROLES, USERS, COLLECTIONS, BOOKS, CHAPTERS, COMMENTS, GENRES\_TYPES, BOOKS\_GENRES, BOOKS\_COLLECTIONS. Ниже будет описание про каждую из них более подробно.

Таблица USERS содержит информацию о пользователях. Состоит из следующих столбцов (таблица 3.1):

Таблица 3.1 – Столбцы таблицы USER

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| user\_id | Уникальный идентификатор пользователя, первичный ключ | NUMBER(10) |
| user\_name | Имя пользователя | VARCHAR2(100) |
| user\_email | Электронная почта пользователя | VARCHAR2(100) |
| user\_password | Пароль пользователя | VARCHAR2(100) |
| user\_role | Уникальный идентификатор роли, внешний ключ | NUMBER(10) |

Таблица BOOKS содержит информацию о книгах. Состоит из следующих столбцов (таблица 3.2):

Таблица 3.2 – Столбцы таблицы BOOKS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| book\_id | Уникальный идентификатор книги, первичный ключ | NUMBER(10) |
| creator\_id | Уникальный идентификатор пользователя, внешний ключ | NUMBER(10) |
| book\_name | Название книги | VARCHAR2(100) |
| book\_publish\_date | Дата публикации книги | DATE |
| description | Описание книги | VARCHAR2(200) |

Таблица CHAPTERS содержит информацию о главах в книгах. Состоит из следующих столбцов (таблица 3.3):

Таблица 3.3 – Столбцы таблицы CHAPTERS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| chapter\_id | Уникальный идентификатор главы, первичный ключ | NUMBER(10) |
| book\_id | Уникальный идентификатор книги, внешний ключ | NUMBER(10) |
| chapter\_name | Название главы | VARCHAR2(100) |
| chapter\_publish\_date | Дата публикации главы | DATE |
| chapter\_content | Путь к содержимому главы | VARCHAR2(100) |
| chapter\_num | Порядковый номер главы в книге | NUMBER(10) |

Таблица GENRES\_TYPES содержит информацию о жанрах. Состоит из следующих столбцов (таблица 3.4):

Таблица 3.4 – Столбцы таблицы GENRES\_TYPES

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| genre\_id | Уникальный идентификатор жанра, первичный ключ | NUMBER(10) |
| genre\_name | Название женра | VARCHAR2(50) |

Таблица BOOKS\_GENRES содержит информацию о том какие жанры есть в книге. Состоит из следующих столбцов (таблица 3.5):

Таблица 3.5 – Столбцы таблицы BOOKS\_GENRES

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| book\_genre\_id | Уникальный идентификатор жанра в книге, первичный ключ | NUMBER(10) |
| genre\_id | Уникальный идентификатор жанра, внешний ключ | NUMBER(10) |
| book\_id | Уникальный идентификатор книги, внешний ключ | NUMBER(10) |

Таблица COLLECTIONS содержит информацию о сборниках. Состоит из следующих столбцов (таблица 3.6):

Таблица 3.6 – Столбцы таблицы COLLECTIONS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| collection\_id | Уникальный идентификатор сборника, первичный ключ | NUMBER(10) |
| creator\_id | Уникальный идентификатор пользователя, внешний ключ | NUMBER(10) |
| collection\_name | Название сборника | VARCHAR2(100) |
| description | Описание сборника | VARCHAR2(200) |

Таблица BOOKS\_COLLECTIONS содержит информацию о книгах в сборниках. Состоит из следующих столбцов (таблица 3.7):

Таблица 3.7 – Столбцы таблицы BOOKS\_COLLECTIONS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| book\_collection\_id | Уникальный идентификатор книги в сборнике, внешний ключ | NUMBER(10) |
| collection\_\_id | Уникальный идентификатор сборника, внешний ключ | NUMBER(10) |
| book\_id | Уникальный идентификатор книги, внешний ключ | NUMBER(10) |

Таблица ROLES содержит информацию о ролях. Состоит из следующих столбцов (таблица 3.8):

Таблица 3.8 – Столбцы таблицы ROLES

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| role\_id | Уникальный идентификатор роли, внешний ключ | NUMBER(10) |
| role\_name | Наименование роли | VARCHAR2(50) |

Таблица COMMENTS содержит информацию о комментариях. Состоит из следующих столбцов (таблица 3.9):

Таблица 3.9 – Столбцы таблицы COMMENTS

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| comment\_id | Уникальный идентификатор комментария, первичный ключ | NUMBER(10) |
| chapter\_id | Уникальный идентификатор главы, внешний ключ | NUMBER(10) |
| user\_id | Уникальный идентификатор автора комментария, внешний ключ | NUMBER(10) |
| comment\_content | Содержание комментария | VARCHAR2(200) |
| comment\_publish\_date | Дата публикации комментария | DATE |

Перед созданием таблиц, для их структуризации и изоляции было принято решение использовать табличные пространства.

Табличное пространство – это механизм, который помогает связать объекты базы данных, такие как таблицы, индексы и представления, с файловой системой. Оно позволяет логически разделять объекты базы данных на разные физические устройства или диски, что может улучшить производительность работы с базой данных. Скрипт для создания табличных пространств представлен в листинге 3.1.

|  |
| --- |
| CREATE TABLESPACE TS\_PRIME  DATAFILE 'C:\oraDB\oradata\ORCLSEC\TS\_PRIME.dbf'  SIZE 100M  AUTOEXTEND ON; |

Листинг 3.1 – Cкрипт для создания табличных пространств

Листинг SQL-кода для создания таблиц находится в [приложении А](#_Приложение_А).

Для эффективного использования базы данных в проекте, необходимо создать индексы на столбцах, используемых в запросах с поиском данных. Для таблицы parts был создан индекс на основании столбцов type, manufacturer и model, так как они редко изменяются и часто используются для поиска.

Для более изоляции пользователя от таблиц и облегчения некоторого функционала создано несколько процедур и представлений.

## 3.2 Разработка представлений базы данных

Представление (view) в базе данных представляет собой виртуальную таблицу, которая создается на основе запроса к одной или нескольким таблицам в базе данных. Представления позволяют обращаться к данным из нескольких таблиц одновременно, при этом не изменяя структуру этих таблиц.

В данном проекте были созданы 14 представлений:

ALL\_ROLES - показывает весь список ролей;

ALL\_USER - показывает информацию о всех пользователей, кроме паролей;

ALL\_COLLECTIONS - показывает информацию о сборниках;

ALL\_BOOKS - показывает информацию о книгах;

ALL\_CHAPTERS - показывает информацию о всех главах;

ALL\_COMMENTS - показывает информацию о всех комментария;

ALL\_GENRES\_TYPES - показывает информацию о всех жанрах;

ALL\_BOOKS\_GENRES - показывает информацию о всех жанрах в книгах;

ALL\_BOOKS\_COLLECTIONS - показывает информацию о всех книгах в коллекциях;

ALL\_USER\_ROLES - показывает пользователей вместе с их ролями;

ALL\_USER\_BOOKS - показывает пользователей вместе с их книгами;

ALL\_USER\_COLLECTIONS - показывает пользователей вместе с их коллекциями;

ALL\_BOOKS\_IN\_COLLECTIONS - показывает все книги в коллекциях;

ALL\_CHAPTERS\_IN\_BOOK – показывает все главы в книгах;

Реализация нескольких из представлений представлено в листинге 3.2. Остальные представления будут аналогичны, работая с другими данными. Их код находится в приложении в листинге Г

|  |
| --- |
| CREATE OR REPLACE VIEW ALL\_USER\_COLLECTIONS AS  SELECT USER\_ID, USER\_NAME, COLLECTION\_ID, COLLECTION\_NAME  FROM USERS  JOIN COLLECTIONS  ON USERS.USER\_ID = COLLECTIONS.CREATOR\_ID  ORDER BY USER\_NAME;  CREATE OR REPLACE VIEW ALL\_BOOKS\_IN\_COLLECTIONS AS  SELECT COLLECTIONS.COLLECTION\_ID, COLLECTION\_NAME, BOOKS.BOOK\_ID, BOOK\_NAME  FROM BOOKS  JOIN BOOKS\_COLLECTIONS  ON BOOKS.BOOK\_ID = BOOKS\_COLLECTIONS.BOOK\_ID  JOIN COLLECTIONS  ON COLLECTIONS.COLLECTION\_ID = BOOKS\_COLLECTIONS.COLLECTION\_ID  ORDER BY COLLECTION\_NAME; |

Листинг 3.2 – Представления ALL\_USER\_COLLECTIONS и ALL\_BOOKS\_IN\_COLLECTIONS

## 3.3 Разработка индексов базы данных

Индекс — объект базы данных, который используется для ускорения поиска данных. В случае большого количества строк в таблице, последовательный поиск данных может занимать много времени. Индекс формируется на основе значений одного или нескольких столбцов таблицы и указателей на соответствующие строки таблицы, что позволяет быстро искать строки, удовлетворяющие заданному критерию поиска. Использование индексов ускоряет работу с базой данных, потому что они имеют оптимизированную структуру для поиска, например, сбалансированное дерево.

Создание индексов представлено в листинге 3.3.

|  |
| --- |
| CREATE INDEX IDX\_USER\_NAME ON USERS(USER\_NAME);  CREATE INDEX IDX\_ROLE\_NAME ON ROLES(role\_name);  CREATE INDEX IDX\_COLLECTION\_NAME ON COLLECTIONS(collection\_name);  CREATE INDEX IDX\_BOOK\_NAME ON BOOKS(book\_name);  CREATE INDEX IDX\_CHAPTER\_NAME ON CHAPTERS(chapter\_name);  CREATE INDEX IDX\_GENRE\_NAME ON GENRES\_TYPES(genre\_name); |

Листинг 3.3 – Индексы таблиц

Применение индексов в базе данных значительно повышает скорость выполнения операций поиска, сортировки и фильтрации данных, особенно в случае больших объемов информации. Так были созданы:

IDX\_USER\_NAME - для ускорения поиска по имени пользователя;

IDX\_ROLE\_NAME - для ускорения поиска по названию роли;

IDX\_COLLECTION\_NAME - для ускорения поиска по имени сборника;

IDX\_BOOK\_NAME - для ускорения поиска по названию книги;

IDX\_CHAPTER\_NAME - для ускорения поиска по имени главы;

IDX\_GENRE\_NAME - для ускорения поиска по названию жанру;

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. А также использовать индексы на значениях, которые изменяются редко или вообще не меняются.

## 3.4 Разработка процедур базы данных

Подпрограммы PL / SQL называются блоками PL / SQL, которые могут быть вызваны с помощью набора параметров. PL / SQL предоставляет два вида подпрограмм: функции и процедуры. Процедуры не могут возвращать значений (но могут иметь выходные параметры) и имеют возможность в теле использовать DML-операторы. Всего для проекта было разработано 27 процедуры. На листинге 3.4 представлен пример реализации одной из процедур.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE CREATE\_CHAPTER(  book\_id IN NUMBER,  chapter\_name IN VARCHAR2,  chapter\_content IN VARCHAR2,  chapter\_publish\_date IN DATE,  chapter\_num IN NUMBER)  is  begin  insert into CHAPTERS (book\_id,chapter\_name ,chapter\_content, chapter\_publish\_date, chapter\_num)  values (book\_id,chapter\_name ,chapter\_content, chapter\_publish\_date, chapter\_num);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END; |

Листинг 3.4 – Процедура добавления главы

Данная процедура добавляет новую главу и в качестве параметров принимает идентификатор книги, название, путь к содержимому, дату публикации, порядковый номер главы в книге.

CREATE\_ROLE - процедура для добавления нового роли в базу данных;

DELETE\_ROLE - процедура для удаления данных роли из базы данных;

UPDATE\_ROLE - процедура для обновления данных роли в базе данных;

CREATE\_USER - процедура для добавления нового пользователя в базу данных;

DELETE\_USER - процедура для удаления данных пользователя из базы данных;

UPDATE\_USER - процедура для обновления данных пользователя в базе данных;

CREATE\_COLLECTION - процедура для добавления нового сборника в базу данных;

DELETE\_COLLECTION - процедура для удаления данных коллекции из базы данных;

UPDATE\_COLLECTION - процедура для обновления данных коллекции в базе данных;

CREATE\_BOOK - процедура для добавления новой книги в базу данных;

DELETE\_BOOK - процедура для удаления данных книги из базы данных;

UPDATE\_BOOK - процедура для обновления данных книги в базе данных;

CREATE\_CHAPTER - процедура для добавления новой главы в базу данных;

DELETE\_CHAPTER - процедура для удаления данных главы из базы данных;

UPDATE\_CHAPTER - процедура для обновления данных главы в базе данных;

CREATE\_COMMENT - процедура для добавления новой главы в базу данных;

DELETE\_COMMENT - процедура для удаления данных комментария из базы данных;

UPDATE\_COMMENT - процедура для обновления данных комментария в базе данных;

CREATE\_GENRES\_TYPE - процедура для добавления нового жанра в базу данных;

DELETE\_GENRES\_TYPE - процедура для удаления данных жанра из базы данных;

UPDATE\_GENRES\_TYPE - процедура для обновления данных жанра в базе данных;

CREATE\_BOOKS\_GENRE - процедура для добавления жанра в книгу;

DELETE\_BOOKS\_GENRE - процедура для удаления жанра из книги;

UPDATE\_BOOKS\_GENRE - процедура для обновления жанра в книге;

CREATE\_BOOKS\_COLLECTION - процедура для добавления книги в коллекцию;

DELETE\_BOOKS\_COLLECTION - процедура для удаления книги из сборника;

UPDATE\_BOOKS\_COLLECTION - процедура для обновления книги в сборнике;

Скрипты других процедур приведен в [приложении Б](#_Приложение_В).

## 3.5 Разработка функций базы данных

Функции в Oracle - это именованные блоки программного кода, которые принимают входные значения, обрабатывают их и возвращают результат. Функции используются для выполнения различных операций, например, для вычислений, обработки данных. В отличие от процедур функции используются для получения значений или их получение в обработанном виде. В листинге 3.5 указан код функции

|  |
| --- |
| CREATE OR REPLACE FUNCTION show\_users\_books (p\_user\_name IN USERS.user\_name%TYPE) RETURN sys\_refcursor  IS  ord\_cursor sys\_refcursor;  rcount INT;  id\_u NUMBER;  BEGIN  SELECT COUNT(\*)  INTO rcount  FROM USERS  WHERE USERS.user\_name = p\_user\_name;  IF rcount = 1 THEN  SELECT user\_id  INTO id\_u  FROM USERS  WHERE USERS.user\_name = p\_user\_name;  OPEN ord\_cursor FOR  SELECT creator\_id, book\_name, book\_publish\_date, description  FROM BOOKS  WHERE creator\_id = id\_u;  RETURN ord\_cursor;  ELSE  DBMS\_OUTPUT.PUT\_LINE('проверьте данные.');  RETURN NULL;  END IF;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('ошибка '||SQLCODE);  RETURN NULL;  END; |

Листинг 3.5 – Скрипт функции show\_users\_books

show\_users\_books. Данная функция выводит данные о книгах пользователя в виде курсора, на вход она принимает имя пользователя.

check\_authentication – функция проверки пароля для аутентификации;

## 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.

в листинге 3.6 приведен код триггера для проверки номера главы.

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_CHAPTER\_NUM  BEFORE INSERT ON CHAPTERS  FOR EACH ROW  DECLARE  CH\_NUM NUMBER(10);  CH\_LAST\_NUM NUMBER(10);  BEGIN  SELECT COUNT(\*)  INTO CH\_NUM  FROM CHAPTERS  WHERE CHAPTERS.book\_id = :new.book\_id AND CHAPTERS.chapter\_num = :new.chapter\_num;  SELECT MAX(chapter\_num)  INTO CH\_LAST\_NUM  FROM CHAPTERS  WHERE CHAPTERS.book\_id = :new.book\_id;  IF CH\_NUM != 0 THEN  IF CH\_LAST\_NUM != NULL OR CH\_LAST\_NUM != 0 THEN  :NEW.chapter\_num := CH\_LAST\_NUM + 1;  ELSE  :NEW.chapter\_num := 1;  END IF;  END IF;  END; |

Листинг 3.6 – Скрипт триггера CHECK\_CHAPTER\_NUM

Триггер CHECK\_CHAPTER\_NUM срабатывает при добавлении новой главы в таблицу CHAPTERS. Он получает текущее количество глав в книге и номер последней главы, после чего проверяет. Если нет глав или главы с указанным номером, то триггер разрешает добавление главы с выбранным номером, в противном случае триггер переназначает номер главы на следующий за последней главой

CHECK\_USER\_CREATE\_COLLECTION - проверяет роль пользователя перед добавлением нового сборника;

CHECK\_USER\_CREATE\_BOOK - проверяет роль пользователя перед добавлением новой книги;

CHECK\_CHAPTER\_NUM - проверяет номер главы перед добавлением;

CHECK\_CHAPTER\_DATE - проверяет даты публикации главы перед её добавлением;

CHECK\_COMMENT\_DATE - проверяет даты публикации комментария перед его добавлением;

CHECK\_BOOKS\_GENRES - проверяет отсутствие жанра в книге перед его добавлением;

CHECK\_BOOKS\_COLLECTIONS - проверяет отсутствие книги в коллекции перед её добавлением;

## 3.7 Создание 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 для подключения к базе данных: администратор, читатель, писатель.

Создание пользователей представлено в листинге 3.7.

|  |
| --- |
| CREATE ROLE RL\_ADMIN;  CREATE ROLE RL\_WRITTER;  CREATE ROLE RL\_READER;  CREATE USER ADMIN\_1 IDENTIFIED BY a123;  GRANT RL\_ADMIN TO ADMIN\_1;  alter user ADMIN\_1 default tablespace TS\_PRIME;  CREATE USER WRITTER\_1 IDENTIFIED BY w123;  GRANT RL\_WRITTER TO WRITTER\_1;  alter user WRITTER\_1 default tablespace TS\_PRIME;  CREATE USER READER\_1 IDENTIFIED BY r123;  GRANT RL\_READER TO READER\_1;  alter user READER\_1 default tablespace TS\_PRIME; |

Листинг 3.7 – Создание пользователей

Далее для созданного пользователя базы данных были выданы необходимые привилегии необходимые для функционала клиента. Код выдачи привилегий показан в приложении Д.

## 3.8 Вывод

В данном разделе была рассмотрена разработка объектов базы данных для интернет-магазина компьютерных комплектующих. Все рассмотренные объекты вместе создают сложную, но гибкую и эффективную структуру, которая обеспечивает правильное функционирование и управление данными в системе.

# 4 Описание процедур импорта и экспорта

Для таблицы BOOKS в базе данных доступна функция экспорта и импорта данных в формате xml. Это может пригодиться, если необходимо переместить данные на другой сервер или создать резервную копию.

Для реализации экспорта данных в xml, была разработана функция, результатом которой стало создание xml файла, с записанными строками из определенной таблицы в xml формат. Скрипт процедуры представлен в листинге 4.1.

|  |
| --- |
| CREATE OR REPLACE PROCEDURE BOOKS\_XML IS  xml\_data CLOB;  BEGIN  SELECT XMLELEMENT("BOOKS",  XMLFOREST(book\_id as "book\_id",creator\_id as "creator\_id",book\_name as "book\_name",book\_publish\_date as "book\_publish\_date",description as "description")).getClobVal()  INTO xml\_data  FROM BOOKS  WHERE ROWNUM = 1;  DBMS\_OUTPUT.PUT\_LINE(xml\_data);  END; |

Листинг 4.1 – процедура GenerateAddressXML

Для обратного экспорта данные в xml формат из файла, реализован запросом, который обращается к xml, и выбирает данные в обычные строки, это можно реализовать с помощью функции, либо по мере надобности записать данные во временную таблицу, либо таблицу в другой базе данных. Скрипт запроса представлен в [приложении Е](#_Приложение_Г).

Таким образом, пользователи базы данных могут легко экспортировать и импортировать данные в форматах xml, что делает управление базой данных более удобным и эффективным.

# 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 мы создали анонимный блок, и так как работаем именно с действующей базой данных, то можем использовать готовую процедуру для вставки данных в таблицу. Разработанный анонимный блок позволяет добавить большое количество строк за одно выполнение (листинг 5.1).

|  |
| --- |
| DECLARE  j NUMBER;  book\_id CHAPTERS.book\_id%TYPE;  chapter\_name CHAPTERS.chapter\_name%TYPE;  chapter\_content CHAPTERS.chapter\_content%TYPE;  chapter\_publish\_date CHAPTERS.chapter\_publish\_date%TYPE;  BEGIN  FOR j IN 1..1000000 LOOP  book\_id := DBMS\_RANDOM.VALUE(1, 5000);  chapter\_name := DBMS\_RANDOM.STRING('A', 10);  chapter\_content:= DBMS\_RANDOM.STRING('A', 30);  chapter\_publish\_date:= TO\_DATE(TRUNC(DBMS\_RANDOM.VALUE(TO\_CHAR(DATE '2000-01-01','J'),TO\_CHAR(DATE '2023-12-31','J'))),'J');  CREATE\_CHAPTER(book\_id,chapter\_name ,chapter\_content, chapter\_publish\_date, 1);  END LOOP;  end; |

Листинг 5.1 – Заполнение таблицы 1000000 строк

Для получения выборки данных использовался запрос, который представлен на листинге 5.2.

|  |
| --- |
| select \* from CHAPTERS where chapter\_name = '%Jsd%' |

Листинг 5.2 – Запрос к таблице CHAPTERS

Результаты выполнения запроса к таблице указывают на значительные затраты времени и ресурсов, особенно при сканировании всей таблицы и применении фильтра. Время планирования – 42 мс. Результаты запроса будут представлены на рисунке 5.1.

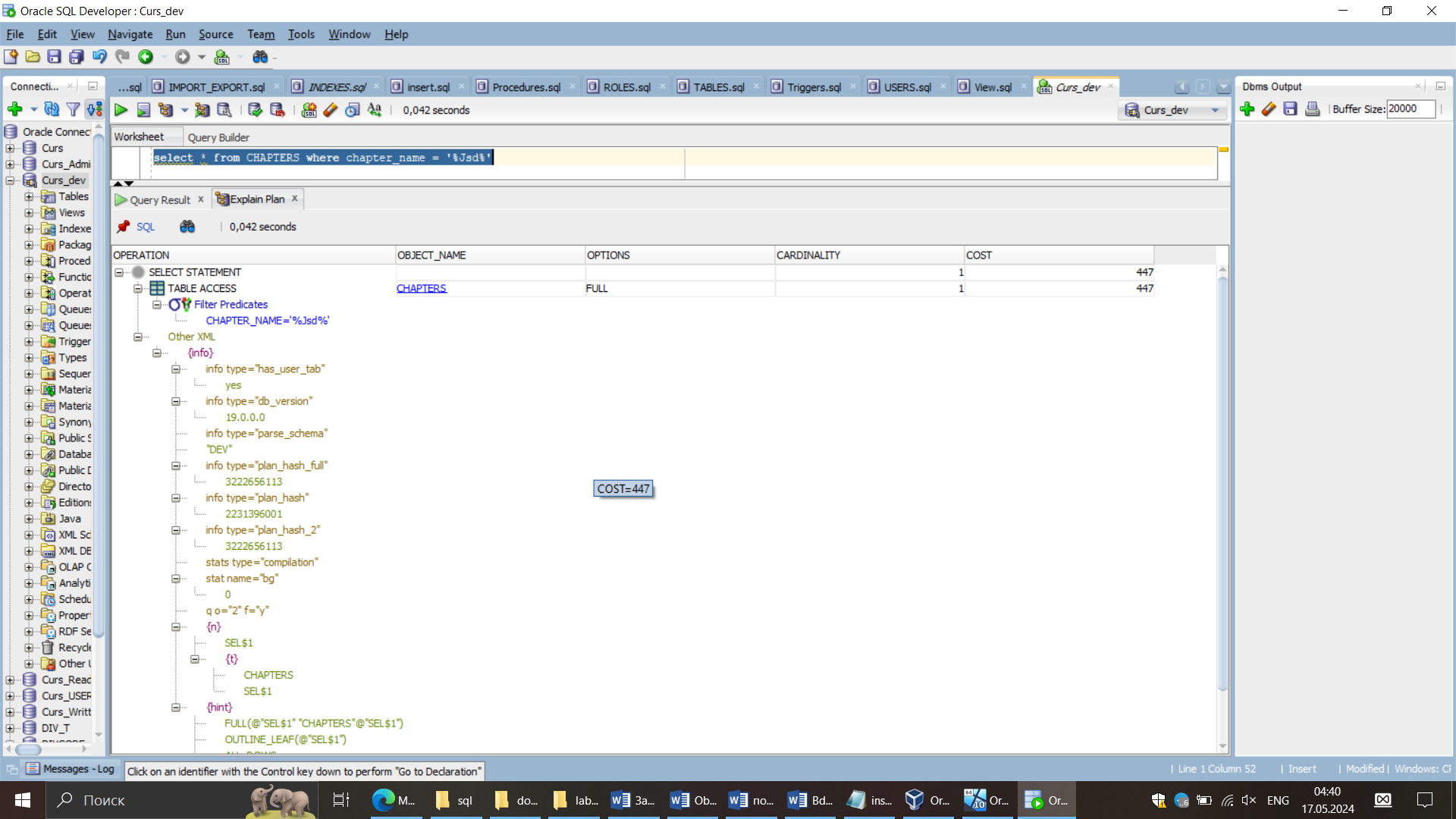


Рисунок 5.1 – Результат выполнения запроса

Для ускорения данного процесса создадим индекс на поле имя части так как именно по этому полю выполняется фильтрация. После создания индекса, можно повторить запрос и сравнить стоимость с предыдущим запросом. Результат будет представлен на рисунке 5.2.

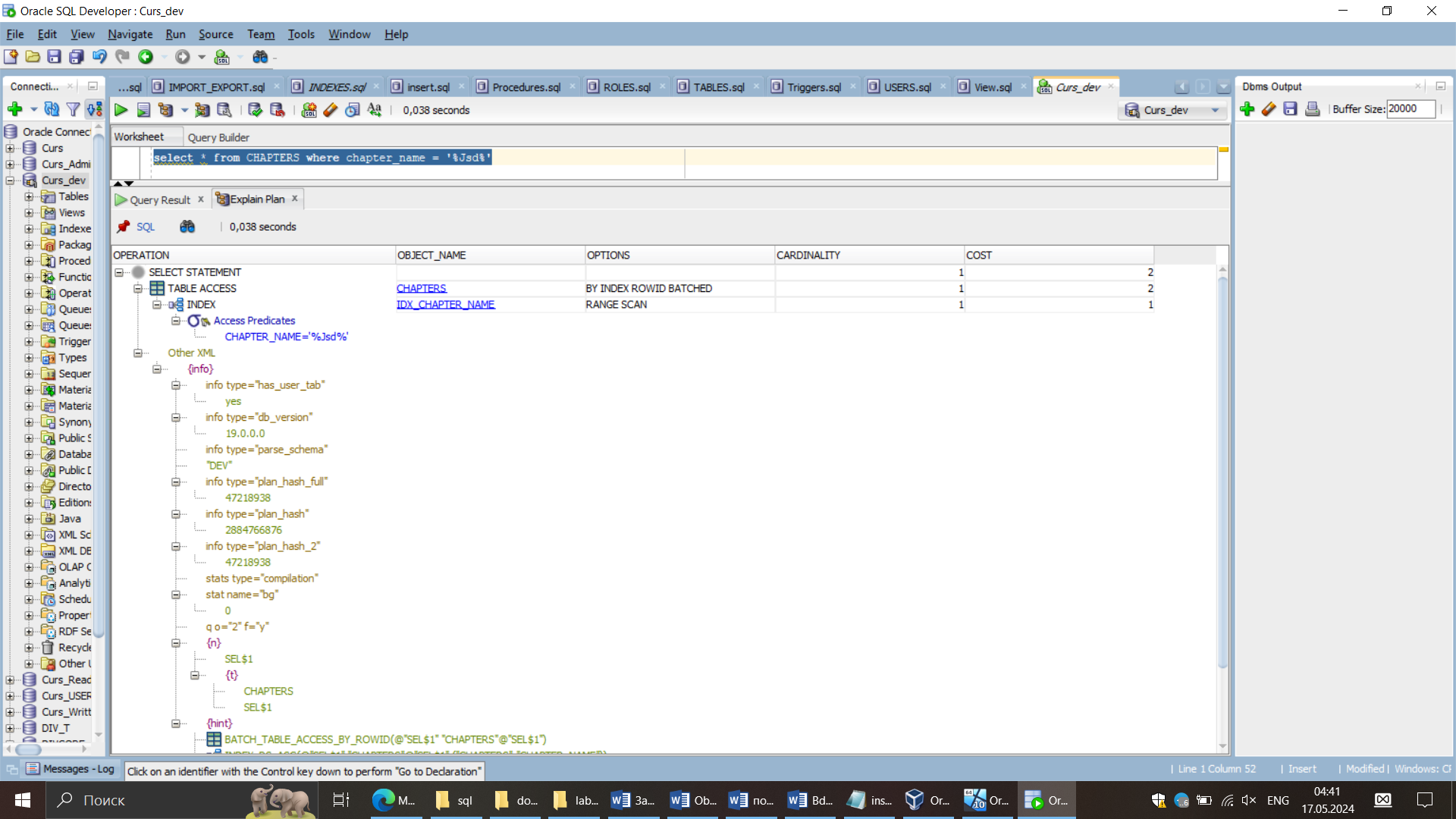


Рисунок 5.2 – Результат выполнения запроса

После создания индексов видно, что части стали генерировать намного быстрее. Время планирования заняло 38 мс.

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

# 6 Описание технологии и ее применение в базе данных

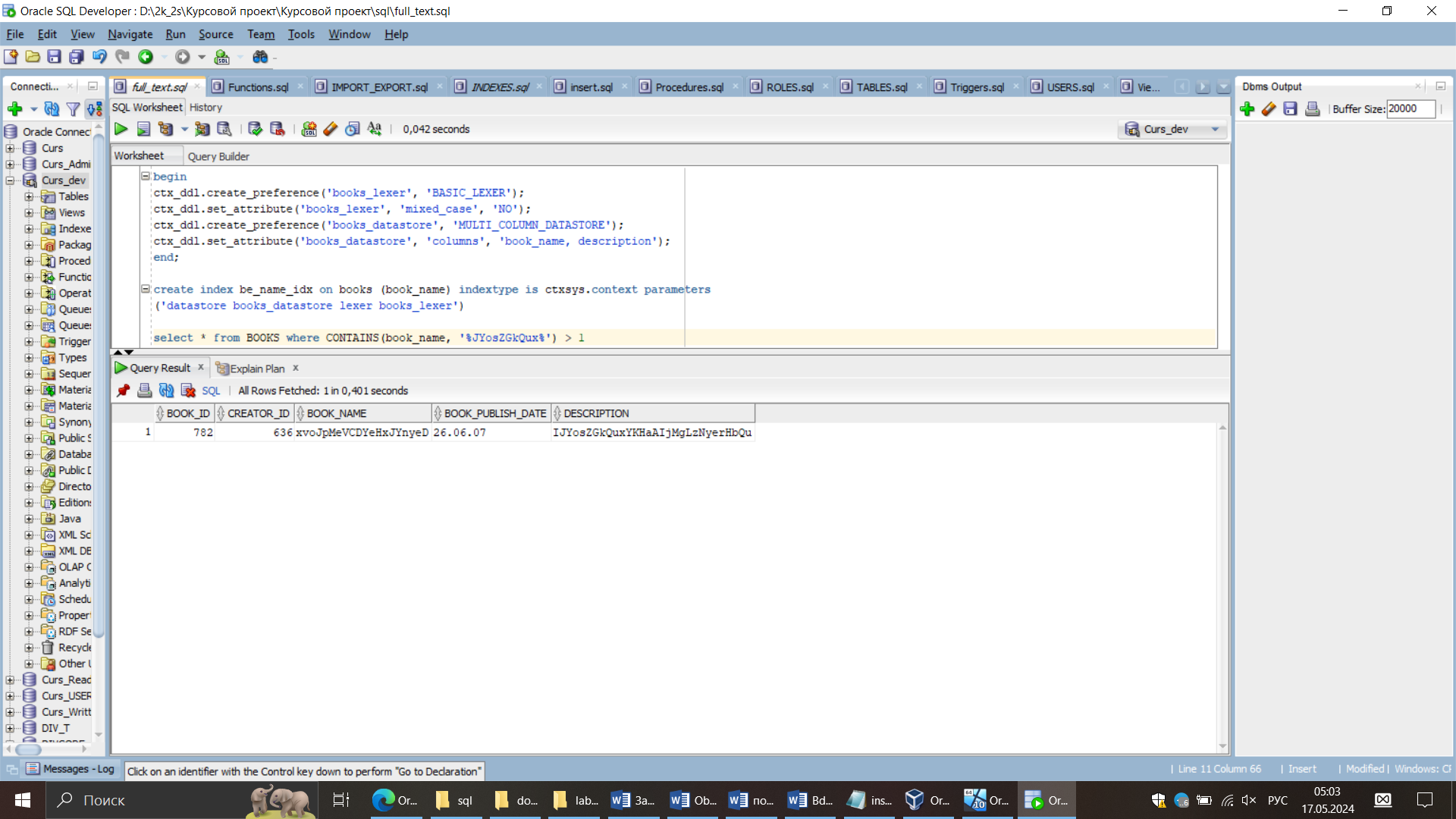
Full Text Search - технология позволяющая быстро находить заданную информацию в большом объеме текста. Для реализации используется специальный индекс c добавленными к нему преференций MULTI\_COLUMN\_DATASTORE и BASIC\_LEXER.

На листинге 6.1 показано создание индекса.

|  |
| --- |
| begin  ctx\_ddl.create\_preference('books\_lexer', 'BASIC\_LEXER');  ctx\_ddl.set\_attribute('books\_lexer', 'mixed\_case', 'NO');  ctx\_ddl.create\_preference('books\_datastore', 'MULTI\_COLUMN\_DATASTORE');  ctx\_ddl.set\_attribute('books\_datastore', 'columns', 'book\_name, description');  end;  create index be\_name\_idx on books (book\_name) indextype is ctxsys.context parameters  ('datastore books\_datastore lexer books\_lexer') |

Листинг 6.1 – Создание индекса

Также данный индекс позволяет проводить поиск по смежным столбцам, тем самым улучшая поиск. Индекс создан для таблицы BOOKS к столбцу book\_name. При выборке по столбцу book\_name индекс позволяет проверять значения и из столбца description. Результат выполнения запрос к таблице показан на рисунке 6.1



Листинг 6.1 – Создание индекса

# 7 Руководство пользователя

При входе в магазин, мы будем иметь статус гостя, это значит – ограниченный функционал. Главная функция у гостя – регистрация (процедура CREATE\_USER). При регистрации указывается имя, почта, роль(Читатель, Писатель), пароль.

После регистрации, пользователь уже имеет одноименный статус – Пользователь и выбранную роль.

В роли читателя пользователь может создать сборник (CREATE\_COLLECTION) и управлять ими. Также он может добавлять книги в свои коллекции(CREATE\_BOOKS\_COLLECTION) и управлять ими в них

В роли читателя пользователь может создавать книги(CREATE\_BOOK), управлять ими и добавлять к ним жанры(CREATE\_BOOKS\_GENRE) и части(CREATE\_CHAPTER)

Есть ещё пользователь в роли администратора, который может удалять любые данные и создавать новые жанры(CREATE\_GENRES\_TYPE)

Для всех пользователей доступен просмотр книг, частей, жанров и сборников. Для всех пользователей доступно возможность написания отзывов к главам(CREATE\_COMMENT)

Вход с учетной записью разработчика дает нам полное управление базой данных библиотеки.

Реализация каждой из вышеупомянутой роли приведена в [приложении Д](#_Приложение_Д).

# Заключение

В данном проекте была поставлена задача разработки базы данных для сервиса для публикации и чтения электронных книг с применение Full Text Search- технологии в СУБД oracle 19c.

В ходе выполнения проекта были использованы различные объекты, такие как таблицы, представления, процедуры, индексы триггеры, специальные функции, чтобы обеспечить структурированное хранение данных, изолированность данных таблиц от прямого вмешательства 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 xml, что значительно облегчило управление данными и повысило удобство использования базы данных.

Разработанная паза данных для сервиса для публикации и чтения электронных книг обладает особенность. За счёт реализованной технологии Full Text Search поиск книг по базе данных намного шире и эффективнее

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. Сервиса для публикации и чтения электронных книг выполняет необходимую функцию хранения и управления данными.

# Список используемых источников

1. Oracle.com [Электронный ресурс] – <https://www.oracle.com> – Дата доступа 12.04.2024
2. dreamix.eu [Электронный ресурс] – https://dreamix.eu – Дата доступа 18.04.2024
3. oracleplsql.ry [Электронный ресурс] – https://oracleplsql.ru – Дата доступа 21.04.2024
4. ittutorial.org [Электронный ресурс] – https://ittutorial.org – Дата доступа 25.04.2024

# Приложение А. Листинг создания таблиц

|  |
| --- |
| ---------------------TABLE ROLE ------------------------------------  CREATE TABLE ROLES(  role\_id NUMBER(10) generated by default as identity PRIMARY KEY,  role\_name VARCHAR2(50)  )TABLESPACE TS\_PRIME;  ---------------------TABLE USERS -----------------------------------  CREATE TABLE USERS (  user\_id NUMBER(10) generated by default as identity PRIMARY KEY,  user\_name VARCHAR2(100),  user\_email VARCHAR2(100),  user\_password VARCHAR2(100),  user\_role NUMBER(10),  CONSTRAINT FK\_USERS\_ROLES FOREIGN KEY (USER\_ROLE) REFERENCES ROLES (ROLE\_ID)  )TABLESPACE TS\_PRIME;  ---------------------TABLE COLLECTIONS -----------------------------  CREATE TABLE COLLECTIONS (  collection\_id NUMBER(10) generated by default as identity PRIMARY KEY,  creator\_id NUMBER(10),  collection\_name VARCHAR2(100),  description VARCHAR2(200),  CONSTRAINT FK\_COLLECTIONS\_USERS FOREIGN KEY (CREATOR\_ID) REFERENCES USERS (USER\_ID)  )TABLESPACE TS\_PRIME;  ---------------------TABLE BOOKS -----------------------------------  CREATE TABLE BOOKS (  book\_id NUMBER(10) generated by default as identity PRIMARY KEY,  creator\_id NUMBER(10),  book\_name VARCHAR2(100),  book\_publish\_date DATE,  description VARCHAR2(200),  CONSTRAINT FK\_BOOKS\_USERS FOREIGN KEY (CREATOR\_ID) REFERENCES USERS (USER\_ID)  )TABLESPACE TS\_PRIME;  ---------------------TABLE CHAPTERS --------------------------------  CREATE TABLE CHAPTERS (  chapter\_id NUMBER(10) generated by default as identity PRIMARY KEY,  book\_id NUMBER(10),  chapter\_name VARCHAR2(100),  chapter\_content VARCHAR2(100),  chapter\_publish\_date DATE,  chapter\_num NUMBER(10),  CONSTRAINT FK\_CHAPTERS\_BOOKS FOREIGN KEY (BOOK\_ID) REFERENCES BOOKS (BOOK\_ID)  )TABLESPACE TS\_PRIME;  ---------------------TABLE COMMENTS --------------------------------  CREATE TABLE COMMENTS (  comment\_id NUMBER(10) generated by default as identity PRIMARY KEY,  chapter\_id NUMBER(10),  user\_id NUMBER(10),  comment\_content VARCHAR2(200),  comment\_publish\_date DATE,  CONSTRAINT FK\_COMMENTS\_CHAPTERS FOREIGN KEY (CHAPTER\_ID) REFERENCES CHAPTERS (CHAPTER\_ID),  CONSTRAINT FK\_COMMENTS\_USERS FOREIGN KEY (USER\_ID) REFERENCES USERS (USER\_ID)  )TABLESPACE TS\_PRIME;  ---------------------TABLE GENRES\_TYPES ----------------------------  CREATE TABLE GENRES\_TYPES (  genre\_id NUMBER(10) generated by default as identity PRIMARY KEY,  genre\_name VARCHAR2(50)  )TABLESPACE TS\_PRIME;  ---------------------TABLE BOOKS\_GENRES ----------------------------  CREATE TABLE BOOKS\_GENRES (  book\_genre\_id NUMBER(10) generated by default as identity PRIMARY KEY,  genre\_id NUMBER(10),  book\_id NUMBER(10),  CONSTRAINT FK\_BOOKS\_GENRES\_GENRES\_TYPES FOREIGN KEY (GENRE\_ID) REFERENCES GENRES\_TYPES (GENRE\_ID),  CONSTRAINT FK\_BOOKS\_GENRES\_BOOKS FOREIGN KEY (BOOK\_ID) REFERENCES BOOKS (BOOK\_ID)  )TABLESPACE TS\_PRIME;  ---------------------TABLE BOOKS\_COLLECTIONS -----------------------  CREATE TABLE BOOKS\_COLLECTIONS (  book\_collection\_id NUMBER(10) generated by default as identity PRIMARY KEY,  collection\_id NUMBER(10),  book\_id NUMBER(10),  CONSTRAINT FK\_BOOKS\_COLLECTIONS\_GENRES\_COLLECTIONS FOREIGN KEY (collection\_id) REFERENCES COLLECTIONS (collection\_id),  CONSTRAINT FK\_BOOKS\_COLLECTIONS\_BOOKS FOREIGN KEY (BOOK\_ID) REFERENCES BOOKS (BOOK\_ID)  )TABLESPACE TS\_PRIME; |

# Приложение Б. Листинг процедур базы данных

|  |
| --- |
| ----------ROLES--------------  CREATE OR REPLACE PROCEDURE CREATE\_ROLE(  role\_name IN VARCHAR2)  IS  BEGIN  INSERT INTO ROLES(ROLE\_NAME)values(role\_name);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_ROLE(  role\_id IN NUMBER)  IS  BEGIN  DELETE FROM ROLES WHERE role\_id = role\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_ROLE(  role\_id IN NUMBER,  role\_name IN VARCHAR2)  IS  BEGIN  UPDATE ROLES SET role\_name = role\_name WHERE role\_id = role\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  ------------USERS-------------  CREATE OR REPLACE PROCEDURE CREATE\_USER(  user\_name IN VARCHAR2,  user\_email IN VARCHAR2,  user\_password IN VARCHAR2,  user\_role IN NUMBER)  is  begin  insert into USERS (user\_name, user\_email, user\_password, user\_role)  values (user\_name, user\_email, user\_password, user\_role);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_USER(  user\_id IN NUMBER)  IS  BEGIN  DELETE FROM USERS WHERE user\_id = user\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_USER(  user\_id IN NUMBER,  user\_name IN VARCHAR2,  user\_email IN VARCHAR2,  user\_password IN VARCHAR2,  user\_role IN NUMBER)  IS  BEGIN  UPDATE USERS SET  user\_name = user\_name,  user\_email = user\_email,  user\_password = user\_password,  user\_role = user\_role  WHERE user\_id = user\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  -------Collections---  CREATE OR REPLACE PROCEDURE CREATE\_COLLECTION(  creator\_id IN NUMBER,  collection\_name IN VARCHAR2,  description IN VARCHAR2)  is  begin  insert into COLLECTIONS (creator\_id,collection\_name,description)  values (creator\_id,collection\_name,description);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_COLLECTION(  collection\_id IN NUMBER)  IS  BEGIN  DELETE FROM COLLECTIONS WHERE collection\_id = collection\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_COLLECTION(  collection\_id IN NUMBER,  collection\_name IN VARCHAR2,  description IN VARCHAR2)  IS  BEGIN  UPDATE COLLECTIONS SET  collection\_name = collection\_name,  description = description  WHERE collection\_id = collection\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  ----------BOOKS-----------  CREATE OR REPLACE PROCEDURE CREATE\_BOOK(  creator\_id in number,  book\_name in VARCHAR2,  book\_publish\_date in DATE,  description in varchar2)  is  begin  insert into BOOKS (creator\_id, book\_name, book\_publish\_date, description)  values (creator\_id, book\_name, book\_publish\_date, description);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_BOOK(  book\_id IN NUMBER)  IS  BEGIN  DELETE FROM BOOKS WHERE book\_id = book\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_BOOK(  book\_id in number,  book\_name in VARCHAR2,  description in varchar2)  is  begin  UPDATE BOOKS SET  book\_name = book\_name,  description = description  WHERE book\_id = book\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  --------CHAPTERS--------  CREATE OR REPLACE PROCEDURE CREATE\_CHAPTER(  book\_id IN NUMBER,  chapter\_name IN VARCHAR2,  chapter\_content IN VARCHAR2,  chapter\_publish\_date IN DATE,  chapter\_num IN NUMBER)  is  begin  insert into CHAPTERS (book\_id,chapter\_name ,chapter\_content, chapter\_publish\_date, chapter\_num)  values (book\_id,chapter\_name ,chapter\_content, chapter\_publish\_date, chapter\_num);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_CHAPTER(  chapter\_id IN NUMBER)  IS  BEGIN  DELETE FROM CHAPTERS WHERE chapter\_id = chapter\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_CHAPTER(  chapter\_id IN NUMBER,  chapter\_name IN VARCHAR2,  chapter\_content IN VARCHAR2,  chapter\_num IN NUMBER)  is  begin  UPDATE CHAPTERS SET  chapter\_name = chapter\_name,  chapter\_content = chapter\_content,  chapter\_num = chapter\_num  WHERE chapter\_id = chapter\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  -------COMMENTS-------  CREATE OR REPLACE PROCEDURE CREATE\_COMMENT(  chapter\_id IN NUMBER,  user\_id IN NUMBER,  comment\_content IN VARCHAR2,  comment\_publish\_date IN DATE)  is  begin  insert into COMMENTS (chapter\_id ,user\_id ,comment\_content ,comment\_publish\_date)  values (chapter\_id ,user\_id ,comment\_content ,comment\_publish\_date);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_COMMENT(  comment\_id IN NUMBER)  IS  BEGIN  DELETE FROM COMMENTS WHERE comment\_id = comment\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_COMMENT(  comment\_id IN NUMBER,  comment\_content IN VARCHAR2)  is  begin  UPDATE COMMENTS SET  comment\_content = comment\_content  WHERE comment\_id = comment\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  -------GENRES\_TYPES------  CREATE OR REPLACE PROCEDURE CREATE\_GENRES\_TYPE(  genre\_name IN VARCHAR2)  IS  BEGIN  INSERT INTO GENRES\_TYPES(genre\_name)values(genre\_name);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_GENRES\_TYPE(  genre\_id IN NUMBER)  IS  BEGIN  DELETE FROM GENRES\_TYPES WHERE genre\_id = genre\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_GENRES\_TYPE(  genre\_id IN NUMBER,  genre\_name IN VARCHAR2)  IS  BEGIN  UPDATE GENRES\_TYPES SET genre\_name = genre\_name WHERE genre\_id = genre\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  -------BOOKS\_GENRES------  CREATE OR REPLACE PROCEDURE CREATE\_BOOKS\_GENRE(  genre\_id IN NUMBER,  book\_id IN NUMBER)  IS  BEGIN  INSERT INTO BOOKS\_GENRES(genre\_id ,book\_id)values(genre\_id ,book\_id);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_BOOKS\_GENRE(  book\_genre\_id IN NUMBER)  IS  BEGIN  DELETE FROM BOOKS\_GENRES WHERE book\_genre\_id = book\_genre\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_BOOKS\_GENRE(  book\_genre\_id IN NUMBER,  genre\_id IN NUMBER,  book\_id IN NUMBER)  IS  BEGIN  UPDATE BOOKS\_GENRES SET genre\_id = genre\_id , book\_id=book\_id  WHERE book\_genre\_id = book\_genre\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  -------BOOKS\_COLLECTIONS------  CREATE OR REPLACE PROCEDURE CREATE\_BOOKS\_COLLECTION(  collection\_id IN NUMBER,  book\_id IN NUMBER)  IS  BEGIN  INSERT INTO BOOKS\_COLLECTIONS(collection\_id ,book\_id)  values(collection\_id ,book\_id);  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE DELETE\_BOOKS\_COLLECTION(  book\_collection\_id IN NUMBER)  IS  BEGIN  DELETE FROM BOOKS\_COLLECTIONS WHERE book\_collection\_id = book\_collection\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END;  CREATE OR REPLACE PROCEDURE UPDATE\_BOOKS\_COLLECTION(  book\_collection\_id IN NUMBER,  collection\_id IN NUMBER,  book\_id IN NUMBER)  IS  BEGIN  UPDATE BOOKS\_COLLECTIONS SET collection\_id = collection\_id , book\_id=book\_id  WHERE book\_collection\_id = book\_collection\_id;  EXCEPTION  WHEN OTHERS THEN  DBMS\_OUTPUT.PUT\_LINE('Произошла ошибка: ' || SQLERRM);  END; |

# Приложение В. Листинг функций базы данных

|  |
| --- |
| CREATE OR REPLACE FUNCTION show\_users\_books (p\_user\_name IN USERS.user\_name%TYPE) RETURN sys\_refcursor IS  ord\_cursor sys\_refcursor;  rcount INT;  id\_u NUMBER;  BEGIN  SELECT COUNT(\*)  INTO rcount  FROM USERS  WHERE USERS.user\_name = p\_user\_name;  IF rcount = 1 THEN  SELECT user\_id  INTO id\_u  FROM USERS  WHERE USERS.user\_name = p\_user\_name;  OPEN ord\_cursor FOR  SELECT creator\_id, book\_name, book\_publish\_date, description  FROM BOOKS  WHERE creator\_id = id\_u;  RETURN ord\_cursor;  ELSE DBMS\_OUTPUT.PUT\_LINE('проверьте данные.');  RETURN NULL;  END IF;  EXCEPTION  WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('ошибка '||SQLCODE);  RETURN NULL;  END;  CREATE OR REPLACE FUNCTION check\_authentication (p\_user\_name IN USERS.user\_name%TYPE, p\_password IN USERS.user\_password%TYPE) RETURN Boolean IS  num\_c NUMBER;  BEGIN  SELECT COUNT(\*)  INTO num\_c  FROM USERS  WHERE USERS.user\_name = p\_user\_name and USERS.user\_password = p\_password;  IF num\_c < 1 or num\_c = null then  return false;  else  return true;  end if;  EXCEPTION  WHEN OTHERS THEN DBMS\_OUTPUT.PUT\_LINE('ошибка '||SQLCODE);  RETURN NULL;  end; |

# Приложение Г. Листинг представлений базы данных

|  |
| --- |
| CREATE OR REPLACE VIEW ALL\_ROLES AS  SELECT \* FROM ROLES;  CREATE OR REPLACE VIEW ALL\_USER AS  SELECT user\_id, user\_name, user\_email, user\_role FROM USERS;  CREATE OR REPLACE VIEW ALL\_COLLECTIONS AS  SELECT \* FROM COLLECTIONS;  CREATE OR REPLACE VIEW ALL\_BOOKS AS  SELECT \* FROM BOOKS;  CREATE OR REPLACE VIEW ALL\_CHAPTERS AS  SELECT \* FROM CHAPTERS;  CREATE OR REPLACE VIEW ALL\_COMMENTS AS  SELECT \* FROM COMMENTS;  CREATE OR REPLACE VIEW ALL\_GENRES\_TYPES AS  SELECT \* FROM GENRES\_TYPES;  CREATE OR REPLACE VIEW ALL\_BOOKS\_GENRES AS  SELECT \* FROM BOOKS\_GENRES;  CREATE OR REPLACE VIEW ALL\_BOOKS\_COLLECTIONS AS  SELECT \* FROM BOOKS\_COLLECTIONS;  CREATE OR REPLACE VIEW ALL\_USER\_ROLES AS  SELECT USER\_ID, USER\_NAME, ROLE\_NAME  FROM USERS  JOIN ROLES ON USERS.USER\_ROLE = ROLES.ROLE\_ID  ORDER BY USER\_NAME;  CREATE OR REPLACE VIEW ALL\_USER\_BOOKS AS  SELECT USER\_ID, USER\_NAME, BOOK\_ID, BOOK\_NAME  FROM USERS  JOIN BOOKS ON USERS.USER\_ID = BOOKS.creator\_id  ORDER BY USER\_NAME;  CREATE OR REPLACE VIEW ALL\_USER\_COLLECTIONS AS  SELECT USER\_ID, USER\_NAME, COLLECTION\_ID, COLLECTION\_NAME  FROM USERS  JOIN COLLECTIONS ON USERS.USER\_ID = COLLECTIONS.CREATOR\_ID  ORDER BY USER\_NAME;  CREATE OR REPLACE VIEW ALL\_BOOKS\_IN\_COLLECTIONS AS  SELECT COLLECTIONS.COLLECTION\_ID, COLLECTION\_NAME, BOOKS.BOOK\_ID, BOOK\_NAME  FROM BOOKS  JOIN BOOKS\_COLLECTIONS ON BOOKS.BOOK\_ID = BOOKS\_COLLECTIONS.BOOK\_ID  JOIN COLLECTIONS ON COLLECTIONS.COLLECTION\_ID = BOOKS\_COLLECTIONS.COLLECTION\_ID  ORDER BY COLLECTION\_NAME;  CREATE OR REPLACE VIEW ALL\_CHAPTERS\_IN\_BOOK AS  SELECT BOOKS.book\_id, book\_name, book\_publish\_date, chapter\_id, chapter\_name, chapter\_publish\_date  FROM CHAPTERS  JOIN BOOKS ON BOOKS.book\_id = CHAPTERS.book\_id  order by book\_name, chapter\_name; |

# Приложение Д. Листинг кода создания пользователей

|  |
| --- |
| CREATE ROLE RL\_ADMIN;  CREATE ROLE RL\_WRITTER;  CREATE ROLE RL\_READER;  ------------ADMIN-------  GRANT CONNECT TO RL\_ADMIN;  GRANT CREATE SESSION TO RL\_ADMIN;  GRANT SELECT ON ALL\_ROLES TO RL\_ADMIN;  GRANT SELECT ON ALL\_USER TO RL\_ADMIN;  GRANT SELECT ON ALL\_COLLECTIONS TO RL\_ADMIN;  GRANT SELECT ON ALL\_BOOKS TO RL\_ADMIN;  GRANT SELECT ON ALL\_CHAPTERS TO RL\_ADMIN;  GRANT SELECT ON ALL\_COMMENTS TO RL\_ADMIN;  GRANT SELECT ON ALL\_GENRES\_TYPES TO RL\_ADMIN;  GRANT SELECT ON ALL\_BOOKS\_GENRES TO RL\_ADMIN;  GRANT SELECT ON ALL\_BOOKS\_COLLECTIONS TO RL\_ADMIN;  GRANT SELECT ON ALL\_USER\_ROLES TO RL\_ADMIN;  GRANT SELECT ON ALL\_USER\_BOOKS TO RL\_ADMIN;  GRANT SELECT ON ALL\_USER\_COLLECTIONS TO RL\_ADMIN;  GRANT SELECT ON ALL\_BOOKS\_IN\_COLLECTIONS TO RL\_ADMIN;  GRANT SELECT ON ALL\_CHAPTERS\_IN\_BOOK TO RL\_ADMIN;  GRANT EXECUTE ON CREATE\_USER TO RL\_ADMIN;  GRANT EXECUTE ON DELETE\_USER TO RL\_ADMIN;  GRANT EXECUTE ON UPDATE\_USER TO RL\_ADMIN;  GRANT EXECUTE ON DELETE\_COLLECTION TO RL\_ADMIN;  GRANT EXECUTE ON CREATE\_COMMENT TO RL\_ADMIN;  GRANT EXECUTE ON DELETE\_COMMENT TO RL\_ADMIN;  GRANT EXECUTE ON UPDATE\_COMMENT TO RL\_ADMIN;  GRANT EXECUTE ON CREATE\_GENRES\_TYPE TO RL\_ADMIN;  GRANT EXECUTE ON DELETE\_COLLECTION TO RL\_ADMIN;  GRANT EXECUTE ON DELETE\_GENRES\_TYPE TO RL\_ADMIN;  GRANT EXECUTE ON UPDATE\_GENRES\_TYPE TO RL\_ADMIN;  GRANT EXECUTE ON DELETE\_BOOKS\_COLLECTION TO RL\_ADMIN;  ----------Writter-------  GRANT CONNECT TO RL\_WRITTER;  GRANT CREATE SESSION TO RL\_WRITTER;  GRANT SELECT ON ALL\_COLLECTIONS TO RL\_WRITTER;  GRANT SELECT ON ALL\_BOOKS TO RL\_WRITTER;  GRANT SELECT ON ALL\_CHAPTERS TO RL\_WRITTER;  GRANT SELECT ON ALL\_COMMENTS TO RL\_WRITTER;  GRANT SELECT ON ALL\_GENRES\_TYPES TO RL\_WRITTER;  GRANT SELECT ON ALL\_BOOKS\_GENRES TO RL\_WRITTER;  GRANT SELECT ON ALL\_BOOKS\_COLLECTIONS TO RL\_WRITTER;  GRANT SELECT ON ALL\_USER\_BOOKS TO RL\_WRITTER;  GRANT SELECT ON ALL\_USER\_COLLECTIONS TO RL\_WRITTER;  GRANT SELECT ON ALL\_BOOKS\_IN\_COLLECTIONS TO RL\_WRITTER;  GRANT SELECT ON ALL\_CHAPTERS\_IN\_BOOK TO RL\_WRITTER;  GRANT EXECUTE ON CREATE\_BOOK TO RL\_WRITTER;  GRANT EXECUTE ON DELETE\_BOOK TO RL\_WRITTER;  GRANT EXECUTE ON UPDATE\_BOOK TO RL\_WRITTER;  GRANT EXECUTE ON CREATE\_CHAPTER TO RL\_WRITTER;  GRANT EXECUTE ON DELETE\_CHAPTER TO RL\_WRITTER;  GRANT EXECUTE ON UPDATE\_CHAPTER TO RL\_WRITTER;  GRANT EXECUTE ON CREATE\_COMMENT TO RL\_WRITTER;  GRANT EXECUTE ON DELETE\_COMMENT TO RL\_WRITTER;  GRANT EXECUTE ON UPDATE\_COMMENT TO RL\_WRITTER;  GRANT EXECUTE ON CREATE\_BOOKS\_GENRE TO RL\_WRITTER;  GRANT EXECUTE ON DELETE\_BOOKS\_GENRE TO RL\_WRITTER;  GRANT EXECUTE ON UPDATE\_BOOKS\_GENRE TO RL\_WRITTER;  -------------Reader-------------  GRANT CONNECT TO RL\_READER;  GRANT CREATE SESSION TO RL\_READER;  GRANT SELECT ON ALL\_COLLECTIONS TO RL\_READER;  GRANT SELECT ON ALL\_BOOKS TO RL\_READER;  GRANT SELECT ON ALL\_CHAPTERS TO RL\_READER;  GRANT SELECT ON ALL\_COMMENTS TO RL\_READER;  GRANT SELECT ON ALL\_GENRES\_TYPES TO RL\_READER;  GRANT SELECT ON ALL\_BOOKS\_GENRES TO RL\_READER;  GRANT SELECT ON ALL\_BOOKS\_COLLECTIONS TO RL\_READER;  GRANT SELECT ON ALL\_USER\_BOOKS TO RL\_READER;  GRANT SELECT ON ALL\_USER\_COLLECTIONS TO RL\_READER;  GRANT SELECT ON ALL\_BOOKS\_IN\_COLLECTIONS TO RL\_READER;  GRANT SELECT ON ALL\_CHAPTERS\_IN\_BOOK TO RL\_READER;  GRANT EXECUTE ON CREATE\_COMMENT TO RL\_READER;  GRANT EXECUTE ON DELETE\_COMMENT TO RL\_READER;  GRANT EXECUTE ON UPDATE\_COMMENT TO RL\_READER;  GRANT EXECUTE ON CREATE\_COLLECTION TO RL\_READER;  GRANT EXECUTE ON DELETE\_COLLECTION TO RL\_READER;  GRANT EXECUTE ON UPDATE\_COLLECTION TO RL\_READER;  GRANT EXECUTE ON CREATE\_BOOKS\_COLLECTION TO RL\_READER;  GRANT EXECUTE ON DELETE\_BOOKS\_COLLECTION TO RL\_READER;  GRANT EXECUTE ON UPDATE\_BOOKS\_COLLECTION TO RL\_READER;  CREATE USER ADMIN\_1 IDENTIFIED BY a123;  GRANT RL\_ADMIN TO ADMIN\_1;  alter user ADMIN\_1 default tablespace TS\_PRIME;  CREATE USER WRITTER\_1 IDENTIFIED BY w123;  GRANT RL\_WRITTER TO WRITTER\_1;  alter user WRITTER\_1 default tablespace TS\_PRIME;  CREATE USER READER\_1 IDENTIFIED BY r123;  GRANT RL\_READER TO READER\_1;  alter user READER\_1 default tablespace TS\_PRIME; |

# Приложение Е. Листинг процедур для импорта и экспорта в xml

|  |
| --- |
| CREATE OR REPLACE PROCEDURE BOOKS\_XML IS  xml\_data CLOB;  BEGIN  SELECT XMLELEMENT("BOOKS",  XMLFOREST(book\_id as "book\_id",creator\_id as "creator\_id",book\_name as "book\_name",book\_publish\_date as "book\_publish\_date",description as "description")).getClobVal()  INTO xml\_data  FROM BOOKS  WHERE ROWNUM = 1;  DBMS\_OUTPUT.PUT\_LINE(xml\_data);  END;  CALL BOOKS\_XML();  CREATE OR REPLACE Procedure import\_xml\_book  (xml\_text in nvarchar2) is  v\_xml XMLType;  BEGIN  v\_xml := XMLType(xml\_text);  CREATE\_BOOK(v\_xml.extract('/BOOKS/creator\_id/text()').getNumberVal(),  v\_xml.extract('/BOOKS/book\_name/text()').getStringVal(),  TO\_DATE(v\_xml.extract('/BOOKS/book\_publish\_date/text()').getStringVal(),'J'),  v\_xml.extract('/BOOKS/description/text()').getStringVal());  END; |

# Приложение Ж. Листинг Триггеров

|  |
| --- |
| CREATE OR REPLACE TRIGGER CHECK\_USER\_CREATE\_COLLECTION  BEFORE INSERT ON COLLECTIONS  FOR EACH ROW  DECLARE  ROLE\_NUM NUMBER(10);  BEGIN  SELECT user\_role  INTO ROLE\_NUM  FROM USERS  WHERE USERS.user\_id = :new.creator\_id;  IF ROLE\_NUM != 2 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'wrong user');  END IF;  END;  CREATE OR REPLACE TRIGGER CHECK\_USER\_CREATE\_BOOK  BEFORE INSERT ON BOOKS  FOR EACH ROW  DECLARE  ROLE\_NUM NUMBER(10);  BEGIN  SELECT user\_role  INTO ROLE\_NUM  FROM USERS  WHERE USERS.user\_id = :new.creator\_id;  IF ROLE\_NUM != 1 THEN  RAISE\_APPLICATION\_ERROR(-20001, 'wrong user');  END IF;  END;  CREATE OR REPLACE TRIGGER CHECK\_CHAPTER\_NUM  BEFORE INSERT ON CHAPTERS  FOR EACH ROW  DECLARE  CH\_NUM NUMBER(10);  CH\_LAST\_NUM NUMBER(10);  BEGIN  SELECT COUNT(\*)  INTO CH\_NUM  FROM CHAPTERS  WHERE CHAPTERS.book\_id = :new.book\_id AND CHAPTERS.chapter\_num = :new.chapter\_num;  SELECT MAX(chapter\_num)  INTO CH\_LAST\_NUM  FROM CHAPTERS  WHERE CHAPTERS.book\_id = :new.book\_id;  IF CH\_NUM != 0 THEN  IF CH\_LAST\_NUM != NULL OR CH\_LAST\_NUM != 0 THEN  :NEW.chapter\_num := CH\_LAST\_NUM + 1;  ELSE  :NEW.chapter\_num := 1;  END IF;  END IF;  END;  CREATE OR REPLACE TRIGGER CHECK\_CHAPTER\_DATE  BEFORE INSERT ON CHAPTERS  FOR EACH ROW  DECLARE  BOOK\_DATE DATE;  BEGIN  SELECT book\_publish\_date  INTO BOOK\_DATE  FROM BOOKS  WHERE BOOKS.book\_id = :new.book\_id;  IF :new.chapter\_publish\_date < BOOK\_DATE THEN  RAISE\_APPLICATION\_ERROR(-20001, 'wrong date');  END IF;  END;  CREATE OR REPLACE TRIGGER CHECK\_COMMENT\_DATE  BEFORE INSERT ON COMMENTS  FOR EACH ROW  DECLARE  CHAPTER\_DATE DATE;  BEGIN  SELECT chapter\_publish\_date  INTO CHAPTER\_DATE  FROM CHAPTERS  WHERE CHAPTERS.chapter\_id = :new.chapter\_id;  IF :new.comment\_publish\_date < CHAPTER\_DATE THEN  RAISE\_APPLICATION\_ERROR(-20001, 'wrong date');  END IF;  END;  CREATE OR REPLACE TRIGGER CHECK\_BOOKS\_GENRES  BEFORE INSERT ON BOOKS\_GENRES  FOR EACH ROW  DECLARE  NUM\_C NUMBER(10);  BEGIN  SELECT COUNT(\*)  INTO NUM\_C  FROM BOOKS\_GENRES  WHERE BOOKS\_GENRES.genre\_id = :new.genre\_id AND BOOKS\_GENRES.book\_id = :new.book\_id;  IF NUM\_C != 0 OR NUM\_C != NULL THEN  RAISE\_APPLICATION\_ERROR(-20001, 'ALREDY EXIST');  END IF;  END;  CREATE OR REPLACE TRIGGER CHECK\_BOOKS\_COLLECTIONS  BEFORE INSERT ON BOOKS\_COLLECTIONS  FOR EACH ROW  DECLARE  NUM\_C NUMBER(10);  BEGIN  SELECT COUNT(\*)  INTO NUM\_C  FROM BOOKS\_COLLECTIONS  WHERE BOOKS\_COLLECTIONS.collection\_id = :new.collection\_id AND BOOKS\_COLLECTIONS.book\_id = :new.book\_id;  IF NUM\_C != 0 OR NUM\_C != NULL THEN  RAISE\_APPLICATION\_ERROR(-20001, 'ALREDY EXIST');  END IF;  END; |